eleventh
international
conference
on
autonomous
agents and
multiagent
systems

AAMAS 2012

4th- 8th June 2012
Valencia

W4
Workshop on Adaptive and Learning Agents (ALA)

Proceedings of the Adaptive and Learning Agents Workshop 2012

June 4-5, 2012

Valencia, Spain

Editors:
Enda Howley, Peter Vrancx and Matt Knudson

Held in conjunction with the eleventh International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012

Preface

This book contains the papers accepted for presentation at the 2012 edition of the Adaptive and Learning Agents (ALA) workshop. ALA is the result of the merger of the ALAMAS and ALAg workshops. ALAMAS was an annual European workshop on Adaptive and Learning Agents and Multi-Agent Systems, held eight times. ALAg was the international workshop on Adaptive and Learning agents, typically held in conjunction with AAMAS. To increase the strength, visibility, and quality of the workshops, ALAMAS and ALAg were combined into the ALA workshop, and a steering committee was appointed to guide its development.

The goal of the workshop is to increase awareness and interest in adaptive agent research, encourage collaboration, and give a representative overview of current research in the area of adaptive and learning agents. It aims at bringing together not only different areas of computer science (e.g., agent architectures, reinforcement learning, and evolutionary algorithms) but also from different fields studying similar concepts (e.g., game theory, bio-inspired control, and mechanism design). The workshop serves as an inclusive forum for the discussion of ongoing or completed work in adaptive and learning agents and multi-agent systems.

Organizing an event such as ALA would not be possible without the efforts and contributions of many motivated people. We would like to thank all authors who responded to our call-for-papers. We expect that the workshop will be both lively and informative, and will aid participants in refining and further developing their research. We are also thankful to the members of the program committee for their high quality reviews, which ensured the strong scientific content of the workshop. Finally, we would like to thank the members of the ALA steering committee for their guidance, and the AAMAS conference for providing an excellent venue for our workshop.

Enda Howley, Peter Vrancx, and Matt Knudson
ALA 2012 Co-Chairs

Organization

Program Chairs

Enda Howley
National University of Ireland
Ireland
ehowley@nuigalway.ie

Peter Vrancx
Vrije Universiteit Brussel
Belgium
pvrancx@vub.ac.be

Matt Knudson
Carnegie Mellon University
USA
matt.knudson@gmail.com

Program Committee

Adrian Agogino, UCSC, NASA Ames Research Center, USA
Bikramjit Banerjee, The University of Southern Mississippi, USA
Enda Barrett, National University of Ireland, Ireland
Samuel Barrett, University of Texas at Austin, USA
Ana L.C. Bazzan, Universidade Federal do Rio Grande do Sul, Brazil
David Catteeuw, Vrije Universiteit Brussels, Belgium
Vincent Corruble, University of Paris 6, France
Yann-Michaël De Hauwere, Vrije Universiteit Brussels, Belgium
Sam Devlin, University of York, UK
Marek Grzes, University of Waterloo, Canada
Daniel Hennes, Maastricht University, The Netherlands
Todd Hester, University of Texas at Austin, USA
Chris Holmes Parker, Oregon State University, USA
Atil Iscen, Oregon State University, USA
Shivaram Kalyanakrishnan, Yahoo Labs Bangalore, India
Franziska Klügl, University of Orebro, Sweden
W. Bradley Knox, University of Texas at Austin, USA
Daniel Kudenko, University of York, UK
Mihail Mihailov, Vrije Universiteit Brussels, Belgium
Ann Nowé, Vrije Universiteit Brussels, Belgium
Liviu Panait, Google Inc Santa Monica, USA
Matthew Taylor, Lafayette College, USA
Karl Tuyls, Maastricht University, The Netherlands
Logan Yliniemi, Oregon State University, USA

Steering Committee

Franziska Klügl, University of Orebro, Sweden
Daniel Kudenko, University of York, UK
Ann Nowé, Vrije Universiteit Brussels, Belgium
Lynne E. Parker, University of Tulsa, USA
Sandip Sen, University of Tulsa, USA
Peter Stone, University of Texas at Austin, USA
Kagan Tumer, Oregon State University, USA
Karl Tuyls, Maastricht University, The Netherlands

Contents

Regulation of Social Exchanges in Open MAS: the problem of reciprocal conversions between POMDPs and HMMs

Graçaliz Pereira Dimuro, Antônio Carlos da Rocha Costa
PPGComp, PPGMC, Centro de Ciências Computacionais, Universidade Federal do Rio Grande - FURG
Campus Carreiros, 96200-970 Rio Grande, Brazil
{gracaliz,ac.rocha.costa}@gmail.com

ABSTRACT

An important problem in open multiagent systems is that of the regulation of social exchanges, towards producing social equilibrium. In the present work, we take an abstract and generalizing approach to this issue. The paper introduces a model for the sequential decision making of an agent, acting in an open partially observable stochastic environment, where that agent aims to induce another autonomous agent to interact in certain way, so as to lead both agents toward a target exchange state configuration. The regulation model introduced here is a combination of a Partially Observable Markov Decision Process (POMDP), to structure the regulator agent decision process, with a Hidden Markov Model (HMM), to structure its exchange strategy learning process. The main challenge we face is the conversions between the models, i.e.: How can an agent obtain a POMDP model from a newly learned HMM, so that it can find a regulation strategy for the newly learned exchange strategy of its partner? And, conversely: How can an agent obtain a good initial HMM model of its partner, from a POMDP only barely good to derive a policy to regulate a new exchange strategy of the latter, so as to accelerate the learning process of that new exchange strategy? This problem we call the problem of the reciprocal conversion between POMDPs and HMMs. The solution we have found builds on the particular structures of the POMDPs and HMMs that arise in the context of the regulation of social exchanges, which allow for the establishment of an isomorphism between the sets of states of the POMDPs and the sets of "extended" states of the HMMs. Such isomorphism forms the foundation, then, for the definition of the mappings that provide the reciprocal conversions between the POMDPs and the HMMs. The paper formally develops these ideas and shows their application to an example problem.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*

General Terms

Theory, Algorithms

Keywords

Social exchanges, social regulation, POMDPs, HMMs

1. INTRODUCTION

Systems of social relationships have often been seen as systems of social exchanges [11]. As extensively discussed in [20], a central problem in such systems of social exchanges is that of the regulation of the exchanges, towards producing social equilibrium. In particular, this is a difficult problem when the social system is an open one (that is, one where agents can enter and leave freely).

In the literature, there are various works dealing with different aspects of such regulation problem, in various Multiagent System (MAS) contexts (see, e.g., [12, 14, 21, 23]). In our previous works (e.g, [7, 9, 19]), we have evolved different models for the social exchange regulation problem, based on BDI (Beliefs, Desires, Intentions)[1] agents with plans derived from optimal policies of POMDPs (Partially Observable Markov Decision Process [16]). In particular, in [8], we outlined models for recognizing and learning BDI models of social exchange strategies for the regulation of social interactions in open agent societies.

In the present work, we take a more abstract and generalizing formal approach for the proposal introduced in [8]. Here, we introduce a model for the sequential decision making of an agent, α, acting in an open partially observable stochastic environment, where α aims to induce another autonomous agent, β, to interact in certain way, so as to lead both agents toward a certain configuration of exchange states, called the *target exchange state configuration*.

We call α the agent that is playing the role of the *regulator* of another agent β, which, in turn, has its own preferences on exchange states to visit, adopting a specific transition strategy for that. β is, then, said to be a *strategy-based* agent.

The system, thus, is composed by different strategy-based agents, some playing the role of regulator agents. Since the system is assumed to be open, agents with new exchange strategies may enter it, and the agents in the system may modify their strategies. Also, the agents may change their roles.

The regulation model introduced here is a combination of the POMDP structuring the regulator agent decision process as it was done in previous work, with a Hidden Markov Model (HMM) [18] to structure the *exchange strategy learning process*. The main challenge we face, then, is the conversion between the models, i.e.: How α can obtain a POMDP model from a newly learned HMM of β, so that α can find a regulation strategy for the newly learned strategy of β? And, conversely: How can α obtain a good initial HMM model of β, from a POMDP only approximately good to derive a policy to regulate a new strategy of β, so as to accelerate the learning process of the new exchange strategy of β?[2]

This problem we call the problem of the reciprocal conversion of POMDPs to HMMs. The problem arises from the fact that the POMDPs have state transition and observation functions based on the actions performed by the agents in each state, whereas, in the HMMs, the state transition and observation functions are not explicitly related to action performances.

[1]The models were developed for the Jason platform. [2]

[2]An introductory informal discussion about this conversions appeared in [8].

The solution we have found builds on the particular structures of the POMDPs and HMMs that arise in the context of the regulation of social exchanges: the HMMs involved in such problems are structured on the basis of certain "extended" states, and that allows for the establishment of an isomorphism between the sets of states of the POMDPs and the sets of "extended" states of the HMMs. Such isomorphism forms the foundation, then, for the definition of the mappings that provide the reciprocal conversions between the POMDPs and the HMMs.

In the literature, there exist other models for the problem of decision making of autonomous agents acting in the presence of other agents in uncertain environments (e.g.: MMDP, Markov Games, DEC-POMDP), as discussed by Gmytrasiewicz and Doshiby [13], which introduced Interactive POMDPs (I-POMDP) in 2005. I-POMDPs include the models of other agents in its state space, which is then composed by the beliefs about the physical environment and about the other agent(s), possibly recursively including others' beliefs about others, and so on. Differently to all these models, the POMDP for the exchange strategy regulation problem models directly the interactions between agents as exchange processes whose results constitute the state space and the agent exchange strategies are modeled as transition functions. The "environment" here is the actual context of the agent interactions.

The paper is organized as follows. Section 2 introduces the POMDP model for the strategy regulation problem. The HMM for the strategy learning problem is discussed in Sect. 3. In Sect. 4, we establish the formal relationship between POMDPs and HMMs, in terms of commutative diagrams, which allow the conversions between the models. In Sect. 5, we discuss the application of the proposed strategy regulation/learning model to the particular problem of regulating social exchanges in open MAS, giving some examples to clarify the formalism. Section 6 is the Conclusion.

2. THE POMDP FOR THE STRATEGY REGULATION PROBLEM

Consider two agents, a regulator agent α and a strategy-based agent β. Each agent has its own model of the states of the world. The sets of the states of the world, according to the points of view of the agents α and β, are given, respectively, by:

$$\mathrm{S}_\alpha = \{S^1_\alpha, \ldots, S^k_\alpha\} \text{ and } \mathrm{S}_\beta = \{S^1_\beta, \ldots, S^l_\beta\}. \tag{1}$$

In order to combine those two points of view, the states of the world for α's regulation process are modeled as ordered pairs $(S^*_\alpha, S^\dagger_\beta)$, with $S^*_\alpha \in \mathrm{S}_\alpha$ and $S^\dagger_\beta \in \mathrm{S}_\beta$, so the set of the states of the world is given by $\mathrm{S}_{\alpha\beta} = \mathrm{S}_\alpha \times \mathrm{S}_\beta$. For simplicity, a world state $(S^*_\alpha, S^\dagger_\beta) \in \mathrm{S}_{\alpha\beta}$ is denoted by $S^{*\dagger}_{\alpha\beta}$, with $* \in \{1, \ldots, k\}$ and $\dagger \in \{1, \ldots, l\}$. The target configuration of world states that α pursues is denoted by $(S^\theta_\alpha, S^\vartheta_\beta)$, where $1 \leq \theta \leq k$ and $1 \leq \vartheta \leq j$.

We assume that α always knows the current state of the world according to its own point of view (S^*_α), but it is not able to determine the current state of the world according to β's $(S^\dagger_\beta)$. Thus, α operates in a partially observable setting. We consider that the regulation process conduced by α on β suffers no influence from the regulation process occurring between other pairs of agents.

The *set of proposals* that the regulator agent α may make to a strategy-based agent β is given by $\mathrm{P} = \{p_1, \ldots, p_n\}$. β may adopt different *strategies* in reaction to α's proposals in each state, corresponding to probabilistic *state transition functions* of type:

$$T_\beta : \mathrm{S}_\beta \times \mathrm{P} \to \Pi(\mathrm{S}_\beta), \tag{2}$$

which specify, given the current state of the world according to β's point of view in S_β and α's proposal in P, a probability distribution over the set of states that β will achieve next.

According to β's reaction to α's proposal, the state of the world may change. Although α is assumed to have direct access only to its own world model (S^*_α), it is able to make observations on β's reactions/responses to its proposals. The *set of observations* about β's reactions/responses is given by $\Omega = \{\omega_1, \ldots, \omega_m\}$. The probabilistic *observation function*, based on the possible reactions/responses of β, is of type:

$$O_\beta : \mathrm{S}_\beta \times \mathrm{P} \to \Pi(\Omega), \tag{3}$$

which, given the (non-observable) current state of the world according to β's point of view in S_β and the α's proposal in P, gives a probability distribution over the set of possible observations Ω (i.e., the probability of β's reactions/responses to α's proposals).

Clearly, the regulation process can be model as a POMDP for the regulator agent α. Denote by $\mathrm{POMDP}_{\alpha\beta}$ the decision process on the best proposals α can do for an agent β in order to lead both toward the target world state configuration.

Considering α's model of the set of states of the world, and fixing a state S^*_α among the possible k values in S_α, a partitioning of the set of world states $\mathrm{S}_{\alpha\beta}$ is possible, obtaining the following sets of states, for $* \in \{1, \ldots, k\}$:

$$\mathrm{S}^*_{\alpha\beta} = \left\{ S^{*1}_{\alpha\beta}, \ldots, S^{*l}_{\alpha\beta} \right\}. \tag{4}$$

The partitioning of the set of states of the $\mathrm{POMDP}_{\alpha\beta}$ gives rise to k sub-POMDPs, one for each possible state of the world, according to α's point of view, denoted by $\mathrm{POMDP}^*_{\alpha\beta}$, whenever the current state of the world for α is S^*_α, with $* \in \{1, \ldots, k\}$.[3] Then, for each $* \in \{1, \ldots, k\}$, we define:

Definition 1. The $\mathrm{POMDP}^*_{\alpha\beta}$ model for the regulation process conduced by α on β, when the current state of the world is S^*_α (according to α's point of view) is defined as a tuple:

$$\mathrm{POMDP}^*_{\alpha\beta} = \left(\mathrm{S}^*_{\alpha\beta}, \mathrm{P}, T^*, \Omega, O^*, R^*\right), \text{ where:}$$

(i) $\mathrm{S}^*_{\alpha\beta}$ is the set of $\mathrm{POMDP}^*_{\alpha\beta}$ states, given by Eq. (4);

(ii) P is the set of proposals available for α to perform;

(iii) $T^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi\left(\mathrm{S}^*_{\alpha\beta}\right)$ is the state transition function, embedding the state transition function T_β of β's strategy model given in Eq. (2), such that, for all $p \in \mathrm{P}$, $S^\dagger_\beta, S^\ddagger_\beta \in \mathrm{S}_\beta$:

$$T^*\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*\ddagger}_{\alpha\beta}\right) = T_\beta\left(S^\dagger_\beta, p\right)\left(S^\ddagger_\beta\right); \tag{5}$$

(iv) Ω is the set of observations that may be realized by α about β's reactions/responses;

(v) $O^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi(\Omega)$ is the observation function, embedding the observation function O_β of β's strategy model given in Eq. (3), such that, for all $p \in \mathrm{P}$, $S^\dagger_\beta \in \mathrm{S}_\beta$ and $\omega \in \Omega$:

$$O^*\left(S^{*\dagger}_{\alpha\beta}, p\right)(\omega) = O_\beta\left(S^\dagger_\beta, p\right)(w); \tag{6}$$

(vi) $R^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \mathbb{R}$ is the reward function for the agent α, giving the expected immediate reward to be gained by α for each proposal in each state.

For each $* \in \{1, \ldots, k\}$, the solution of a $\mathrm{POMDP}^*_{\alpha\beta}$ is an *optimal policy* [16] for the agent α to follow when the current state of the world, according to its point of view, is S^*_α, guiding the agent α in the elaboration of proposals that may lead both agents toward the target state configuration of the world. Then, in fact, α follows a *non-stationary* optimal policy.[4]

[3] The restriction of the full POMDP allows to reduce the state space and to obtain directly the optimal proposal for each state of the world, according to α's point of view. So, when changing the state, α will have to change the policy, accordingly.

[4] In [8, 19] we have showed how to extract BDI plans for $\mathrm{POMDP}^*_{\alpha\beta}$ optimal policies.

3. THE HMM FOR THE STRATEGY LEARNING PROBLEM

If the regulator agent α is not able to recognize β's strategy[5], then α uses a mechanism based on HMMs in order to discover the state transition and observation functions that best fit the sequence of observations about β's reactions/responses to its proposals.

Denote by $\text{HMM}_{\alpha\beta}$ any HMM of α's strategy learning mechanism related to an agent β. Given a sequence of observations on β's reactions/responses and an arbitrary initial $\text{HMM}_{\alpha\beta}$, it is possible to apply the well-known Baum Welch algorithm [18] in order to optimize the $\text{HMM}_{\alpha\beta}$ for β's strategy model. Then, the state transition and observation functions of the optimized $\text{HMM}_{\alpha\beta}$ can be converted into the ones of a $\text{POMDP}_{\alpha\beta}$ (cf. Sect. 4), allowing the agent α to compute optimal policies for the new exchange strategy that the agent β is adopting.

Although $\text{HMM}_{\alpha\beta}$ and $\text{POMDP}_{\alpha\beta}$ models have many elements in common, the translation of the $\text{HMM}_{\alpha\beta}$ state transition and observation functions into the action-dependent $\text{POMDP}_{\alpha\beta}$ respective functions (and vice-versa) can not be done directly, for in the $\text{POMDP}_{\alpha\beta}$ model, the probability distribution of the set of states is directly linked to the kind of action performed by α in each state, whereas in the $\text{HMM}_{\alpha\beta}$ those distributions provide overall probabilistic values, independent of the action that might have been performed at each state.

However, in the specific case of the regulation problem, it is possible to relate the state transition matrices of both models, by constructing the state space of the $\text{HMM}_{\alpha\beta}$ in a way that it is possible to establish an isomorphism between the state spaces of both models. For that, we unify the $\text{POMDP}_{\alpha\beta}$ state transition matrices for all kinds of α's proposals into a single extended state transition matrix. Then, the states of the $\text{HMM}_{\alpha\beta}$ are extended to specify the kind of proposal that may be performed by α, obtaining the following set of states:

$$\begin{aligned} \text{SX}_{\alpha\beta} &= \text{S}_\alpha \times \text{S}_\beta \times \text{P} \qquad (7)\\ &= \{S^{*\dagger}_{\alpha\beta(p)} \mid * \in \{1,\ldots,k\}, \dagger \in \{1,\ldots,l\}, p \in \text{P}\}. \end{aligned}$$

In order to reduce the state space of the learning procedure, the set of extended states $\text{SX}_{\alpha\beta}$ is restricted to fixed values of the world states in S_α (according α's point of view). Then, for each $* \in \{1,\ldots,k\}$, we define:

$$\text{SX}^*_{\alpha\beta} = \{S^*_\alpha\} \times \text{S}_\beta \times \text{P} = \{S^{*\dagger}_{\alpha\beta(p)} \mid \dagger \in \{1,\ldots,l\}, p \in \text{P}\}. \quad (8)$$

Analogously to what was obtained in Sect. 2, the partitioning of the set of states of the $\text{HMM}_{\alpha\beta}$ gives rise to k sub-HMMs, one for each state of the world according to α, which we denote by $\text{HMM}^*_{\alpha\beta}$, whenever the current world state for α is S^*_α. Then, for each $* \in \{1,\ldots,k\}$, we define:

Definition 2. The $\text{HMM}^*_{\alpha\beta}$ of α's strategy learning mechanism, when the current world state for α is S^*_α, is defined as a tuple:

$$\text{HMM}^*_{\alpha\beta} = \left(\text{SX}^*_{\alpha\beta}, \Pi^0_{\text{SX}^*_{\alpha\beta}}, TX^*, \Omega, OX^*\right), \text{ where:}$$

(i) $\text{SX}^*_{\alpha\beta}$ is the set of extended states related to S^*_α, given by Eq. (8);

(ii) $\Pi^0_{\text{SX}^*_{\alpha\beta}}$ is the initial probability distribution of the set $\text{SX}^*_{\alpha\beta}$;

(iii) $TX^* : \text{SX}^*_{\alpha\beta} \to \Pi\left(\text{SX}^*_{\alpha\beta}\right)$ is the state transition function patterned on β's strategy model, which, for the current state in $\text{SX}^*_{\alpha\beta}$, gives a probability distribution over the set $\text{SX}^*_{\alpha\beta}$;

(iv) Ω is the set of observations that may be realized by α about β's reactions/responses;

(v) $OX^* : \text{SX}^*_{\alpha\beta} \to \Pi(\Omega)$ is the observation function patterned on β's strategy model, which, for the current state in $\text{SX}^*_{\alpha\beta}$, gives a probability distribution over the set Ω of observations.

In the following section, we establish the formal mappings between $\text{POMDP}_{\alpha\beta}$ and $\text{HMM}_{\alpha\beta}$ models that allow the conversions of a $\text{POMDP}_{\alpha\beta}$ model into a $\text{HMM}_{\alpha\beta}$ and vice-versa, on which the strategy learning process is based.

4. THE CONVERSION PROCEDURES

Let α be the agent playing the role of a regulator agent and $\text{POMDP}^*_{\alpha\beta}$ be the model related to a strategy-based agent β, for a given $* \in \{1,\ldots,k\}$. The following result, although almost immediate, is the key of the conversion processes between $\text{POMDP}^*_{\alpha\beta}$ and $\text{HMM}^*_{\alpha\beta}$ models:

PROPOSITION 1. *For all* $* \in \{1,\ldots,k\}$*, the set* $\text{S}^*_{\alpha\beta} \times \text{P}$ *is isomorphic to the set of extended states* $\text{SX}^*_{\alpha\beta}$.

PROOF. Consider, for any $* \in \{1,\ldots,k\}$, the function $f^* : \text{S}^*_{\alpha\beta} \times \text{P} \to \text{SX}^*_{\alpha\beta}$, defined, for all $S^{*\dagger}_{\alpha\beta} \in \text{S}^*_{\alpha\beta}$ and $p \in \text{P}$, by

$$f^*\left(S^{*\dagger}_{\alpha\beta}, p\right) = S^{*\dagger}_{\alpha\beta(p)}. \qquad (9)$$

The proof that f^* is well defined is straightforward, following directly from Eq. (7) and Eq. (8). It remains to show that f^* is bijective. For that, consider, for any $* \in \{1,\ldots,k\}$, the function $t^* : \text{SX}^*_{\alpha\beta} \to \text{S}^*_{\alpha\beta} \times \text{P}$, defined, for each $S^{*\dagger}_{\alpha\beta(p)} \in \text{SX}^*_{\alpha\beta}$, by

$$t^*\left(S^{*\dagger}_{\alpha\beta(p)}\right) = \left(S^{*\dagger}_{\alpha\beta}, p\right). \qquad (10)$$

It follows that $f^* \circ t^* = t^* \circ f^* = I$, where I is the identity function, and, thus, f^* has an inverse $(f^*)^{-1} = t^*$ and is bijective. □

In the following, consider any probability distributions over $\text{S}^*_{\alpha\beta}$ represented by

$$\pi\left(\text{S}^*_{\alpha\beta}\right) = \left\{\left(S^{*\dagger}_{\alpha\beta}, \pi^\dagger\right) | \dagger \in \{1,\ldots,l\}\right\} \in \Pi\left(\text{S}^*_{\alpha\beta}\right), \text{ such that}$$
$$\sum\left\{\pi^\dagger \mid \left(S^{*\dagger}_{\alpha\beta}, \pi^\dagger\right) \in \pi\left(\text{S}^*_{\alpha\beta}\right)\right\} = 1, \qquad (11)$$

where the probability of being at state $S^{*\dagger}_{\alpha\beta}$ is denoted by $\pi^\dagger$.

Analogously, represent a probability distributions over $\text{SX}^*_{\alpha\beta}$ by

$$\pi\left(\text{SX}^*_{\alpha\beta}\right) = \left\{\left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^\dagger_i\right) \mid \dagger \in \{1,\ldots,l\}, i \in \{1,\ldots,n\}\right\}$$
$$\in \Pi\left(\text{SX}^*_{\alpha\beta}\right), \text{ such that}$$
$$\sum\left\{\pi^\dagger_i \mid \left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^\dagger_i\right) \in \pi\left(\text{SX}^*_{\alpha\beta}\right)\right\} = 1, \qquad (12)$$

where the probability of being at state $S^{*\dagger}_{\alpha\beta(p_i)}$ is denoted by $\pi^\dagger_i$.

In the results bellow, we assume that the agent α performs, during the strategy learning process, a proposal $p_i \in \text{P} = \{p_1,\ldots,p_n\}$ with probability $\pi_{(p_i)}$, such that

$$\sum_{i=1}^{n} \pi_{(p_i)} = 1. \qquad (13)$$

PROPOSITION 2. *For all* $* \in \{1,\ldots,k\}$*, the set* $\Pi\left(\text{S}^*_{\alpha\beta}\right)$ *is isomorphic to the set* $\Pi\left(\text{SX}^*_{\alpha\beta}\right)$.

PROOF. For $\text{P}=\{p_1,\ldots,p_n\}$ and $*\in\{1,\ldots,k\}$, define the function $g^*_\text{P} : \Pi\left(\text{S}^*_{\alpha\beta}\right) \to \Pi\left(\text{SX}^*_{\alpha\beta}\right)$, for all $\pi\left(\text{S}^*_{\alpha\beta}\right) \in \Pi\left(\text{S}^*_{\alpha\beta}\right)$, by:

$$g^*_\text{P}\left(\pi\left(\text{S}^*_{\alpha\beta}\right)\right) = g^*_\text{P}\left(\left\{\left(S^{*\dagger}_{\alpha\beta}, \pi^\dagger\right) \mid \dagger \in \{1,\ldots,l\}\right\}\right) \qquad (14)$$
$$= \left\{\left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^\dagger \cdot \pi_{(p_i)}\right) \mid \dagger \in \{1,\ldots,l\}, i \in \{1,\ldots,n\}\right\},$$

[5]It is supposed that the regulator agent is able to recognize exchange strategy models that it has already learned. However, for the lack of space, we do not discuss the recognition problem in this paper, since it is not the objective here. See, e.g., [8].

where $\pi_{(p_i)}$ is the probability of the agent α to perform a proposal $p_i \in \mathrm{P}$ in the state $S^{*\dagger}_{\alpha\beta}$. By Eq. (11) and Eq. (13), it follows that

$$\sum \left\{ \pi^{\dagger} \cdot \pi_{(p_i)} \mid \left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^{\dagger} \cdot \pi_{(p_i)} \right) \in \pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right) \right\} = 1$$

and g^{*}_{P} is well defined. It remains to show that g^{*}_{P} is bijective. Define, for $* \in \{1, \ldots, k\}$ and $\mathrm{P} = \{p_1, \ldots, p_n\}$, the function $h^{*}_{\mathrm{P}} : \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right) \rightarrow \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right)$, so that, for all $\pi\left(\mathrm{S}^{*}_{\alpha\beta(p_i)}\right) \in \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$:

$$\begin{aligned} & h^{*}_{\mathrm{P}}\left(\pi\left(\mathrm{S}^{*}_{\alpha\beta(p_i)}\right)\right) \qquad (15) \\ & = h^{*}_{\mathrm{P}}\left(\left\{ \left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^{\dagger}_{i} \right) \mid \dagger \in \{1, \ldots, l\}, i \in \{1, \ldots, n\} \right\}\right) \\ & = \left\{ \left(S^{*\dagger}_{\alpha\beta}, \sum_{i=1}^{n} \pi^{\dagger}_{i} \right) \mid \dagger \in \{1, \ldots, l\} \right\}. \end{aligned}$$

It follows that h^{*}_{P} is well defined, since $\pi^{\dagger}_{i} = \pi^{\dagger} \cdot \pi_{(p_i)}$ and $\sum_{\dagger=1}^{l} \sum_{i=1}^{n} \pi^{\dagger}_{i} = 1$. One has that $g^{*}_{\mathrm{P}} \circ h^{*}_{\mathrm{P}} = h^{*}_{\mathrm{P}} \circ g^{*}_{\mathrm{P}} = I$, where I is the identity function, and, therefore, there exists the inverse $\left(g^{*}_{\mathrm{P}}\right)^{(-1)} = h^{*}_{\mathrm{P}}$, and g^{*}_{P} is bijective. □

The isomorphisms stated in Prop. 1 and Prop. 2 guarantee that the conversions between $\mathrm{HMM}^{*}_{\alpha\beta}$ and $\mathrm{POMDP}^{*}_{\alpha\beta}$ models, in both directions, are viable. The following result shows the relation between state transition functions.

PROPOSITION 3. *Consider the function $f^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \mathrm{SX}^{*}_{\alpha\beta}$, defined in Eq. (9), and its inverse $\left(f^{*}\right)^{(-1)} : \mathrm{SX}^{*}_{\alpha\beta} \rightarrow \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P}$, given in Eq. (10). Consider the function $g^{*}_{\mathrm{P}} : \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right) \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$, given in Eq. (14), and its inverse $\left(g^{*}_{\mathrm{P}}\right)^{(-1)} : \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right) \rightarrow \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right)$, given in Eq. (15). Let $T^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right)$ and $TX^{*} : \mathrm{SX}^{*}_{\alpha\beta} \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$ be the state transition functions of the $\mathrm{POMDP}^{*}_{\alpha\beta}$ and the $\mathrm{HMM}^{*}_{\alpha\beta}$, defined, respectively in Eq. (5) and Def. 2 (iii). Then the diagram of Fig. 1(a) commutes.*

PROOF. Consider the function $g^{*}_{\mathrm{P}} \circ T^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$. Given $\left(S^{*\dagger'}_{\alpha\beta}, p'\right) \in \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P}$, one has that

$$\begin{aligned} \left(g^{*}_{\mathrm{P}} \circ T^{*}\right)\left(S^{*\dagger'}_{\alpha\beta}, p'\right) &= g^{*}_{\mathrm{P}}\left(T^{*}\left(S^{*\dagger'}_{\alpha\beta}, p'\right)\right) \\ &= g^{*}_{\mathrm{P}}\left(\left\{\left(S^{*\dagger}_{\alpha\beta}, \pi^{\dagger}\right) \mid \dagger \in \{1, \ldots, l\}\right\}\right) \\ &= \left\{\left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^{\dagger} \cdot \pi_{(p_i)}\right) \mid \dagger \in \{1, \ldots, l\}, i \in \{1, \ldots, n\}\right\}. \end{aligned}$$

Now, considering the function $TX^{*} \circ f^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$ and given $\left(S^{*\dagger'}_{\alpha\beta}, p'\right) \in \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P}$, one has that

$$\begin{aligned} \left(TX^{*} \circ f^{*}\right)\left(S^{*\dagger'}_{\alpha\beta}, p'\right) &= TX^{*}\left(f^{*}\left(S^{*\dagger'}_{\alpha\beta}, p'\right)\right) = TX^{*}\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \\ &= \left\{\left(S^{*\dagger}_{\alpha\beta(p_i)}, \pi^{\dagger}_{i}\right) \mid \dagger \in \{1, \ldots, l\}, i \in \{1, \ldots, n\}\right\}. \end{aligned}$$

It follows that $\pi^{\dagger}_{i} = \pi^{\dagger} \cdot \pi_{(p_i)}$, and, therefore $g^{*}_{\mathrm{P}} \circ T^{*} = TX^{*} \circ f^{*}$. On the other hand, one has that $\left(g^{*}_{\mathrm{P}}\right)^{(-1)} \circ g^{*}_{\mathrm{P}} \circ T^{*} \circ \left(f^{*}\right)^{(-1)} = \left(g^{*}_{\mathrm{P}}\right)^{(-1)} \circ TX^{*} \circ f^{*} \circ \left(f^{*}\right)^{(-1)}$, that is, $T^{*} \circ \left(f^{*}\right)^{(-1)} = \left(g^{*}_{\mathrm{P}}\right)^{(-1)} \circ TX^{*}$, which shows that the diagram of Fig. 1(a) commutes. □

The commutativity of the diagram of Fig. 1(a) allows to obtain the state transition function of a $\mathrm{HMM}^{*}_{\alpha\beta}$ from the one of a $\mathrm{POMDP}^{*}_{\alpha\beta}$, and vice-versa, as stated in the following theorems:

THEOREM 4. *The state transition function $TX^{*} : \mathrm{SX}^{*}_{\alpha\beta} \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$ of the $\mathrm{HMM}^{*}_{\alpha\beta}$ of α's strategy learning mechanism, when the current world state for α is S^{*}_{α}, can be obtained from the state transition function $T^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right)$ of the related $\mathrm{POMD}^{*}_{\alpha\beta}$, defining, for each $S^{*\dagger}_{\alpha\beta(p)}, S^{*\dagger'}_{\alpha\beta(p')} \in \mathrm{SX}^{*}_{\alpha\beta}$:*

$$TX^{*}\left(S^{*\dagger}_{\alpha\beta(p)}\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) = \pi_{(p')} \cdot T^{*}\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \qquad (16)$$

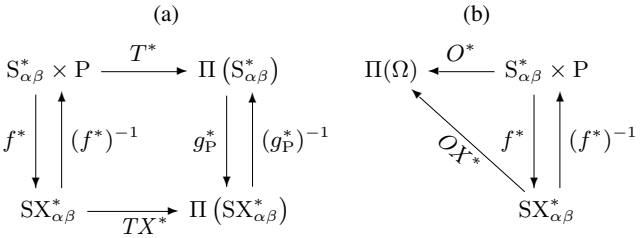


Figure 1: The relationship between $\mathrm{POMDP}^{*}_{\alpha\beta}$ and $\mathrm{HMM}^{*}_{\alpha\beta}$ (a) state transition and (b) observation functions

where $\dagger, \dagger' \in \{1, \ldots, l\}$, $S^{*\dagger}_{\alpha\beta}, S^{*\dagger'}_{\alpha\beta} \in \mathrm{S}^{*}_{\alpha\beta}$, $p, p' \in \mathrm{P}$, $\mathrm{S}^{*}_{\alpha\beta}, T^{*}, \mathrm{P}$ are as defined in Def. 1 and $\mathrm{SX}^{*}_{\alpha\beta}$, TX^{*} are as defined in Def. 2.

PROOF. Consider the functions $f^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \mathrm{SX}^{*}_{\alpha\beta}$, given in Eq. (9), and its inverse $\left(f^{*}\right)^{-1} : \mathrm{SX}^{*}_{\alpha\beta} \rightarrow \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P}$, defined in Eq. (10). Then, it follows that $TX^{*} \circ f^{*} \circ \left(f^{*}\right)^{-1} = TX^{*}$, and, from the commutativity of the diagram of Fig. 1(a), one has that $TX^{*} = g^{*}_{\mathrm{P}} \circ T^{*} \circ \left(f^{*}\right)^{-1}$, where $g^{*}_{\mathrm{P}} : \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right) \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$ is defined by Eq. (14). Thus, for $S^{*\dagger}_{\alpha\beta(p)}, S^{*\dagger'}_{\alpha\beta(p')} \in \mathrm{SX}^{*}_{\alpha\beta}$:

$$\begin{aligned} & TX^{*}\left(S^{*\dagger}_{\alpha\beta(p)}\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \\ &= \left(\left(g^{*}_{\mathrm{P}} \circ T^{*} \circ \left(f^{*}\right)^{-1}\right)\left(S^{*\dagger}_{\alpha\beta(p)}\right)\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \\ &= \left(\left(g^{*}_{\mathrm{P}} \circ T^{*}\right)\left(\left(f^{*}\right)^{-1}\left(S^{*\dagger}_{\alpha\beta(p)}\right)\right)\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \\ &= \left(\left(g^{*}_{\mathrm{P}} \circ T^{*}\right)\left(S^{*\dagger}_{\alpha\beta}, p\right)\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \text{ by Eq. (10)} \\ &= \left(g^{*}_{\mathrm{P}}\left(T^{*}\left(S^{*\dagger}_{\alpha\beta}, p\right)\right)\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \\ &= \left(g^{*}_{\mathrm{P}}\left(\left\{\left(S^{*\dagger''}_{\alpha\beta}, T^{*}\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*''}_{\alpha\beta}\right)\right)\right.\right.\right. \\ &\qquad \left.\left.\left. \mid \dagger'' \in \{1, \ldots, l\}\right\}\right)\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \text{ by Def. 1 (iii)} \\ &= \left(\left\{\left(S^{*\dagger''}_{\alpha\beta(p'')}, \pi_{(p'')} \cdot T^{*}\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*''}_{\alpha\beta}\right)\right)\right.\right. \\ &\qquad \left.\left. \mid \dagger'' \in \{1, \ldots, l\}, p'' \in \mathrm{P}\right\}\right)\left(S^{*\dagger'}_{\alpha\beta(p')}\right) \text{ by Eq. (14)} \\ &= \pi_{(p')} \cdot T^{*}\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*\dagger'}_{\alpha\beta}\right), \end{aligned}$$

which proves the result. □

THEOREM 5. *The state transition function $T^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right)$ of α's $\mathrm{POMDP}^{*}_{\alpha\beta}$ for some strategy-based agent β, when the current world state for α is S^{*}_{α}, can be obtained from the state transition function $TX^{*} : \mathrm{SX}^{*}_{\alpha\beta} \rightarrow \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right)$ of α's $\mathrm{HMM}^{*}_{\alpha\beta}$ strategy learning model, defining, for $S^{*\dagger}_{\alpha\beta}, S^{*\dagger'}_{\alpha\beta} \in \mathrm{S}^{*}_{\alpha\beta}$ and $p \in \mathrm{P}$:*

$$T^{*}\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*\dagger'}_{\alpha\beta}\right) = \sum_{i=1}^{n} TX^{*}\left(S^{*\dagger}_{\alpha\beta(p)}\right)\left(S^{*\dagger'}_{\alpha\beta(p_i)}\right) \qquad (17)$$

where $\dagger, \dagger' \in \{1, \ldots, l\}$, $S^{*\dagger}_{\alpha\beta(p)}, S^{*\dagger'}_{\alpha\beta(p_i)} \in \mathrm{SX}^{*}_{\alpha\beta}$, $p_i \in \mathrm{P}$, $\mathrm{S}^{*}_{\alpha\beta}$, T^{*}, P are defined in Def. 1 and $\mathrm{SX}^{*}_{\alpha\beta}$, TX^{*} are defined in Def. 2.

PROOF. Analogously to the proof of Theorem 4, from the commutativity of the diagram of Fig. 1(a), it holds that $T^{*} = \left(g^{*}_{\mathrm{P}}\right)^{(-1)} \circ TX^{*} \circ f^{*}$, where $\left(g^{*}_{\mathrm{P}}\right)^{(-1)} : \Pi\left(\mathrm{SX}^{*}_{\alpha\beta}\right) \rightarrow \Pi\left(\mathrm{S}^{*}_{\alpha\beta}\right)$ is defined by Eq. (15) and $f^{*} : \mathrm{S}^{*}_{\alpha\beta} \times \mathrm{P} \rightarrow \mathrm{SX}^{*}_{\alpha\beta}$ is given in Eq. (9). Thus, for $S^{*\dagger}_{\alpha\beta}, S^{*\dagger'}_{\alpha\beta} \in \mathrm{S}^{*}_{\alpha\beta}$ and $p \in \mathrm{P}$, one has that:

$$
\begin{aligned}
T^*\left(S^{*\dagger}_{\alpha\beta}, p\right)\left(S^{*\dagger'}_{\alpha\beta}\right) &= \left(\left((g^*_{\mathrm{P}})^{(-1)} \circ TX^* \circ f^*\right)\left(S^{*\dagger}_{\alpha\beta}, p\right)\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \\
&= \left(\left((g^*_{\mathrm{P}})^{(-1)} \circ TX^*\right)\left(f^*\left(S^{*\dagger}_{\alpha\beta}, p\right)\right)\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \\
&= \left(\left((g^*_{\mathrm{P}})^{(-1)} \circ TX^*\right)\left(S^{*\dagger}_{\alpha\beta(p)}\right)\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \text{ by Eq. (9)} \\
&= \left((g^*_{\mathrm{P}})^{(-1)}\left(TX^*\left(S^{*\dagger}_{\alpha\beta(p)}\right)\right)\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \\
&= \left((g^*_{\mathrm{P}})^{(-1)}\left(\left\{\left(S^{*\dagger''}_{\alpha\beta(p_i)}, TX^*\left(S^{*\dagger}_{\alpha\beta(p)}\right)\left(S^{*''}_{\alpha\beta(p_i)}\right)\right)\right.\right. \right. \\
&\quad \left.\left.\left. \mid \dagger'' \in \{1,\ldots,l\}, i \in \{1,\ldots,n\}\right\}\right)\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \text{ by Def. 2 (iii)} \\
&= \left(\left\{\left(S^{*\dagger''}_{\alpha\beta}, \sum_{i=1}^{n} TX^*\left(S^{*\dagger}_{\alpha\beta(p)}\right)\left(S^{*\dagger''}_{\alpha\beta(p_i)}\right)\right)\right.\right. \\
&\quad \left.\left. \mid \dagger'' \in \{1,\ldots,l\}\right\}\right)\left(S^{*\dagger'}_{\alpha\beta}\right) \text{ by Eq. (15)} \\
&= \sum_{i=1}^{n} TX^*\left(S^{*\dagger}_{\alpha\beta(p)}\right)\left(S^{*\dagger'}_{\alpha\beta(p_i)}\right),
\end{aligned}
$$

which proves the result. □

In the following, consider any probability distributions over the set of observations Ω represented by

$$\pi(\Omega) = \left\{\left(\omega, \pi_{(\omega)}\right) \mid \omega \in \Omega\right\} \in \Pi(\Omega),$$

such that $\sum\left\{\pi_{(\omega)} \mid \left(\omega, \pi_{(\omega)}\right) \in \pi(\Omega)\right\} = 1$, where the probability of an observation ω is denoted by $\pi_{(\omega)}$.

The following result shows the relation between $\mathrm{HMM}^*_{\alpha\beta}$ observation functions and $\mathrm{POMDP}^*_{\alpha\beta}$ observation functions.

PROPOSITION 6. *Let $f^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \mathrm{SX}^*_{\alpha\beta}$ and its inverse $(f^*)^{(-1)} : \mathrm{SX}^*_{\alpha\beta} \to \mathrm{S}^*_{\alpha\beta} \times \mathrm{P}$ be given in Eqs. (9) and (10), respectively. Let $O^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi(\Omega)$ (Eq. (5)) and $OX^* : \mathrm{SX}^*_{\alpha\beta} \to \Pi(\Omega)$ (Def. 2(v)) be the $\mathrm{POMDP}^*_{\alpha\beta}$ and $\mathrm{HMM}^*_{\alpha\beta}$ observation functions, respectively. Then the diagram of Fig. 1(b) commutes.*

PROOF. Considering the function $OX^* \circ f^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi(\Omega)$ and given $\left(S^{*\dagger}_{\alpha\beta}, p\right) \in \mathrm{S}^*_{\alpha\beta} \times \mathrm{P}$, one has that

$$
\begin{aligned}
(OX^* \circ f^*)\left(S^{*\dagger}_{\alpha\beta}, p\right) &= OX^*\left(f^*\left(S^{*\dagger}_{\alpha\beta}, p\right)\right) = OX^*\left(S^{*\dagger}_{\alpha\beta(p)}\right) \\
&= \left\{\left(\omega, \pi_{(\omega)}\right) \mid \omega \in \Omega\right\} = O^*\left(S^{*\dagger}_{\alpha\beta}, p\right),
\end{aligned}
$$

that is, $O^* = OX^* \circ f^*$. It follows obviously that $OX^* = O^* \circ (f^*)^{(-1)}$, showing that the diagram of Fig. 1(b) commutes. □

The following theorem shows that it is possible to obtain a $\mathrm{HMM}^*_{\alpha\beta}$ observation function from the one of a $\mathrm{POMDP}^*_{\alpha\beta}$, and vice-versa.

THEOREM 7. *The observation functions $OX^* : \mathrm{SX}^*_{\alpha\beta} \to \Pi(\Omega)$ and $O^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi(\Omega)$ of, respectively, the $\mathrm{HMM}^*_{\alpha\beta}$ of α's strategy learning mechanism and the $\mathrm{POMD}^*_{\alpha\beta}$ of α's regulation mechanism, when the current world state for α is S^*_α, can be obtained from each other, defining, for each $S^{*\dagger}_{\alpha\beta(p)} \in \mathrm{SX}^*_{\alpha\beta}$, $S^{*\dagger}_{\alpha\beta} \in \mathrm{S}^*_{\alpha\beta}$, $p \in \mathrm{P}$ and $\omega \in \Omega$:*

$$OX^*\left(S^{*\dagger}_{\alpha\beta(p)}\right)(\omega) = O^*\left(S^{*\dagger}_{\alpha\beta}, p\right)(\omega) \tag{18}$$

where $\dagger \in \{1,\ldots,l\}$, $\mathrm{S}^*_{\alpha\beta}$, O^*, P, Ω are as defined in Def. 1 and $\mathrm{SX}^*_{\alpha\beta}$, OX^* are as defined in Def. 2.

PROOF. From the commutativity of the diagram of Fig. 1(b), one has that $OX^* = O^* \circ (f^*)^{(-1)}$, where $(f^*)^{-1} : \mathrm{SX}^*_{\alpha\beta} \to \mathrm{S}^*_{\alpha\beta} \times \mathrm{P}$ is defined in Eq. (10). For $S^{*\dagger}_{\alpha\beta(p)} \in \mathrm{SX}^*_{\alpha\beta}$ and $\omega \in \Omega$:

$$
\begin{aligned}
OX^* S^{*\dagger}_{\alpha\beta(p)}(\omega) &= \left(O^* \circ (f^*)^{-1}\right)\left(S^{*\dagger}_{\alpha\beta(p)}\right)(\omega) \\
&= O^*\left((f^*)^{-1}\left(S^{*\dagger}_{\alpha\beta(p)}\right)\right)(\omega) = O^*\left(S^{*\dagger}_{\alpha\beta}, p\right)(\omega),
\end{aligned}
$$

by Eq. (10), which proves the result. □

5. REGULATING SERVICE EXCHANGES IN OPEN MAS

In this section, we discuss the application of the POMDP/HMM conversion model to the particular problem of regulating social exchanges in open multiagent systems.

The modeling of agent social interactions based on the evaluation of social exchanges has been considered in different works and contexts, e.g., by Rodrigues, Costa and Bordini [22], Rodrigues [21], Grimaldo et al. [14], Costa and Dimuro [4, 5], Dimuro et al. [7, 9, 8], Pereira et al. [19], Franco, Costa and Coelho [12].

In particular, Rodrigues and Luck (e.g., [21, 23]) introduced a rich approach based on the Theory of Social Exchanges for the modeling of interactions in open multiagent systems, presenting a system for analysing/evaluating partner selection and cooperative interactions in the Bioinformatics domain, which is characterized by frequent, extensive and dynamic social exchanges. On the other hand, our previous works (e.g., [7, 9, 19]) were concerned with the regulation of personality-based social exchanges in MAS, obtained by hybrid models based on POMDPs and BDI (Beliefs, Desires, Intentions) agents. The specific problem of recognizing and learning social exchange strategies was informally discussed in [8]. Here, we adopt a simplified model of non-economic social exchanges between two agents α and β, considered as an exchange of services between the agents, inspired on the approaches of social exchanges in the literature (e.g, [1, 3, 11, 15, 20]).

For simplicity, let $\mathrm{P} = \{offer\_service, request\_service\}$ be the *set of service proposals* that the agents may opt to do to each other in a social exchange. When a service proposal done by an agent α (β) is accepted by an agent β (α), a *service exchange stage* happens. If, in this stage, it is the case that the agent α (β) realizes a service for β (α), then α (β) associates to it a qualitative cost value, which is the value of the investment done by α (β) for the realization of the service for β (α). On the other hand, the agent β (α) evaluates the qualitative benefit value for receiving the service done by α.

Let α be the regulator agent that performs service proposals, and let β be the strategy-based agent that has to decide on accepting or refusing α's proposals according to some strategy. For example, α may offer/request a specific service for/from β, and β may accept/reject α's proposal depending on the service offered/requested. The set of β's responses to α's service proposals is given by $\Omega = \{A, R\}$, where A and R mean that the agent β accepts and refuses the service exchange proposal, respectively.

Observe that an agent has to assume the role of α (that is, it has to play the role of the regulator agent that makes service proposals) if it wants/needs to regulate service exchange processes with another agent (which then assumes the role of the strategy-based agent β).

A *service exchange process* is a sequence of service exchange stages. At any time, the *service exchange result* can be evaluated according to the different points of view of α and β, as the accumulated "sum" of their respective cost and benefit values in all stages that occurred until this time. Since the values are usually of a qualitative nature [20], it is supposed that there exists a kind of qualitative algebra where the "sum" operation is available. Here, we represent the qualitative values by intervals, adopting the interval algebra proposed in [7, 9].

The service exchange results are classified according to their position relatively to a given reference point θ. Let S^θ_ε be an interval of service exchange results ρ that are around the reference point θ, such that $\mid \rho - \theta \mid \leq \varepsilon$, with $\varepsilon \geq 0$, that is, $S^\theta_\varepsilon = [\theta - \varepsilon, \theta + \varepsilon]$. In the following, we assume that the value of ε is known, so we simplify the notation to S^θ.

Any service exchange result ρ' such that $\rho' > \theta + \varepsilon$ is called a *favorable* result. Suppose that there is a maximal favorable result $\rho_{\max}$ that an agent can afford. Then it is possible to distinguish f different ranges of favorable results, denoted by $S^{+1}, \ldots, S^{+f}$. For simplicity, consider $f = 1$, so that all favorable result are contained in the interval $S^{+} = (\theta + \varepsilon, \rho_{\max}]$.

Similarly, a service exchange result ρ'' such that $\rho'' < \theta - \varepsilon$ is called an *unfavorable* result. Considering that there is a minimal unfavorable result $\rho_{\min}$ that an agent can afford, then it is possible to consider all unfavorable results contained in $S^{-} = [\rho_{\min}, \theta - \varepsilon)$.

The possible ranges of results of service exchange processes, evaluated differently according to each agent's points of view, constitute the states of the world for each agent. The *sets of the world states* for the agents α and β are given by $\mathrm{S}_\alpha = \left\{S_\alpha^-, S_\alpha^\theta, S_\alpha^+\right\}$ and $\mathrm{S}_\beta = \left\{S_\beta^-, S_\beta^\theta, S_\beta^+\right\}$, the set of states of the POMDP$_{\alpha\beta}$ of α's strategy regulation model is $\mathrm{S}_{\alpha\beta} = \left\{S_{\alpha\beta}^{*\dagger} \mid *, \dagger \in \{-, \theta, +\}\right\}$, and the target state aimed by α is $S_{\alpha\beta}^{\theta\theta}$.

A strategy-based agent β may adopt different *service exchange strategies* during its service exchanges with the regulator agent α, corresponding to different state transition functions, modeled by Eq. (2). For example, an agent adopting a strategy that can be characterized as "egoistic" is mostly seeking its own benefit, with a very high probability to accept service exchanges that represent *transitions to favorable results* (i.e., exchanges in which the other agent performs any kind of valuable service to it).

EXAMPLE 1. *Consider the so-called* egoism-80 *exchange strategy model of an agent β, whose probabilities of accepting service exchange offerings and refusing service exchange requests are around 80%, that is, the agent looks for its own benefit in 80% of the services exchanges. The POMDP$^*_{\alpha\beta}$ state transition and observation functions T^* and O^*, for a given $* \in \{-, \theta, +\}$, are presented in Table 1 and 2, respectively.*

Observe, in Table 1(a), that if β is currently with favorable results (state $S_{\alpha\beta}^{*+}$) then there is no possibility for it to decrease its material results with an offer_service proposal by α. Conversely, in Table 1(b), it is possible to observe that β is not able to increase its material results with a request_service proposal by α, whenever β is with unfavorable results (state $S_{\alpha\beta}^{*-}$).

The following example shows how to obtain HMM$^*_{\alpha\beta}$ from the POMDP$^*_{\alpha\beta}$ of the *egoism-80* exchange strategy model of Ex. 1.

EXAMPLE 2. *Consider that the regulator agent α, for the learning process, uses the following probability distribution over the set* P *of service exchanges proposals: $\pi_{(offer\_service)} = 0.6$ and $\pi_{(request\_service)} = 0.4$. Also, consider that α takes the* egoism-80 *strategy model of Ex. 1 as reference for the learning process.*

Figure 2 illustrates the conversion process stated by Theorem 4. On the top of Fig. 2, observe part of the POMDP$^*_{\alpha\beta}$ state transition function $T^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi\left(\mathrm{S}^*_{\alpha\beta}\right)$ for the offer_service proposal (Table 1(a)). The corresponding part of the function T^* for the request_service proposal (Table 1(b)) is on the bottom of the figure. The dot lines show how to split the POMDP$^*_{\alpha\beta}$ states into HMM$^*_{\alpha\beta}$ states, according to the two types of service exchange proposals. Then, in the middle of the figure, the corresponding resulting part of the HMM$^*_{\alpha\beta}$ state transition function $TX^* : \mathrm{SX}^*_{\alpha\beta} \to \Pi\left(\mathrm{SX}^*_{\alpha\beta}\right)$ is shown. One can observe, for example, that

$$TX^*\left(S^{*\theta}_{\alpha\beta(offer)}\right)\left(S^{*+}_{\alpha\beta(request)}\right)$$
$$= \pi_{(request)} \cdot T^*\left(S^{*\theta}_{\alpha\beta}, \mathit{offer\_service}\right)\left(S^{*+}_{\alpha\beta}\right) = 0.4 \cdot 0.8 = 0.32.$$

The function TX^* is fully presented in Table 7. Applying Theorem 7, the POMDP$^*_{\alpha\beta}$ observation function $O^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi(\Omega)$ (Table 2(a) and (b)) can be converted into the HMM$^*_{\alpha\beta}$ observation function $OX^* : \mathrm{SX}^*_{\alpha\beta} \to \Pi(\Omega)$, obtaining the definition shown in Table 3. Observe that Table 3 is obtained by interleaving Table 2(a) and Table 2(b) accordingly when splitting the set $\mathrm{S}^*_{\alpha\beta}$ into the set $\mathrm{SX}^*_{\alpha\beta}$ considering the two kinds of service proposals.

Finally, consider that the service exchange process for the learning process starts at the state $S^{++}_{\alpha\beta}$. Then , $* = +$ (α's current state) and the initial probability distribution $\Pi^0_{\mathrm{SX}_{\alpha\beta}}$ of the extended set of states is as follows: $\Pi^0_{\mathrm{SX}_{\alpha\beta}}\left(S^{*+}_{\alpha\beta(offer)}\right) = 0.60$, $\Pi^0_{\mathrm{SX}_{\alpha\beta}}\left(S^{*+}_{\alpha\beta(request)}\right) = 0,40$, and $\Pi^0_{\mathrm{SX}_{\alpha\beta}}\left(S^{*\dagger}_{\alpha\beta(p)}\right) = 0.00$, for any $S^{*\dagger}_{\alpha\beta(p)} \neq S^{*+}_{\alpha\beta(offer)}, S^{*+}_{\alpha\beta(request)}$. $\Pi^0_{\mathrm{SX}_{\alpha\beta}}$, together with the functions TX^* (Table 7) and OX^* (Table 3), constitute the input data for the learning process, for which one can use, for example, the well-knowm Baum Welch algorithm [18].

The following example shows how to obtain a POMDP$^*_{\alpha\beta}$ regulation model from a HMM$^*_{\alpha\beta}$ that was specialized for a service exchange strategy model, which was previously unknown by the regulator agent α, by applying the results of Theorems 5 and 7.

EXAMPLE 3. *Considering the input HMM$^*_{\alpha\beta}$ constructed in Ex. 2, and a sequence of observations done by α when performing service exchange proposals, the application of the Baum Welch algorithm produces the optimized HMM$^*_{\alpha\beta}$ specified by the state transition and observation functions given in Table 8 and 4, respectively.*

Figure 3 illustrates the conversion process stated by Theorem 5. On the middle of Fig. 3, observe part of the HMM$^*_{\alpha\beta}$ state transition function $TX^* : \mathrm{SX}^*_{\alpha\beta} \to \Pi\left(\mathrm{SX}^*_{\alpha\beta}\right)$, extracted from Table 8. The dot lines show how to unify two HMM$^*_{\alpha\beta}$ states into

Table 1: The POMDP$^*_{\alpha\beta}$ state transition function T^* of an egoism-80 service exchange strategy model

	(a) *offer_service*			(b) *request_service*		
$\Pi\left(\mathrm{S}^*_{\alpha\beta}\right)$	$S^{*\theta}_{\alpha\beta}$	$S^{*+}_{\alpha\beta}$	$S^{*-}_{\alpha\beta}$	$S^{*\theta}_{\alpha\beta}$	$S^{*+}_{\alpha\beta}$	$S^{*-}_{\alpha\beta}$
$S^{*\theta}_{\alpha\beta}$	0.20	0.80	0.00	0.80	0.00	0.20
$S^{*+}_{\alpha\beta}$	0.00	1.00	0.00	0.12	0.80	0.08
$S^{*-}_{\alpha\beta}$	0.50	0.30	0.20	0.00	0.00	1.00

Table 2: The POMDP$^*_{\alpha\beta}$ observation function O^* of an egoism-80 service exchange strategy model

	(a) *offer_service*		(b) *request_service*	
$\Pi(\Omega)$	A	R	A	R
$S^{*\theta}_{\alpha\beta}$	0.80	0.20	0.20	0.80
$S^{*+}_{\alpha\beta}$	0.75	0.25	0.25	0.75
$s^{*-}_{\alpha\beta}$	0.85	0.15	0.15	0.85

Table 3: The HMM$^*_{\alpha\beta}$ observation function OX^* obtained from an egoism-80 exchange strategy model

$\Pi(\Omega)$	A	R	$\Pi(\Omega)$	A	R
$S^{*\theta}_{\alpha\beta(offer)}$	0.80	0.20	$S^{*\theta}_{\alpha\beta(request)}$	0.20	0.80
$S^{*+}_{\alpha\beta(offer)}$	0.75	0.25	$S^{*+}_{\alpha\beta(request)}$	0.25	0.75
$S^{*-}_{\alpha\beta(offer)}$	0.85	0.15	$S^{*-}_{\alpha\beta(request)}$	0.15	0.85

Table 4: The optimized HMM$^*_{\alpha\beta}$ observation function OX^*

$\Pi(\Omega)$	A	R	$\Pi(\Omega)$	A	R
$S^{*\theta}_{\alpha\beta(offer)}$	0.94	0.06	$S^{*\theta}_{\alpha\beta(request)}$	0.11	0.89
$S^{*+}_{\alpha\beta(offer)}$	0.81	0.19	$S^{*+}_{\alpha\beta(request)}$	0.16	0.84
$S^{*-}_{\alpha\beta(offer)}$	0.97	0.03	$S^{*-}_{\alpha\beta(request)}$	0.07	0.93

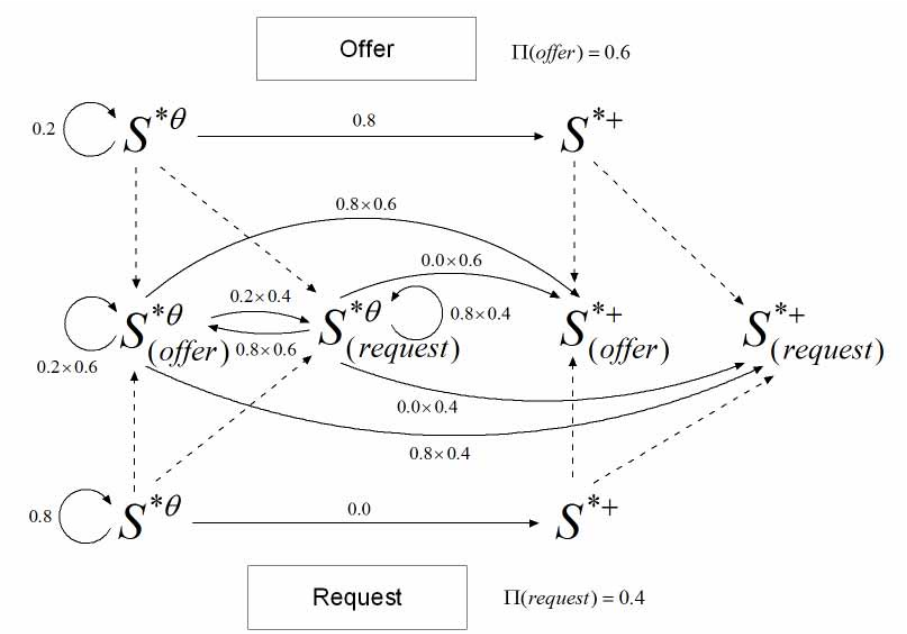


Figure 2: The conversion process from T^* to TX^*

Table 5: The POMDP$^*_{\alpha\beta}$ state transition function T^* of the egoism-90 exchange strategy model

	(a) *offer_service*			(b) *request_service*		
$\Pi\left(\mathrm{S}^*_{\alpha\beta}\right)$	$S^{*\theta}_{\alpha\beta}$	$S^{*+}_{\alpha\beta}$	$S^{*-}_{\alpha\beta}$	$S^{*\theta}_{\alpha\beta}$	$S^{*+}_{\alpha\beta}$	$S^{*-}_{\alpha\beta}$
$S^{*\theta}_{\alpha\beta}$	0.09	0.91	0.00	0.89	0.00	0.11
$S^{*+}_{\alpha\beta}$	0.00	1.00	0.00	0.08	0.90	0.02
$S^{*-}_{\alpha\beta}$	0.70	0.20	0.10	0.00	0.00	1.00

Table 6: The new POMDP$^*_{\alpha\beta}$ observation function O^* of the egoism-90 service exchange strategy model

	(a) *offer_service*		(b) *request_service*	
$\Pi(\Omega)$	A	R	A	R
$S^{*\theta}_{\alpha\beta}$	0.94	0.06	0.11	0.89
$S^{*+}_{\alpha\beta}$	0.81	0.19	0.16	0.84
$S^{*-}_{\alpha\beta}$	0.97	0.03	0.07	0.93

correspondent POMDP$^*_{\alpha\beta}$ states, considering the two types of service exchange proposals. The corresponding resulting part of the POMDP$^*_{\alpha\beta}$ state transition function $T^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi\left(\mathrm{S}^*_{\alpha\beta}\right)$ for the offer_service and request_service proposals are shown, respectively, on the top and on the bottom of the figure. One can observe, for example, that

$$T^*\left(S^{*\theta}_{\alpha\beta}, \mathit{offer\_service}\right)\left(S^{*\theta}_{\alpha\beta}\right) = TX^*\left(S^{*\theta}_{\alpha\beta(\mathit{offer})}\right)\left(S^{*\theta}_{\alpha\beta(\mathit{offer})}\right) + TX^*\left(S^{*\theta}_{\alpha\beta(\mathit{offer})}\right)\left(S^{*\theta}_{\alpha\beta(\mathit{request})}\right) = 0.07 + 0.02 = 0.08.$$

The function T^* is presented in Table 5. Applying Theorem 7 in the reverse order, the HMM$^*_{\alpha\beta}$ observation function $OX^* : \mathrm{SX}^*_{\alpha\beta} \to \Pi(\Omega)$ (Table 4) can be converted into the POMDP$^*_{\alpha\beta}$ observation function $O^* : \mathrm{S}^*_{\alpha\beta} \times \mathrm{P} \to \Pi(\Omega)$ (Table 6). Table 6 is obtained by decomposing Table 4 accordingly when unifying the set $\mathrm{SX}^*_{\alpha\beta}$ into the set $\mathrm{S}^*_{\alpha\beta}$, considering the two kinds of service proposals.

Table 5 and 6 specify one POMDP$^*_{\alpha\beta}$, for a specific $*$, for regulating a "new" service exchange strategy, called the egoism-90 *service exchange strategy model. The regulation processes will require, in this case, three POMDP$^*_{\alpha\beta}$s, one for each $* \in \{-, \theta, +\}$.*

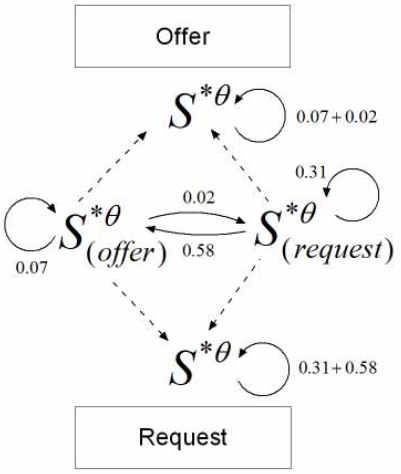


Figure 3: The conversion process from TX^* to T^*

6. CONCLUSION

System openness implies that the set of possible exchange strategies adopted by the agents may vary widely, so that agents aiming to regulate their interaction have to learn the social exchange strategies adopted by their partner agents at any time.

The essential problem of an agent aiming to regulate its exchanges with a partner is, then, to discover the POMDP service exchange strategy model that it should apply in order to lead to equilibrium the exchanges of services that it performs with its partner. We have taken such learning problem to be a problem of learning HMMs for

Table 7: The $\text{HMM}^*_{\alpha\beta}$ state transition function TX^* obtained from the egoism-80 service exchange strategy model

$\Pi\left(\text{SX}^*_{\alpha\beta}\right)$	$S^{*\theta}_{\alpha\beta(offer)}$	$S^{*\theta}_{\alpha\beta(request)}$	$S^{*+}_{\alpha\beta(offer)}$	$S^{*+}_{\alpha\beta(request)}$	$S^{*-}_{\alpha\beta(offer)}$	$S^{*-}_{\alpha\beta(request)}$
$S^{*\theta}_{\alpha\beta(offer)}$	0.120	0.080	0.480	0.320	0.000	0.000
$S^{*\theta}_{\alpha\beta(request)}$	0.480	0.320	0.000	0.000	0.120	0.080
$S^{*+}_{\alpha\beta(offer)}$	0.000	0.000	0.600	0.400	0.000	0.000
$S^{*+}_{\alpha\beta(request)}$	0.072	0.048	0.480	0.320	0.048	0.032
$s^{*-}_{\alpha\beta(offer)}$	0.300	0.200	0.180	0.120	0.120	0.080
$S^{*-}_{\alpha\beta(request)}$	0.000	0.000	0.000	0.000	0.600	0.400

Table 8: The optimized $\text{HMM}^*_{\alpha\beta}$ state transition function TX^*

$\Pi\left(\text{SX}^*_{\alpha\beta}\right)$	$S^{*\theta}_{\alpha\beta(offer)}$	$S^{*\theta}_{\alpha\beta(request)}$	$S^{*+}_{\alpha\beta(offer)}$	$E^{*+}_{\alpha\beta(request)}$	$S^{*-}_{\alpha\beta(offer)}$	$S^{*-}_{\alpha\beta(ask)}$
$S^{*\theta}_{\alpha\beta(offer)}$	0.07	0.02	0.56	0.35	0.00	0.00
$S^{*\theta}_{\alpha\beta(request)}$	0.58	0.31	0.00	0.00	0.10	0.01
$S^{*+}_{\alpha\beta(offer)}$	0.00	0.00	0.67	0.33	0.00	0.00
$S^{*+}_{\alpha\beta(request)}$	0.03	0.05	0.66	0.24	0.01	0.01
$S^{*-}_{\alpha\beta(offer)}$	0.38	0.32	0.10	0.10	0.04	0.06
$S^{*-}_{\alpha\beta(request)}$	0.00	0.00	0.00	0.00	0.65	0.35

the agents' exchange processes, and then concerting those models into POMDPs for such exchange strategies.

The paper formalized in terms of commutative diagrams the relationship between both models, proving the correctness of the conversion procedures. The commutativity of the diagrams was possible due to the isomorphisms that we have constructed between the state spaces of the models. A schematic application to the regulation of service exchange processes in open MAS was presented, with examples helping to clarify the formalism.

Several exchange strategies were defined in previous works (see. e.g., [7, 8, 9, 19]), for which we define POMDPs models which optimal policies derived BDI plans for the regulation problem. The conversion procedures presented here, which are based in first principles, were really effective in learning such models.

The regulation of service exchange processes is an important issue in the simulation of social management in environments rich in service exchanges (e.g., the vegetable urban gardens of Sevilla [6]), hotels [17]), and in order to help the analysis of social interactions and workers/partners' reciprocity for organizational design [10].

Finally, we notice that the paper leaves open the situations where both agents try to regulate each other, at the same time. This introduces a game-theoretic scenario, which is still ongoing work.

Acknowledgments. Work supported by CNPq (Proc. 305131/10-9, 560118/10-4, 476234/2011-5, 559743/2010-6, 470288/2011-6, 310105/2011-0) and FAPERGS (Proc. 11/0872-3, 11/0921-0).

7. REFERENCES

[1] P. Blau. *Exchange And Power In Social Life*. Wiley, New York, 1964.

[2] R. H. Bordini, J. F. Hübner, and M. Wooldrige. *Programming MAS in AgentSpeak Using Jason*. Wiley, Chichester, 2007.

[3] K. Cook and T. Yamaguchi. Power relations in exchange networks. *American Sociological Review*, 55:297–300, 1990.

[4] A. C. R. Costa and G. P. Dimuro. Introducing social groups and group exchanges in the PopOrg model. In *Proc. of AAMAS'09*, pp. 1297–1298, Budapest, 2009.

[5] A. C. R. Costa and G. P. Dimuro. A minimal dynamical organization model. In V. Dignum, ed., *Hand. of Research on MAS Systems: Semantics and Dynamics of Organizational Models*, pp. 419–445. IGI, Hershey, 2009.

[6] G. Dimuro and E. Jerez. La comunidad como escala de trabajo en los ecosistemas urbanos. *Rev. Ciencia y Tecnología*, 10:101–116, 2011.

[7] G. P. Dimuro, A. C. R. Costa, L. V. Gonçalves, and A. Hübner. Centralized regulation of social exchanges between personality-based agents. In *COIN II*, vol. 4386 of *LNAI*, pp. 338–355. Springer, Berlin, 2007.

[8] G. P. Dimuro, A. C. R. Costa, L. V. Gonçalves, and D. R. Pereira. Recognizing and learning models of social exchange strategies for the regulation of social interactions in open agent societies. *Journal of the Brazilian Computer Society*, 17(3):143–161, 2011.

[9] G. P. Dimuro, A. C. R. Costa, and L. A. M. Palazzo. Systems of exchange values as tools for multi-agent organizations. *Journal of the Braz. Computer Society*, 11(1):31–50, 2005.

[10] R. Dur and H. Roelfsema. Social exchange and common agency in organizations. *Journal of Socio-Economics*, 39(1):55 – 63, 2010.

[11] R. Emerson. Social exchange theory. In A. Inkeles et al., editors, *Sociology*, vol. 2, pp. 335–362. Annual Reviews, Palo Alto, 1976.

[12] M. H. I. Franco, A. C. R. Costa, and H. Coelho. Exchange values and social power supporting the choice of partners. *Pueblos y Fronteras Digital*, 6(9), 2010.

[13] P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of AI Research*, 24:49–79, 2005.

[14] F. Grimaldo, M. Lozano, and F. Barber. Coordination and sociability for intelligent virtual agents. In *COIN III*, vol. 4870 of *LNAI*, pp. 58–70. Springer, Berlin, 2007.

[15] G. Homans. *Social Behavior – Its Elementary Forms*. Harcourt, Brace & World, New York, 1961.

[16] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[17] E. Ma and H. Qu. Social exchanges as motivators of hotel employees' organizational citizenship behavior. *International Journal of Hospitality Management*, 30(3):680–688, 2011.

[18] I. L. MacDonald and W. Zucchini. *Hidden Markov and Other Models for Discrete-valued Series*. Chapman, Boca Raton, 1997.

[19] D. Pereira, L. V. Gonçalves, G. P. Dimuro, and A. C. R. Costa. Towards the self-regulation of personality-based social exchange processes in MAS. In *Advances in AI*, vol. 5249 of *LNAI*, pp. 113–123. Springer, Berlin, 2008.

[20] J. Piaget. *Sociological Studies*. Routlege, London, 1995.

[21] M. R. Rodrigues. *Social techniques for effective interactions in open cooperative systems*. PhD thesis, University of Southampton, 2007.

[22] M. R. Rodrigues, A. C. R. Costa, and R. Bordini. A system of exchange values to support social interactions in artificial societies. In *Proc. of AAMAS'03*, pages 81–88, Melbourne, 2003.

[23] M. R. Rodrigues and M. Luck. Effective MAS interactions for open cooperative systems rich in services. In *Proc. AAMAS'09*, pages 1273–1274, 2009.

Learning with more than rewards: The Implicit Signalling of Distributed Punishment

Daniel Villatoro
IIIA - CSIC
Bellatera, Barcelona, Spain
dvillatoro@iiia.csic.es

Giulia Andrighetto
ISTC - CNR
Rome, Italy
giulia.andrighetto@istc.cnr.it

Jordi Brandts
Dept. of Business Economics
(UAB)
IAE - CSIC
jordi.brandts@iae.csic.es

Jordi Sabater-Mir
IIIA - CSIC
Bellatera, Barcelona, Spain
jsabater@iiia.csic.es

Rosaria Conte
ISTC - CNR
Rome, Italy
rosaria.conte@istc.cnr.it

ABSTRACT

Recent studies show that punishment plays a crucial role in favoring and maintaining social order both in real and in virtual societies. However, very little attention has been paid so far to the potential of distributed punishment. With distributed punishment we refer to the practice that occurs when a number n of agents, where $n > 1$, inflicts the target a material damage, such that each punisher sustains a share of the punishment cost. The experiments performed in this work prove that human subjects are not only motivated by the negative reward associated with punishment. The same material reward has a different effect on the subjects future compliance depending on the way it is implemented, having a stronger effect when it conveys an implicit normative message. This provide us with the intuition that human learning is not only affected by the economic rewards. In this work we put forward the hypothesis that distributed punishment is more effective in human subjects than individual punishment, because the higher the number of punishers, the less likely the observers will interpret their behaviors as dictated by the self-interest and, conversely, the more likely they will attribute the punishment to impersonal, possibly normative and legitimate reasons. In support of our hypothesis, we present cross-methodological data, i.e. laboratory experimental data obtained by using a platform populated by both humans and virtual agents, and agent-based simulation data. The two experiments, which yield convergent results, seem to confirm our hypothesis.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Experimentation

Keywords

Incentives for Cooperation, Normative systems, Social simulation, Human-robot/agent interaction, Modeling cognition and socio-cultural behavior

1. INTRODUCTION

Cooperation and norm compliance is a puzzle for both the social scientist and the game theorist. Solutions to the puzzle of cooperation are generally found out in punishment-based social control [17]. Theoretical and laboratory studies indicate that cooperation and the maintenance of social order typically requires a punishment threat, as the temptation to cheat, free-ride and violate norms is always strong for autonomous agents [27, 7, 22, 20]. But what is punishment?

Within the classical economic approach to rational action, punishment is usually modeled as (a) a material damage, i.e. a cost inflicted to the target by (b) a single agent that (c) sustains the costs of the punishing action (including those consequent to possible retaliations) [6]. On the contrary, ethnographic evidence shows that punishment is often distributed, i.e. performed by many, which share the costs of acting, and includes gossip and other forms of explicit or implicit communication among punishers [16]. How to account for the gap between empirical evidence and the expectations of rationality theory?

As claimed by [18], there are at least two different forms of reaction to a wrongdoing: *punishment* and *sanction*. Punishment is a reaction aimed to impose the victim a cost. Sanction is a norm-oriented punishment, aimed to (a) impose a cost, (b) convey a belief, namely a belief that a norm exists and was violated by the target, (c) convey a second belief stating that the cost is consequent to the victim's violation of that norm. We suggest that sanction is more effective than punishment because it is aimed not only to deter further violation in presence of the punisher, but also to develop an autonomous motivation in the target to comply with the norm even in absence of surveillance. Experimental evidence [12, 4, 21] shows that drawing people's attention on a social norm - as sanction does - plays a pivotal role in eliciting compliance: even a strong personal commitment to a norm does not predict behavior if that norm is neither activated nor is a focus of attention (for a detailed analysis see Sec. 4.4). Thus sanction in general is more effective than mere punishment in promoting cooperation, because in addition to impose a material cost it also exploits the motivational power of social norms.

But how to tell when a reaction is a sanction rather than a punishment or vice versa? Often, sanctions are accompanied by an explicit norm-based educational message (like "you shouldn't have done that", "one ought to behave like this", etc.). People in a queue will punish someone who tries to get cute and jump on top by expressing their condemnation explicitly. But educational messages are not always explicit. Sometimes, they are drawn from the context or by behavioral implicit communication. What are the indicators that a material punishment is dictated by normative reasons?

In this work, we focus on the norm signalling power of distributed punishment and we put forward the hypothesis that when distributed punishment is more effective than individual punishment, even if the perceived material reward is the same. The reason is because the higher the number of punishers, the less likely the observers will interpret their behaviors as dictated by the self-interest and, conversely, the more likely they will attribute the punishment to general, impersonal, possibly normative and legitimate reasons. In other words, our hypothesis is that *distributed* punishment is more likely to be interpreted as a sanction than individual punishment, even when the educational message is implicit or even absent at all, and this accounts for the gap mentioned above between empirical evidence and the expectations of rationality theory. With distributed punishment we refer to the practice that occurs when a number n of agents, where $n > 1$, inflicts the target a material damage, such that each punisher sustains a share of the punishment cost. For example, people in the queue will reproach the nasty guy, while feeling supported by one another and sharing both the costs (e.g. the emotional burden) of conveying their reproach and the costs of repressing the cheater's potential counter-reaction.

We present cross-methodological evidence supporting our hypothesis that distributed punishment is more effective than individual punishment. This hypothesis was tested in a laboratory experiment based on a platform populated by both humans and virtual agents. In this experiment, we compare the respective effects of individual versus distributed punishment. Results seem to confirm our hypothesis that the latter condition favors cooperation more than the former even at the same cost for the victim, and at the same time prove inadequate the rationality expectation that material incentive is a sufficient mechanism of norm enforcement and social control. Material incentives are the mechanisms in which *Reinforcement Learning* bases its functioning, however we observe that this category of learning algorithms does not capture the empirical result obtained with human subjects. In this work we observe how a cognitive agent architecture is able to extract the implicit message contained in different types of punishment with the same material values. Notice that the way the experiment has been implemented prevents an explanation based on reputation, as participants cannot identify one another.

The laboratory experiment was replicated in a virtual environment by means of agent-based simulation, yielding similar results. The proposed agent architecture is given with an adaptive mechanism that allowed us to properly explore the power of "moral suasion" of distributed versus individual punishment. Finally, the artificial experiment provided a further setting in which to check the results obtained through the laboratory experiment, and contribute to generalise our model.

The paper is structured in the following way: Sec. 2 describes the approach used in this paper, then in Sec. 3 the motivation behind the present paper and the related work are presented. To test the effectiveness of *Coordinated Punishment*, in Sec. 4, we present a laboratory experiment in which human subjects interact with virtual agents and they coordinate to impose punishment. In Sec. 5, the experiment conducted in the laboratory has been replicated through agent-based simulation. Finally, in Sec. 7, some conclusions are presented.

2. HUMAN-AGENT IN THE LOOP

Quite often multiagent systems are associated only with computational systems populated by artificial entities. Recently, new topics like augmented reality or ambient intelligence are making evident something that has been always there in the original definition of what a multiagent system is: a system composed of multiple interacting intelligent agents, where *intelligent agents* include not only artificial beings but also humans [11, 8]. This means that if we want to develop artificial agents that evolve in an open multiagent system we have to start thinking that those agents will interact not only with other artificial entities but also with human beings. Moreover, the agent usually will not know the real (artificial or biological) nature of its partner.

This fact has important implications for the way in which an agent should behave before others. In particular, when we are talking about social behavior, it is important to have agents that behave like a human: it is artificial entities that have to adapt to humans, not the other way around. Consequently, we need agents that *understand* how humans behave.

Until now, the design of agents that tried to resemble humans in their social behavior was usually approached from the perspective of *Homo Economicus*. According to this conception, the agent is a rational entity that is utility driven and selfish, i.e. it always maximizes its own monetary payoff. This view, which has been the main stream in economics for many years, has recently been questioned. Humans are not always purely selfish and perfectly rational, especially with respect to social behavior, and simple game theoretical models are found insufficient by a widening share of the scientific community. Cognitive models can be the solution. However there has always been a problem associated to these kinds of models: the large number of parameters often required to tune the model, especially when one considers the arbitrariness involved in choosing them. The approach proposed in this paper minimizes this problem by introducing humans in the loop, both to tune the parameters of the cognitive model and also to validate it.

3. MOTIVATION AND RELATED WORK

Self-policing systems are those systems where the participants have the power and the responsibility to control the behaviors of the rest of their peers, normally without the intermission of a centralized entity. In order to properly work, self-policing systems require social norms prescribing the correct conduct and some mechanisms to enforce these norms. The P2P Content Sharing Software like Napster is an example: pro-social swappers would apply a Distributed Denial of Service (DDoS) attack [24] against anti-social peers,

i.e. peers that shared none or a low quantity of files, or bad quality files. This situation represents a social dilemma, i.e. a situation in which collective interests are at odds with private interests. Individuals are better off when make use of public services, without contributing to their maintenance. However, if everyone acted according to their narrow self-interest then these resources would not be provided and everyone would be worse off.

In social dilemmas, including those faced by simulated P2P networks, when correctly designed and sufficiently strong, punishment represents a viable tool to promote pro-social behaviors [5]. Recent works have investigated the effect of punishment in solving social dilemmas with simulation techniques [3], while others have analyzed the types of norms (and punishment associated to their violations) to be incorporated in electronic institutions [19]. [29] focuses on the role of punishment in favoring norms' identification and [30] has explored the effectiveness of different punishment technologies (i.e. punishment and sanction) in favoring and maintaining social order.

The aim of this work is to explore the viability of distributed punishment in favoring cooperation. To test it, a laboratory experiment populated both by humans and virtual agents and reproducing a social dilemma scenario has been conducted.

The experiment conducted in the laboratory has also been replicated by agent-based simulation, obtaining convergent results. As we will describe in Sec. 5, the normative agent used in the simulation experiment is endowed with an architecture allowing it to capture normative information and in particular that explicitly or implicitly conveyed by the sanctioning reactions of others (for important contributions in the modeling of normative agents see also [29, 9, 15]. Finally, the performance of the normative architecture used in the agent-based simulations has been contrasted against that of classical reinforcement learning agents.

4. DISTRIBUTED PUNISHMENT: MIXING TOGETHER ARTIFICIAL AND HUMAN SUBJECTS

To test the viability of distributed punishment in achieving and maintaining cooperation, we conducted a laboratory experiment reproducing a social dilemma situation. In particular, participants (divided in groups of 4) played a *public goods game* in which they had to decide whether to invest or not their private endowment in a group fund. As shown in Table 1, payoffs are such that it is individually rational to abstain from investing in the group fund, yet the pro-social group best strategy would be investing in the group fund because this yields a bonus. The more people invest in the group fund the larger their share of the bonus. After having decided whether to contribute or not to the group fund, participants have the possibility to punish those who did not contribute.

The experiment consists of four treatments, which differ with respect to the number of the punishing subjects: (1) no punishment, (2) the subject is punished by *one* peer, (3) the subject is punished by *two* peers, (4) the subject is punished by *three* peers.

The material damage imposed on the punished agent in treatments 2, 3 and 4 is *identical* (i.e. it reduces the payoffs of the punished subject to zero) and the way the experiment has been implemented prevents the occurrence of reputational effects, as participants cannot identify one another. Thus, the material and symbolic incentives imposed in treatments 2, 3 and 4 are the same. As claimed in section 1, we suggest that if subjects react differently in treatments 2, 3, and 4, this seems to be owed to the growing probability that the punishment is interpreted by the target as a sanction, i.e. as conveying a normative message of peer condemnation.

Since we are interested in exploring the normative effect of distributed punishment rather than the motives driving individuals to coordinate when punishing, we introduced *confederate subjects* to achieve coordination among punishers. To have a completely controlled situation and observe the subjects' reactions, three confederates are needed for each experimental subject. For the sake of resource optimization (in terms of economic and time expenses), we decided these confederates had to be *virtual* pre-programmed agents. Each human player was set in a group with three pre-programmed confederate agents, which remained the same for the entire session. Subjects knew that the experiment would last 40 rounds, but did neither know the existence of virtual entities nor the identity of the other participants in their group[1].

4.1 Experimental Design

The game consists of 40 rounds, and each round is structured in 3 stages:

1. In the *first stage*, participants decide simultaneously and without communication whether or not to contribute to a public good (as shown by the payoffs matrix in Table 1, not contributing is the dominant strategy). If they do not contribute the whole amount remains in the player's own account.

2. In the *second stage*, participants are informed of their individual earnings, the decisions of the other participants in their group, and their related earnings. After being informed, participants are given the opportunity to punish each other simultaneously. Punishment has two distinct effects on the payoffs of the punisher and of the punished. It reduces the payoffs of the punished to *zero*. Additionally, the punisher faces a cost of 10 ECUs, possibly shared among the other agents punishing the same target. As in stage 1, punishment decisions are made simultaneously and without communication.

3. In the *third stage*, after all the decisions are taken, participants are informed of the decisions of the other participants in their group and the resulting payoffs of that round are shown.

In order to observe whether monolateral punishment has a different effect on cooperation than distributed one, 4 treatments have been done, in which the number of punishing confederate agents per group varied:

- *0 Punishers Treatment*: Each human subject is assigned to a group with three non-punisher confederate agents.

[1]At the end of the experiment participants were asked to reply to a questionnaire. Only 3 out of the 80 subjects replied that they might be playing with virtual pre-programmed agents; the remaining participants did not find out that they were playing with virtual agents.

	$0C$	$1C$	$2C's$	$3C's$
C	5	10	15	20
D	10	15	20	25

Table 1: Payoff Matrix for the Distributed Punishment Experiment. C stands for Contribute and D for Defect.

	1^{st} Stage	2^{nd} Stage
$round < 10$	$Pr(C) = 0.5$	$Pr(P, A_x) = 0.25$
$round >= 10$	$Pr(C) = 0.9$	$Pr(P, A_D) = 0.9$

(a) Punisher Confederate

	1^{st} Stage	2^{nd} Stage
$round < 10$	$Pr(C) = 0.5$	$Pr(P, A_x) = 0.25$
$round >= 10$	$Pr(C) = 0.9$	$Pr(P, A_x) = 0.0$

(b) Non-Punisher Condeferate

Figure 1: Confederate Agents Behaviors.

- *1 Punisher Treatment*: Each human subject is assigned to a group with one punisher confederate agent and two non-punisher confederate agents.
- *2 Punishers Treatment*: Each human subject is assigned to a group with two punisher confederate agents and one non-punisher confederate agent.
- *3 Punishers Treatment*: Each human subject is assigned to a group with three punisher confederate agents.

Punisher and non-punisher virtual confederate agents act in the following way: during the first *ten* rounds, the virtual agents' behavior is programmed to be exactly the same, then it changes after the tenth round. From round one to ten, all virtual actors contribute 50% of the times and punish 25% of the other agents (independently of how they acted at the first stage). After the tenth round: (a) all the virtual agents contribute 90% of the times; (b) non-punishers never punish, (c) and punishers act only on defectors 90% of the times and they can punish only if they cooperated at the first stage. These behaviors are summed up in Figure 1. The behavior of the confederate agents has been based on experimental data. We observed that in the first round, roughly 50% of humans cooperate, and 25% of them punish without following a specific trend. Even though we are aware of the possible consequences of the assumption, the confederate agents mimic for the first ten rounds of the experiment the behavioral dynamics observed in humans (during the first round). This initial randomic period will allow us to observe accurately the effect of the different punishment treatments.

4.2 Procedures

A total of 80 subjects participated voluntarily in a repeated Public Goods game experiment conducted in a regular Experimental Economics Lab[2]. All the four sessions were conducted in December 2010, with an average of 20 participants. The participants were students from a wide range of fields of study and interacted anonymously. Subjects were not allowed to participate in more than one session. Prior to the first round of each session, participants were randomly arranged in groups of four and told they would be the same for the entire experiment (following a partner protocol, as in [14]). Participants could neither see other members of their groups, nor communicate with them. At the end of the game, human participants were paid depending on their performance in the game. The experiment was programmed by using HIHEREI [8]. As this platform is built on top of an *Electronic Institution*, we can easily provide it with the capability to also perform experiments with virtual agents, and rapidly change the behavior of the confederate agents.

[2]The name of the lab has been avoided in order to ensure anonymity

4.3 Results

In Figure 2(a), the average cooperation rates obtained in the four treatments are shown. Only the behavior of human subjects is plotted. In the four treatments, from round 0 to 10 human subjects interact with virtual agents that contribute 50 % of the times and punish 25% of the other participants in their group, independently on how they acted at the first stage. During these first ten rounds, the cooperation level decreases but does not collapses. Even though humans are exposed to inconsistent actions, such as cooperators being punished or even worst cooperators being punished by defectors, the fact that there is a 25% probability of being punished prevent them from defecting all the times.

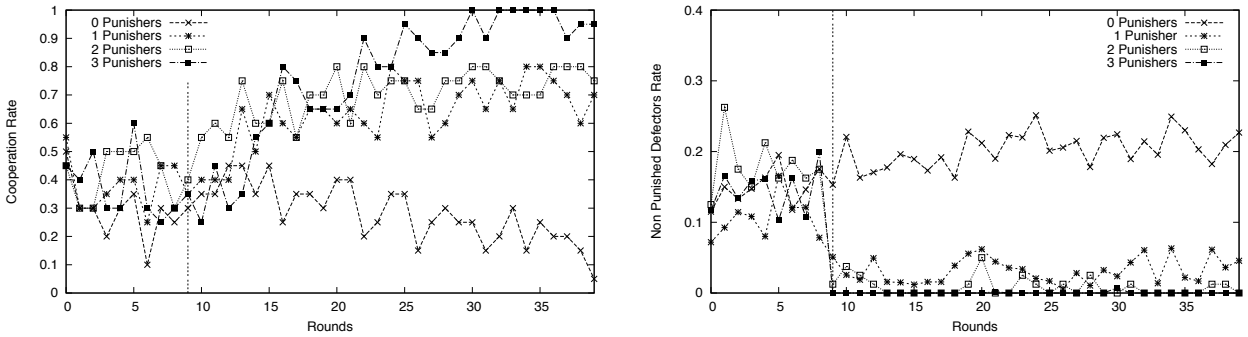
After run 10, it is possible to appreciate the effects of the different experimental treatments on the cooperation level. In the non punishment condition (i.e. 0 punisher treatment), the cooperation level rapidly collapses. Since the incentive scheme is structured in such a way that defecting is the dominant strategy (see Table 1), this result is not surprising. On the contrary, in the three punishing treatments the cooperation level increases with respect to the first 10 rounds and is higher than the one obtained in the non punishment treatment. It is interesting to notice that being punished by three group members (i.e. 3 punishers treatment) leads to a higher cooperation level, than by two or just one subjects.

A small difference between the treatments should be taken into consideration when analyzing the results: when the number of punishing actors increases, the probability of a subject being punished (as each punisher actor punishes 90% of the times) increases as well. In other words, the defector's probability to be punished in each of the treatments is 90% (1 punisher), 99% (2 punishers) and 99.9% (3 punishers) respectively.

As shown in Figure 2(b), these different probabilities directly affect the ratio of non-punished defectors, which remains below 5% in the *1 punisher* treatment, around 2% in the *2 punishers* treatment, and 0,005% in the *3 punishers* treatment. But these differences are not significant enough to explain the differences in cooperation levels shown in Figure 2(a). For this reason, we suggest that the large differences in the cooperation levels are unlikely to be an effect of a difference in the defector's probability to be punished. In Sec. 3, we present and discuss a simulation experiment aimed to check if the the different punishment probabilities implemented affect the results.

4.4 Discussion

As the *same* material damage is imposed in all the treatments (except for the *0 Punishment Treatment* one), we hypothesize that the explanation for the difference on the



(a) Average Cooperation Rates

(b) Non-Punished Defectors Rates

Figure 2: Results of Human Subjects playing with Virtual Confederates

cooperation rates has to be found in additional information that the punished players receive. The result shown in Figure 2(a), i.e. the fact that the same material incentive has a different effect if imposed only by one punisher or by the actions of more than one punishers, is at odds with the perspective that looks at punishment only from the classical economic perspective [3]. This perspective claims that punishment achieves deterrence by modifying the relative costs and benefits available in a given situation, in such a way that wrongdoing becomes a less attractive option [23].

According to this perspective, in our previous experiment, the three punishment treatments should be expected to yield similar results in terms of cooperation. On the contrary, our data show that with the same material incentive, the higher the number of punishers, the higher the cooperation rates obtained. We claim that the classical economic perspective on punishment is incomplete and suggest that a more insightful understanding of this enforcing mechanism is available once the *norm-signalling* nature of punishment is identified.

The term sanction has been previously used to indicate a practice that in addition to imposing a cost also signals the existence of a norm and that its violation is not condoned. This signalling task can be achieved by explicit communication (e.g. by scolding wrongdoers), but in many situations behavioral implicit communication, such as a bad look, is enough (for an analysis of behavioral implicit communication, see [10]). We suggest that the higher the number of punishers, the less likely the observers will interpret their behaviors as dictated by the self-interest and, conversely, the more likely they will attribute the punishment to impersonal, possibly normative and legitimate reasons. In other words, our hypothesis is that distributed, punishment is more likely to be interpreted as sanction than individual punishment.

The normative information explicitly or implicitly communicated by sanction plays an important role in eliciting pro-social behaviors: as shown by recent neuroscientific data [25] it has the effect of framing the situation in such a way that not only motivations to avoid material costs are activated, but normative motivations as well. The activation of normative motivations to cooperate thereby leads pro-social behaviors to increase within the group.

Sanction mixes together material and normative aspects and consequently it changes the future behavior of individuals by influencing both their *cost-avoidance* and *normative* motivations. In order to decide how to behave, the individual will be driven by a combination of cost-avoidance and normative goals. The tandem work of cost-avoidance and normative motivations enables a higher and more durable cooperation level with respect to an enforcement mechanism that impacts on cost-avoidance motivations only [1]. And the experimental data presented seem to give additional support to this hypothesis.

To provide more evidence for it, in the next section, an agent-based simulation is presented and discussed. Agent-based modeling proved particularly apt to test our model, as it allowed to separately implement the material damage and the normative information conveyed by punishment, and explore their respective and combined effects on motivating individuals to abstain from further violations. Finally, it provided an additional setting on which to check the results obtained through the laboratory experiment, thus contributing to develop models that are generalizable.

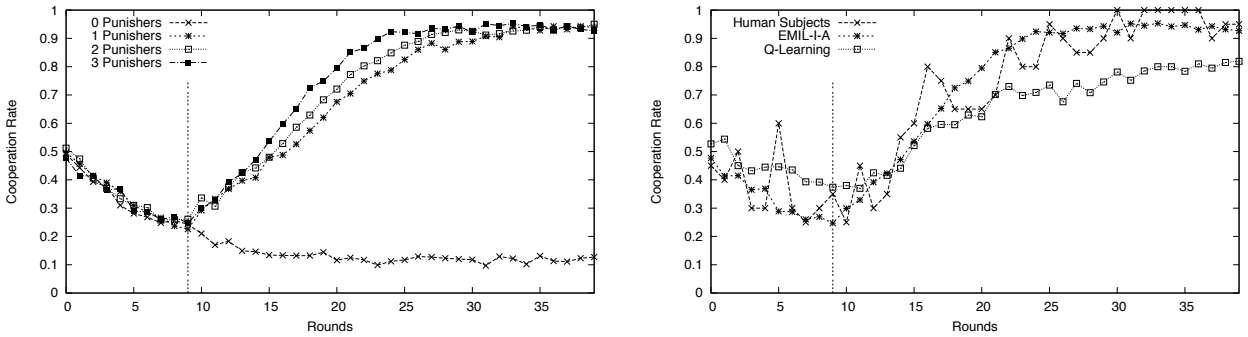
5. DISTRIBUTED PUNISHMENT: A SIMULATION EXPERIMENT

In order to capture the normative information conveyed by punishment, our simulation experiments were populated by agents able to interpret as normative the social information they are exposed to and to include it into their decision-making. The agent architecture used for such task is EMIL-I-A.

This normative architecture allows agents (a) to recognize norms; (b) to generate new normative representations and to act on them; (c) to influence other agents by direct communication and by the use of different types of punishment; and (d) to infer the normative information (explicitly or implicitly) conveyed by punishment. Moreover, these normative mental representations are not static. Depending on several social or mental factors a normative representation can gain or loose its *salience* over time [2, 21] [4]. The actions' of others

[3] As said in Sec. 4, the way in which the game experiment has been conducted prevents the occurrence of reputational effects

[4] For a more detailed description of the EMIL-I-A architecture see [1, 30]



(a) EMIL-I-A Subjects Cooperation Rates

(b) Comparing Humans, EMIL-I-A and RL agents

Figure 3: Simulation Results.

provide information about how important a norm is within a group. For example, the level of compliance, the amount of non punished norm violations, the frequency and typology of punishment, the consistency of the actions performed by other agents and so on are all signs through which people can infer how important and and active a social norm is.

EMIL-I-As' decisions are influenced by an *aggregation* of cost-avoidance and normative considerations. More specifically, their decision making consists of the following drives: (1) self-interested drive: this motivates agents to maximize their individual utility independent of what the norm prescribes; (2) normative drive: this motivates agents to comply with the norm, independent of external outcomes. This drive is directly affected by norm salience; the more salient the norm is, the higher the motivation to comply with it. The aggregation of both drives - individual and normative - will determine the agents' strategies, resulting in a different performance on cooperation rates with respect to classical reinforcement learning agents.

In particular, punishment has two distinct effects on EMIL-I-A's decisions: the costs it imposes, with the consequent reduction of the agent's payoffs, affects the individual drive of the agent (thus reducing the probability that the action that triggered the punishing reaction will be performed again); while the normative information that punishment explicitly or implicitly conveys impacts on the norm's salience and consequently on the normative drive. When EMIL-I-As observe more than one agent punishing a certain conduct, this widespread punishing reaction makes the agent create a candidate belief that the targeted action violated a social norm. Additional information is then needed to confirm this normative canditate belief.

As in the laboratory experiment presented in section 2, agents are arranged in groups of four. Each group consists of 1 EMIL-I-A and of 3 pre-programmed confederate agents (non endowed with the same architecture and that follow the same behavioral rules of the virtual pre-programmed confederate agents described in section 4.1). The same public goods game described in section 2 has been played and the same 4 different treatments (no punishment, 1 punisher, 2 punishers, and 3 punishers) are reproduced.

In Figure 3(a), the average cooperation rates[5] obtained in the four treatments are shown. It is interesting to notice that the cooperation dynamics achieved in the simulation experiment with EMIL-I-As are very similar to the ones obtained in the experiment with human subjects (see Figure 2(a)). However, the difference in the cooperation levels observed in the three punishment treatments in the laboratory experiment (with a higher level of cooperation when 3 punishers acted simultaneously) is not as strong as the one achieved with EMIL-I-A agents. A possible explanation for this difference is that humans in addition to being sensitive to the fact that three punishers acted together, are also influenced by the fact that is the group as a whole that reacts against his conduct. This additional information is not taken into account by EMIL-I-As.

As discussed in Sec. 4.3, the probability of being punished is different depending on the amount of punisher confederates. In order to observe whether EMIL-I-A has been affected by the different punishment probabilities, we contrasted its behavior with other types of agents' architectures. Namely, with agents driven exclusively by utilitarian motivations, that would definitely be affected by different punishment probabilities.

From the different Reinforcement Learning (RL) architectures, we decide to use a Q-Learning [31] (with a fixed 25% exploration rate) for two reasons: (1) it better represents the learning process of an utility-based human: it will maintain its strategy while obtaining benefits, and it will change it and learn otherwise; and (2) its basic parameter configuration allows for easy performance analysis, yet with a powerful potential of applications.

In Figure 3(b), it is shown (in the 3 punishers treatment) that RL agents obtain similar cooperative dynamics to humans, confirming that the utilitarian motivation in humans is very strong, although the cooperation rates are not as high as the ones obtained with humans and EMIL-I-As. The simulation results with RL agents provide us with a methodological validation of the EMIL-I-A architecture, confirming that the probabilistic effect introduced by the behavior of the confederates is not determinant in the higher cooperation rates obtained in the 3 punishers treatment. RL agents - driven only by utilitarian motivations - do not reach the same cooperation levels as EMIL-I-As and humans. Eventually, over a longer period of time, RL agents would converge to the same results. While EMIL-I-As reproduce dynamics

[5]We present the results of the average of 1000 simulation runs per treatment.

very similar to humans, the classical reinforcement-learning architectures are not as representative.

6. FUTURE WORKS

The study and analysis of self-regulated environments populated by humans and virtual agents seems to be of crucial importance for the future of multi-agent systems. Including humans in the agents paradigm also implies building agents that fully exploits the cognitive tricks by which humans decision making is affected.

The most immediate extension of our work would be to conduct new experiments with human subjects with the same probability of punishment in all three treatments in order to empirically conform to our hypothesis.

This work has served us to understand the relative effect of individual and distributed punishment in favouring people's compliance. As a future task experiments should be performed to understand the motivations of the punisher. Costly punishment is known to be a second order dilemma, but it has been shown that humans are willing to engage this cost in certain situations [17]. Our future experiments should provide us with the necessary information to understand what are the motivations, and dynamics, of a punisher.

With respect to the simulation, one of the future contributions should answer the following question: how should the utility function of a reinforcement learning agent look like in order to account for the effects of distributed punishment? This question might be easy to answer by adding some multipliers to the reward to reflect the amount of punishers, but it would directly lead to a more general question: what is the simplest utility function that can be designed for a reinforcement learning agent to mimic EMIL-I-A results? The answer might be trivial to find for special cases, but it would be interesting to analyze a generalizable utility function.

As a more long term objective, we plan to incorporate into the model other cognitive mechanisms studied by experimental economists that could apply to virtual communities, like, for example, communication, reputation or partner selection. These three mechanisms have been widely studied in the MAS literature [26, 28, 13], although very little attention has been paid to the integration of results obtained by experimental economics into existing agents architectures.

7. CONCLUSIONS

In this study, we have provided some experimental evidence to show the viability of distributed punishment in promoting cooperation. These data provide support for the hypothesis that punishment is effective in regulating people's behavior not only through economic incentives, but also thanks to the normative information it conveys and the normative request it asks of people. Distributed punishment is a powerful tool through which messages of peer condemnation and of shared norm defense are (explicitly or implicitly) conveyed.

In the present work, agent technologies have been used for two different, but essential, purposes. The first one allowed us to build a experimental platform where human subjects could participate transparently in an electronic institution and play against pre-programmed confederate agents. Secondly, the experiment conducted in the laboratory has been replicated with an agent-based simulation using the EMIL-I-A architecture, whose decision making is driven both by cost-avoidance and normative motivations.

The comparison of the results from the experiments with humans and of the simulations show that the simulation model captures the essential features of the human data. Finally, we have compared the performance of the EMIL-I-A architecture with other classical reinforcement learning architectures, observing that the former reproduces behavioral dynamics more similar to humans than the latter ones. The more general point we want to make is that agent-based modeling provides a valid methodology for checking whether the results obtained in laboratory experiments with humans are consistent with the theoretical cognitive model proposed.

8. REFERENCES

[1] G. Andrighetto and D. Villatoro. Beyond the carrot and stick approach to enforcement: An agent-based model. In B. Kokinov, A. Karmiloff-Smith, and N. J. Nersessian, editors, *European Perspectives on Cognitive Science*. New Bulgarian University Press, 2011. ISBN 978-954-535-660-5.

[2] G. Andrighetto, D. Villatoro, and R. Conte. Norm internalization in artificial societies. *AI Communications*, 23:325–339, 2010.

[3] A. L. C. Bazzan, S. R. Dahmen, and A. T. Baraviera. Simulating the effects of sanctioning for the emergence of cooperation in a public goods game. In *AAMAS '08*, pages 1473–1476, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[4] C. Bicchieri. *The Grammar of Society: The nature and Dynamics of Social Norms*. Cambridge University Press, 2006.

[5] A. Blanc, Y.-K. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *INFOCOM*, pages 374–385, 2005.

[6] R. Boyd, H. Gintis, and S. Bowles. Coordinated punishment of defectors sustains cooperation and can proliferate when rare. *Science*, 328(5978):617–620, 2010.

[7] R. Boyd and P. Richerson. Punishment allows the evolution of cooperation (or anything else) in sizable groups. *Ethology and Sociobiology*, 13(3):171–195, May 1992.

[8] I. Brito, I. Pinyol, D. Villatoro, and J. Sabater-Mir. Hiherei: human interaction within hybrid environments regulated through electronic institutions. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1417–1418, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[9] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre. The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 9–16, New York, NY, USA, 2001. ACM.

[10] C. Castelfranchi and L. Pezzulo, G. Tummolini. Behavioral implicit communication (bic): Communicating with smart environments via our practical behavior and its traces. *International*

Journal of Ambient Computing and Intelligence, 2(1):1–12, 2010.

[11] M. Cavazza, R. S. de la Camara, and M. Turunen. How was your day?: a companion eca. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1629–1630, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[12] R. B. Cialdini, R. R. Reno, and C. A. Kallgren. A focus theory of normative conduct: Recycling the concept of norms to reduce littering in public places. *Journal of Personality and Social Psychology*, 58(6):1015–1026, 1990.

[13] A. P. de Pinninck, C. Sierra, and W. M. Schorlemmer. Friends no more: Norm enforcement in multi-agent systems. In M. Durfee, E. H.; Yokoo, editor, *Proceedings of AAMAS 2007*, pages 640–642, 2007.

[14] L. Denant-Boemont, D. Masclet, and C. Noussair. Punishment, counterpunishment and sanction enforcement in a social dilemma experiment. *Economic Theory*, 33(1):145–167, October 2007.

[15] F. Dignum, V. Dignum, and C. M. Jonker. Multi-agent-based simulation ix. chapter Towards Agents for Policy Making, pages 141–153. Springer-Verlag, Berlin, Heidelberg, 2009.

[16] R. Ellickson. *Order without Law : How Neighbors Settle Disputes*. Harvard University Press, June 2005.

[17] E. Fehr and S. Gachter. Altruistic punishment in humans. *Nature*, 415:137–140, 2002.

[18] F. Giardini, G. Andrighetto, and R. Conte. A cognitive model of punishment. In S. O. . R. Catrambone, editor, *Proceedings of the 32nd Annual Conference of the Cognitive Science Society Austin, TX: Cognitive Science Society*, pages 1282–1288. Portland, Oregon, 2010.

[19] D. Grossi, H. M. Aldewereld, and F. Dignum. Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, pages 101–114. Springer, 2007.

[20] D. Helbing, A. Szolnoki, M. Perc, and G. SzabÃş. Punish, but not too hard: how costly punishment spreads in the spatial public goods game. *New Journal of Physics*, 12(8):083005, 2010.

[21] D. Houser and E. Xiao. Understanding context effects. *Journal of Economic Behavior & Organization*, 73(1):58–61, January 2010.

[22] K. Jaffe and L. Zaballa. Co-operative punishment cements social cohesion. *Journal of Artificial Societies and Social Simulation*, 13:3, 2010.

[23] D. M. Kreps, P. Milgrom, J. Roberts, and R. Wilson. Rational cooperation in the finitely repeated prisoners' dilemma. *Journal of Economic Theory*, 27(2):245 – 252, 1982.

[24] F. Lau, S. Rubin, M. Smith, and L. Trajkovic. Distributed denial of service attacks. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 2275 –2280 vol.3, 2000.

[25] J. Li, E. Xiao, D. Houser, and P. R. Montague. Neural responses to sanction threats in two-party economic exchange. *Proc Natl Acad Sci U S A*, 106(39):16835–40, 2009.

[26] N. Maudet and B. Chaib-Draa. Commitment-based and dialogue-game-based protocols: new trends in agent communication languages. *Knowl. Eng. Rev.*, 17:157–179, June 2002.

[27] E. Ostrom, J. Walker, and R. Gardner. Covenants with and without a sword: Self-governance is possible. *The American Political Science Review*, 86(2):404–417, 1992.

[28] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24:33–60, September 2005.

[29] B. T. R. Savarimuthu, S. Cranefield, M. A. Purvis, and M. K. Purvis. Obligation norm identification in agent societies. *Journal of Artificial Societies and Social Simulation*, 13(4):3, 2010.

[30] D. Villatoro, G. Andrighetto, R. Conte, and J. Sabater-Mir. Dynamic sanctioning for robust and cost-efficient norm compliance. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*., 2011.

[31] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

A Reinforcement Learning Approach to Common-Interest Continuous Action Games

Abdel Rodríguez*
abrodrig@vub.ac.be
Computational Modeling Lab
Vrije Universiteit Brussel
Brussels, Belgium

Peter Vrancx
pvrancx@vub.ac.be
Computational Modeling Lab
Vrije Universiteit Brussel
Brussels, Belgium

Ricardo Grau
rgrau@uclv.edu.cu
Center of Studies in Informatics
Universidad Central "Marta Abreu" de Las Villas
Villa Clara, Cuba

Ann Nowé
ann.nowe@vub.ac.be
Computational Modeling Lab
Vrije Universiteit Brussel
Brussels, Belgium

ABSTRACT

Learning automata are reinforcement learners belonging to the category of policy iterators. They have already been shown to exhibit nice convergence properties in discrete action games. Recently, a new formulation for a Continuous Action Reinforcement Learning Automaton (CARLA) was proposed. In this paper we study the behavior of these CARLA in continuous action games and propose a novel method for coordinated exploration of the joint-action space. Our method allows a team of independent learners, using CARLA, to find the optimal joint action in common interest settings. We first show that a set of agents using CARLA will converge to a local optimum in a continuous action game. Using this property, we then introduce a method for coordinating exploration to find the global optimum of the game. We also validate our approach in a number of experiments.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms

Keywords

Multi-agent learning, Reinforcement Learning, Teamwork, coalition formation and coordination, Implicit Cooperation

1. INTRODUCTION

In this paper we present a reinforcement learning technique based on Learning Automata (LA). LA are policy iterators, which have shown good convergence results in discrete action games with independent learners. The approach presented in this paper allows LA to deal with continuous action spaces.

In their book [10], Thathachar and Sastry introduced a formulation of an LA for continuous action games known as Continuous Action Learning Automata (CALA). An alternative implementation of LA for continuous action games is the Continuous Action Reinforcement Learning Automaton (CARLA), introduced by Howell et. al. in [6]. The CALA implementation is inefficient from a sampling point of view, since it needs to sample both the selected action and the action corresponding to the mean of the current strategy. This requirement however, is infeasible in many practical settings. The original CARLA implementation has no sampling handicap, but it is much more demanding from the computational point of view. Recently, Rodríguez et al. [9] performed an analysis of the CARLA algorithm. The result of this analysis was an improvement of the CARLA method in terms of computation effort and local convergence properties. The improved automaton performs very well in single agent problems, but still has suboptimal performance with regards to global convergence in multi-agent settings.

The CARLA algorithm has successfully been used for controlling vehicles [6, 5]. However in real world applications such vehicles can be coupled with an implement for a common interest task. The interacting dynamics will produce convergence to suboptimal solutions if the subsystems are controlled ignoring the existence of each other. In such a situation a better exploration of the joint-action space is required.

Exploring Selfish Reinforcement Learning (ESRL), introduced by [16], is a better exploration method for the joint-action space of a set of independent LA playing a repeated discrete action game. The supporting idea of this method is that a set of LA will converge to one of the Nash equilibria of the game, but not necessarily one from the Pareto front. ESRL proposes that once the agents converge to a Nash equilibrium, the learners should delete the selected action from their action spaces and restart learning. This allows the agents to find all dominant equilibria and agree on the best one. As the more interesting Nash equilibria are often also stronger attractors, the agents can quite efficiently reach Pareto optimal Nash equilibria. The problem with applying this approach in continuous action games, is that it makes no sense for the agents to delete a single actions. Instead, a vicinity around the action should be identified and excluded.

This paper introduces an extension of the ESRL method to continuous action games. Standard definitions of random variables and

*the first author is also a member of the Center of Studies in Informatics, Universidad Central "Marta Abreu" de Las Villas

their numerical properties [8] will be used. Section 2 will introduce necessary background information. The convergence analysis of CARLA when playing games will be introduced in section 3 and the need for a better exploration of the joint-action space will be demonstrated. The extension of the discrete action ESRL method to continuous action games will then be introduced in section 4 showing an adequate way of identifying the whole region to exclude from the local information that the agents have. Finally, sections 5 and 6 will show the experimental results and conclusions.

2. BACKGROUND

Three important topics will be addressed in this section. First, we will refer to some standard definitions when dealing with continuous action games in subsection 2.1, then the CARLA method is introduced in subsection 2.2 and finally we summarize ESRL in subsection 2.3.

2.1 Continuous action games

Some definitions are necessary before introducing our approach. We will refer to the action space of agent i as $A^{(i)}$ so the joint-action space on n players is $A^{(1)} \times \cdots \times A^{(n)}$. In this paper we are going to use action spaces that are compact subsets of $\Re$. The strategy of each player is a probability measure on $A^{(i)}$ and will be denoted as $f^{(i)}\left(a^{(i)}\right)$ for all $a^{(i)} \in A^{(i)}$. We will refer to the space of all possible strategies by $\mathcal{F}^{(1)} \times \cdots \times \mathcal{F}^{(n)}$. The player's feedback or reward is $\beta^{(i)}\left(a^{(1)}, \cdots, a^{(n)}\right)$. Our work will be focussed on common interest games so we can simply call that reward $\beta\left(a^{(1)}, \cdots, a^{(n)}\right)$. Although all agents share the same reward function, the expected reward is not the same for all of them since this expectation is dependent on their individual strategies. The expected reward for a given player with respect to the policies of the other agents $\bar{\beta}^{(i)}\left(a^{(i)}\right) \left| f^{(1)}, \cdots, f^{(i-1)}, f^{(i+1)}, \cdots, f^{(n)} \right.$ can be calculated as shown in (1). Notice that this expectation for every agent i is based on the strategy of all other agents. When we refer to this expected value given the current strategies of the players we will use the short notation $\bar{\beta}^{(i)}\left(a^{(i)}\right)$.

$$\begin{array}{c} \bar{\beta}^{(i)}\left(a^{(i)}\right) \left| f^{(1)}, \cdots, f^{(i-1)}, f^{(i+1)}, \cdots, f^{(n)} \right. = \\ \int_{A^{(1)}} \cdots \int_{A^{(i-1)}} \int_{A^{(i+1)}} \cdots \int_{A^{(n)}} \beta\left(a^{(1)}, \cdots, a^{(n)}\right) df^{(n)}\left(a^{(n)}\right) \\ \cdots, df^{(i+1)}\left(a^{(i+1)}\right) df^{(i-1)}\left(a^{(i-1)}\right) \cdots, df^{(1)}\left(a^{(1)}\right) \end{array} \quad (1)$$

The expected reward when using strategy $f^{(i)}$ is defined in (2).

$$\bar{\bar{\beta}}^{(i)} \left| f^{(i)} \right. = \int_{A^{(i)}} \bar{\beta}^{(i)}\left(a^{(i)}\right) df^{(i)}\left(a^{(i)}\right) \quad (2)$$

As a Nash equilibrium in discrete action games we use locally superior strategies [15] which is formally introduced in definition 2.1. A vicinity is defined based on a function $d^{(i)} : \mathcal{F}^{(i)}\left[A^{(i)}\right] \times \mathcal{F}^{(i)}\left[A^{(i)}\right] \to [0,\infty]$ of which we only assume that $d^{(i)}\left(f^{(i)}, f^{(i)}\right) = 0$ and $\forall_{f^{(i)},q^{(i)}\ in \mathcal{F}^{(i)}, f^{(i)} \neq q^{(i)}} : d^{(i)}\left(f^{(i)}, q^{(i)}\right) > 0$, which implies that $d^{(i)}$ not necessarily is a distance. $V^{(i)}$ is a vicinity of $f^{(i)}$ with respect to $d^{(i)}$ if and only if there is a $\delta > 0$ such that $V^{(i)} = \left\{ q^{(i)} \in \mathcal{F}^{(i)} | d^{(i)}\left(q^{(i)}, f^{(i)}\right) < \delta \right\}$.

DEFINITION 2.1. *$f^{(i)}$ is locally superior with respect to $d^{(i)}$ if it has a vicinity $V^{(i)}$ with respect to $d^{(i)}$ such that $\forall_{q^{(i)} \in V, q^{(i)} \neq f^{(i)}}$: $\bar{\bar{\beta}}^{(i)} \left| f^{(i)} \right. > \bar{\bar{\beta}}^{(i)} \left| q^{(i)} \right.$*

We will refer to the most likely joint-action of a locally superior strategies as a local attractor.

2.2 Continuous action reinforcement learning automata

The learning automaton is a simple model for adaptive decision making in unknown random environments. The original concept of an LA originated in the domain of mathematical psychology [1] where it was used to analyze the behavior of human beings from the viewpoint of psychologists and biologists [3, 4].

The engineering research on LA started in the early 1960's [11, 12]. Tsetlin and his colleagues formulated the objective of learning as an optimization of some mathematical performance index[1] [10, 13, 14]:

$$\bar{\beta} = \int_A \beta(a)\, \mathrm{d}f(a) \quad (3)$$

The only assumption on A is that it has to be a compact set. The performance index $\bar{\beta}$ is the expectation of β with respect to the distribution f. This distribution includes randomness in a (probabilistic action selection) and randomness in β (rewards can be stochastic).

LA are useful in applications that involve optimization of a function which is not completely known in the sense that only noise corrupted values of the function for any specific values of arguments are observable [10]. A standard implementation is introduced below.

The implementation we are going to use in this paper is the Continuous Action Reinforcement Learning Automata (CARLA) [6] because of its applicability to real-world problems [9]. In [6] f is implemented as a nonparametric probability density function (PDF). Starting with the uniform distribution over the whole action space A and after exploring action $a_t \in A$ in time step t the PDF is updated as shown in (4). Parameters α and λ are refered to as learning and spreading rate respectively. The first one determines the importance of new observations, the higher the rate the bigger the change in the PDF from one time-step to another. The second parameter determines how much to spread the information of a given observation to the neighboring actions. Finally, γ is a standardization factor so $\int_{-\infty}^{+\infty} f_{t+1}(z)\, dz = 1$

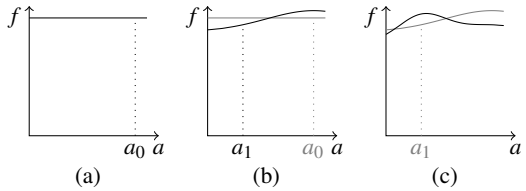
$$f_{t+1}(a) = \begin{cases} \gamma_t \left(f_t(a) + \beta_t(a_t)\, \alpha e^{-\frac{1}{2}\left(\frac{a-a_t}{\lambda}\right)^2} \right) & a \in A \\ 0 & a \notin A \end{cases} \quad (4)$$

Figure 1 shows a sketch of the CARLA action space exploration. At time-step t_0 (Figure 1.a) the learner's strategy f is the uniform PDF and action a_0 is drawn from it. Then f is updated by adding a Gaussian bell around a_0 and normalized again as shown in Figure 1.b. From the new strategy f the action a_1 is drawn and the strategy is again updated as shown in 1.c. In this example both actions (a_0 and a_1 received positive rewards so the probabilities were positively reinforced around them).

Every time the agent selects an action, this action will be drawn stochastically according to the PDF. Therefore the cumulative density function (CDF) is required [8]. Since the PDF used in the

[1]The original formulation is $J(\Lambda) = \int_\Lambda R(\mathbf{a}, \Lambda)\, \mathrm{d}f(\mathbf{a})$ but for consistency reasons, we adapted this formulation to the notation used in this paper.

Figure 1: Sketch of CARLA action space exploration. The solid line represents the current PDF while the gray line represents the previous PDF



CARLA method is nonparametric, it is computationally expensive to generate the CDF every time the function changes. The method used in this paper for the CDF is the approximation introduced in [9] shown in expression (5). Notice that $F_{N(0,1)}$ is the standardized normal cumulative density function. Also notice that the λ parameter is now dynamically changing over time. This is also introduced in [9] as an improvement of the method to speed-up the convergence. The spreading rate used every time-step is calculated as a factor of the original parameter λ_0 and a convergence measure based on the standard deviation of the last n actions σ_t.

$$F_{t+1}(a) = \begin{cases} 0 & a < \min(A) \\ \gamma_t \left(F_t(a) + \delta_t D_t(\min(A), a)\right) & a \in A \\ 1 & a > \max(A) \end{cases}$$
$$\text{where:}$$
$$\delta_t = \beta_t(a_t)\,\alpha\lambda_t\sqrt{2\pi}$$
$$\gamma_t = \frac{1}{1+\delta_t D_t(\min(A),\max(A))}$$
$$D_t(x,y) = F_{N(0,1)}\left(\frac{y-a_t}{\lambda_t}\right) - F_{N(0,1)}\left(\frac{x-a_t}{\lambda_t}\right)$$
$$\lambda_t = \lambda_0 \frac{12\sigma_t}{(\max(A)-\min(A))^2} \tag{5}$$

2.3 Exploring selfish reinforcement learners

In this subsection we will briefly explain exploring selfish reinforcement learning[2] [16]. The main characteristic of ESRL is the way the agents coordinate their exploration. Phases in which agents act independently and behave as selfish optimizers (exploration phase) are alternated by phases in which agents are social and act so as to optimize the group objective (synchronization phase). During every exploration phase the LA will converge to a pure Nash equilibrium. In the synchronization phase the solution converged to in the previous exploration phase is evaluated and removed from the joint-action space. In this way, new attractors can be explored. The removal is achieved by letting at least two agents exclude their private action that is part of the solution found from its private action space. Agents alternate between exploration and synchronization phases to efficiently search a shrinking joint-action space in order to find the Pareto optimal Nash equilibrium.

In common interest games at least one Pareto optimal Nash equilibrium exists. Despite the existence of an optimal solution, there is no guarantee that independent learners converge to such an equilibrium. Only convergence to (possibly suboptimal) Nash equilibria can be guaranteed [2]. Games such as the climbing game [7], shown in Figure 2 are generally accepted as a hard coordination problem for independent learners[3]. In this game, independent learners may get stuck in the suboptimal Nash equilibrium

[2]There is a conflicting interest version of ESRL as well but we only refer to the common interest for it is the one relevant to our work in this paper. Future work will be focussed in this direction

[3]ESRL can also deal with stochastic payoff games but for simplicity of the explanation we use a deterministic example

Figure 2: The climbing game, a common interest game. The Pareto optimal Nash equilibrium (a_{11}, a_{21}) is surrounded by heavy penalties

	a_{21}	a_{22}	a_{23}
a_{11}	11	-30	0
a_{12}	-30	7	6
a_{13}	0	0	5

Figure 3: The climbing game after shrinking the joint-action space. The Pareto optimal Nash equilibrium (a_{11}, a_{21}) is still not the only one Nash equilibrium but it is now the most likely to find

	a_{21}	a_{23}
a_{11}	11	0
a_{13}	0	5

with payoff 7. Whether the agents reach the Pareto optimal Nash equilibrium with payoff 11 mainly dependes on the initialization of the action probabilities when learning starts. The agents may be tempted to play the safe, sub-optimal actions, resulting in joint action (a_{13},a_{23}) with payoff 5.

If the Pareto optimal Nash equilibrium is reached, there is no room for improvement. However if this Pareto optimal Nash equilibrium is not reached, all agents will benefit from searching for it in the long run. Suppose for example that the agents of Figure 2 converged to joint action (a_{12},a_{22}) corresponding to a payoff of 7 for both agents. Independent learners will not have the tendency to explore further to reach the better Nash equilibrium, with payoff 11 for both agents, because of the high punishment of -30 that surrounds this interesting Nash equilibrium.

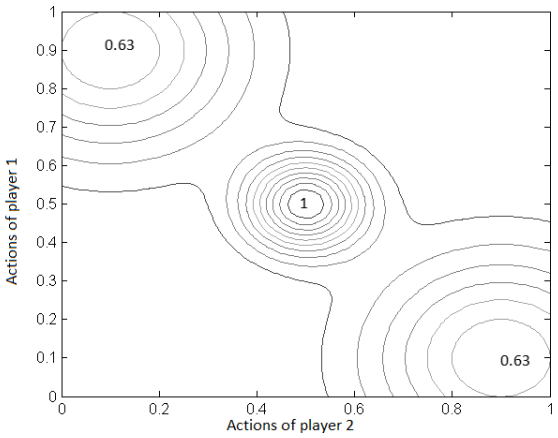
In order not to get too much punishment before discovering the best joint action (a_{11},a_{21}), the agents should coordinate their exploration. This means that the agents should decide together to search for a better Nash equilibrium. In ESRL this is achieved by making the agents exclude the action they converged to and then restarting learning in the remaining action subspace, which will be much smaller. In the case of the climbing game presented before, the joint-action space shrinks from 9 to 4 joint actions as shown in Figure 3. Again the learning itself can be performed in an independent manner, and this will drive the agents toward another Nash equilibrium. The average payoff received during this second period of learning can be compared with the average payoff received during the first period of learning, and as such the two Nash equilibria found can be compared against each other. This alternation between independent learning and excluding actions can be repeated until each has exactly one action left.

3. CONVERGENCE OF CARLA

The convergence proof for a single agent CARLA was introduced in [9]. A similar analysis for continuous action games is given in appendix A. We show that a set of CARLA will converge to a local attractor in the joint-action space as long as the parameter α is significantly low for the learners to match their expected strategies through time. Which of the multiple attractors is selected will depend on the initialization of the parameters.

In order to better understand let us analyze the following example. The analytical expression of the reward signal is shown in (20) in appendix B and its contour representation in Figure 4. There are three attractors in this example. The two local maxima located in the top left and bottom right corners have larger basins of attraction while the global maximum at the center has a narrower basin of

Figure 4: A two players game with three local optima. The contours represents combination of actions with the same reward (also see appendix B). Notice that the global optimum is located in a narrower region, this region does not overlap with the other maxima from both agents point of view.



attraction.

As shown in appendix A if both learners start with the uniform distribution we expect them to converge to the attractor that fits expression (6) the best.

$$\forall_{a\in A^{(i)}} : \int_{-\infty}^{+\infty} \bar{\beta}_t^{(i)}(z)\, e^{-\frac{1}{2}\left(\frac{a_*^{(i)}-z}{\lambda_t^{(i)}}\right)^2} \mathrm{d}z \geq \int_{-\infty}^{+\infty} \bar{\beta}_t^{(i)}(z)\, e^{-\frac{1}{2}\left(\frac{a-z}{\lambda_t^{(i)}}\right)^2} \mathrm{d}z \quad (6)$$

Since we are only interested in the effect of λ we will use a fixed value over all time-steps. Two λ values will be explored: 0.04 and 0.007. The learners are expected to converge to either one of the attractors to the corners (either (0.1,0.9) or (0.9,0.1)) when using $\lambda = 0.04$ and to the global maximum (0.5,0.5) when $\lambda = 0.007$. In order to prevent the learners from deviating too much from their expected strategies, a very low α is used in this case 0.005. In a real setup we will never use such a low value. Because of the low learning rate we run 100 experiments for 30000 time-steps. When using $\lambda = 0.04$, half of the times the learners converged to the attractor (0.1,0.9) while the other half they converged to (0.9,0.1). In the second case, when using $\lambda = 0.007$, both agents converged to the global maximum.

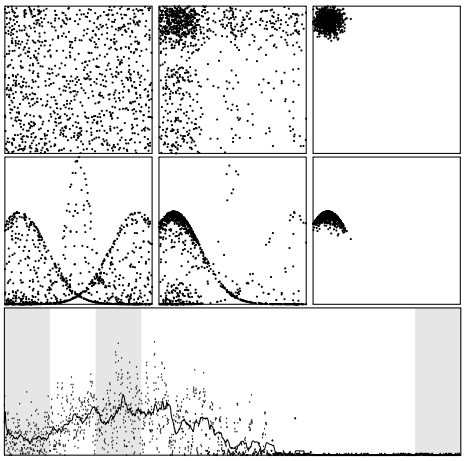
We can conclude that in order to make the learners converge to the best attractor we either need information about the reward function or a better exploration of the joint-action space, as we will introduce in the next section.

4. ESCARLA

In order to introduce Exploring Selfish Continuous Action Reinforcement Learning Automaton (ESCARLA) let us recall ESRL introduced for discrete action games for improving global performance. The supporting idea was to exclude actions after every exploration phase. The problem when facing continuous action games is that it makes no sense that the agents delete one of their actions. Instead we consider removing a neighborhood of the action. Now the agent must estimate when it crosses the boundary of the basin of attraction of the local attractor.

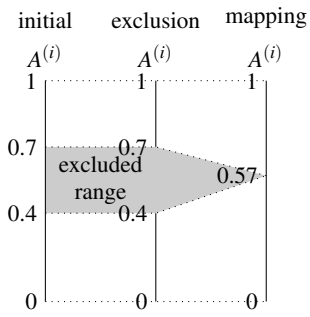
In order to solve this problem we propose to use the absolute value of the covariance between the actions and rewards as a metric. Figure 5 shows the relation between the exploration and the covariance between actions and rewards from a single agent point of view. The parameter setting used for this example is the same that the one selected in section 3. The first row shows a global view of the exploration. Three time intervals are shown. The first interval is the start of learning (time-steps from 0 to 1000). The second interval is when the learners are leaving global exploration (time-steps from 2000 to 3000). Notice this is a good time for deciding on the extension of the neighborhood to exclude for the learners are converging to a locally superior strategy and are trapped within its basin of attraction. The last interval selected is when agents have converged to the local attractor (time-steps from 9000 to 10000). The second row shows the local information that the independent agents take into account. The same time-steps are represented on each column but in this case we are plotting the selected actions by horizontal axis and the corresponding reward by the vertical axis. The bottom row shows the absolute value of the covariance between actions and rewards over the whole learning process. Additionally, in order to have a better idea of how this covariance is evolving, the solid curve represents its average. The time-steps corresponding to the three moments introduced above are shaded in gray. This covariance reaches a low value at the beginning of learning since lots of explorations are performed by both agents. When the agents are exploring within the basin of attraction of a local attractor then the noise in the rewards observed by each agent is minimal so the covariance reaches its maximum. As agents converge to a locally superior strategy, less exploration is performed so therefore the covariance value drops down to zero. The safe region to exclude after the agent's actions have converged to the local optimum, can therefore be estimated at the moment when the absolute value of the covariance reaches its maximum value.

Figure 5: Relation between covariance and exploration. The row on top represents the joint action of the agents. The second row represents the reward collected for each action by the fist agent. The left-most column collects samples from time-step 0 to 1000. The column in the center collects samples from time-step 2000 to 3000. The right-most column collects samples from time-step 9000 to 10000. The last row shows the absolute value of the covariance between actions and rewards for the first agent through the 10000 time-steps. The intervals corresponding to 0 to 1000, 2000 to 3000 and 9000 to 10000 are shaded.



A good way of estimating this region is using the percentiles of the probability density function of the actions. For a given confidence value c we can define a region as shown in expression (7) where percentile(p, f) represents the value where the probability density function f accumulates the probability p. Notice that the

Figure 6: Mapping process. The new action space will stay as the previous one but actions will be mapped in order to avoid the deleted range (0.4,0.7)



lower the c the wider the region defined by expression (7), but also the lower the probability for the actions in the region to have been properly explored.

$$\left[\text{percentile}\left(\frac{1-c}{2},f\right),\text{percentile}\left(1-\frac{1-c}{2},f\right)\right] \qquad (7)$$

The proposal here is to let the agents start learning until they all reach a threshold value of convergence $conv_{stp}$. Then each agent should delete the region defined by (7) from its action space – examples in the experimental results section will use $conv_{stp} = 0.95$ and $c = 0.9$. Deleting the action range implies modifying the action space so we need to map the new one into a compact set again as shown in Figure 6. This method can introduce some noise in the reward function but since CARLA learners do not require continuity in the reward function it will not be a problem. Then all agents must restart the learning process to converge to another attractor. After enough exploration the agents should compare all the results and pick up the strategy that gave them the highest score. In engineering applications we may either know the total amount of maxima of the problem or the desired performance. In such applications the agents could understand by enough exploration by finding all different maxima of the problem or by achieving the desired performance. Please note that we are assuming a common interest game, so therefore the agents can agree on a best combination of actions. The general algorithm is given next.

```
ESCARLA algorithm
    repeat
        explore
        synchronize
    until enough exploration
    select best strategy
```

```
Exploration phase
    initialize parameters
    repeat
        sample action
        update strategy
        if maximum covariance
        then mark interval
    until convergence
```

```
Synchronization phase
    exclude marked interval
```

Figure 7: Characteristic functions β^i and β^{ii}. Notice in the first game that the global optimum is located in a very narrow region, but this region overlaps with the other maximum in both axis (first and second agent's axis). The second game is even more complex since it also has a very narrow optimum region but it does not even overlap with the other maxima.

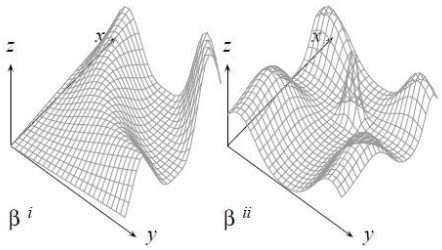
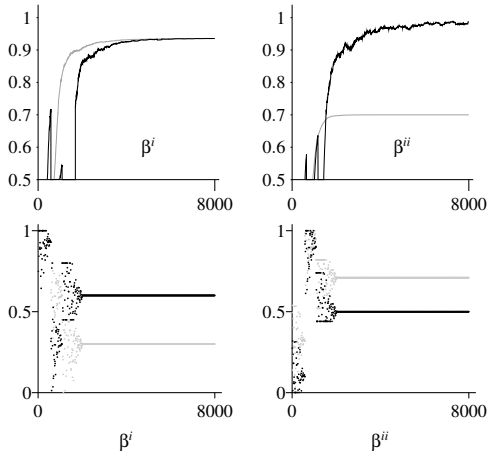


Figure 8: Average rewards for β^i and β^{ii}. For the first game, the improvement just delayed the convergence, but still the learners converged to the global optimum. On the other hand, for the second game, the improvement clearly led the agents to the global optimum in the third exploration restart. Average reward is shown on top while the actions are shown on bottom



5. EXPERIMENTAL RESULTS

All examples will be introduced by the characteristic function form [17]. Formally, a characteristic function form game is given as a pair (N,v), where N denotes a set of players and $v : A^N \longrightarrow \Re$ is a characteristic function that the agents have to optimize with $A \subset \Re$ being the action space.

In this section, two 2-agent games will be explored. The characteristic functions for them are β^i and β^{ii}. Analytical expressions are introduced in (20) in appendix B. These two games are introduced as complex examples where the basins of attractions are overlaping from both agent point of views. These functions are plotted in Figure 7.

Figure 8 shows the average rewards collected over time on top and the actions selected every time-step on bottom. The selected α was 0.1 while λ_0 was set to 0.2. The gray curve shows the result using the CARLA method. The black curve shows the results when using the ESCARLA. Notice that every peak in the rewards implies a restart in the learning. Although a bit slower, the ESCARLA obtains more accurate convergence to the global optimum.

Figure 9 shows the regions elimination process for the second game. In the phase one the agents converged to the joint action $(0.32, 0.10)$ and deleted the region ($[0.04, 0.52]$, $[0.01, 0.44]$) which is shown in black. In the second phase they converged to $(0.95, 0.9)$

Figure 9: Regions elimination process. While it was very unlikely to find the global optimum because it was a very narrow region, when the agents ended the first exploration and restart it again, it was not that unlikely anymore. After the second restart, it was inevitable to find it.

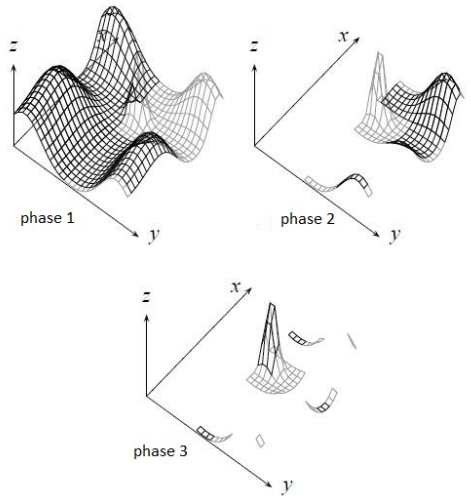


Table 1: Convergence to the global optimum per exploration phase

1^{st} phase	2^{nd} phase	3^{rd} phase	4^{th} phase	total percent
0	0	31	1	64%

and deleted the region ([0.82,0.99],[0.74,0.99]). Finally – phase three –, they easily converged to (0.71,0.5).

We would like to stress that by squaring the first agent actions in the second game (see equation (20) on appendix B) makes a distortion in the reward function so the basin of attraction of the local attractor are not symmetric (roles of agents are not interchangeable). Also notice that the regions eliminated by this first agent are not homogeneous. Since the extension of the above mentioned basin of attractions affects directly the performance, then it also affects indirectly the covariance between actions and rewards. Summing up, the covariance not only marks the time where the rewards can be trusted (in the sense that the variance is due to the stochastic nature of the reward and not due to the exploration of other agents) but it also gives the agents a good measure of the boundary of the regions where the equilibria are locally superior, i.e. the basins of attraction.

Given the stochastic nature of the optimization process and the interaction of both agents we cannot be sure that the agents will always find the global optimum after as many exploration phases as maxima in the game. Two problems may arise, either the global optimum could be excluded with a suboptimal strategy or the range of actions excluded in a synchronization phase could cover just a partial subset of the basin of attraction so probably an extra exploration phase will be necessary. In order to illustrate this issue table 1 shows how many times the global optimum was found in every exploration phase for the second game after 50 trials. Notice that the CARLA method results are the same as the ESCARLA in the first exploration phase.

5.1 Statistical significance

The learners performed well in the explored games but, what should we expect from them in new ones? To answer this question a stronger statistical test is necessary.

Table 2: Average linear error. Every row shows the average difference between the long-run reward and the current maximum reward. First row for the previous CARLA, second using the ESCARLA method. ESCARLA considerably reduces this difference for multi-agent games while it makes no harm in single-agent. Last column shows the Wilcoxon test significance

	CARLA	ESCARLA	sig.
β_1	0.0049	0.0146	0.352
β_2	0.0092	0.0100	0.746
β_3	0.0178	0.0171	0.515
β_4	0.0851	0.0365	0.001

Random functions will be generated for the characteristic function for the games. The functions will be generated by the union of Gaussian bells. Single agent functions will use 1 or 2 bells while multi-agent functions will use between 1 and 6 of these bells. The number of bells is generated at random. The parameters of the bells $\mu \in [0,1]$ and $\sigma \in [0.01,1]$ are generated randomly too, as well as the factor $f \in [0.5,1]$ for which are multiplied the bells to make them different in amplitude. An extra parameter $r \in [0.75,0.95]$ is generated for introducing noise in single agent settings. Two ways for introducing noise in single agent setting are explored with these functions, adding noisy bells or randomly selecting between them. The analytical expressions are introduced in (20) in appendix B

50 games were generated from each type of function. The average linear difference between the global maximum of the function and the final average reward collected by agents over 10000 timesteps are shown in table 2. The same parameter settings from the previous tests are used. A Wilcoxon test shows that the ESCARLA performs better than the original method for multi-agent games. The results of this test can be seen in the last column of the table 2.

6. CONCLUSIONS

Learning automata are reinforcement learners, belonging to the category of policy iterators, that exhibit nice convergence properties in discrete action settings. It has been shown that a set of agents applying independently from each other an LA update scheme can converge to a Nash equilibrium or even the Pareto optimal Nash equilibrium in a discrete action game. We extended this performance result to continuous action games by means of the extension of discrete ESRL to continuous action games improving its global performance.

Acknowledgements

This paper was developed under the Cuba-Flanders collaboration within the Flemish Interuniversity Council (Vlaamse Interuniversitaire Raad/VLIR) project.

7. REFERENCES

[1] R. Bush and F. Mosteller. *Stochastic Models for Learning*. Wiley, 1955.

[2] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *In Proceedings of National Conference on Artificial Intelligence (AAAI-98*, pages 746–752, 1998.

[3] E. Hilgard. *Theories of Learning*. New York: Appleton-Century-Crofts, 1948.

[4] E. Hilgard and G. Bower. *Theories of Learning*. New Jersey: Prentice Hall, 1966.

[5] M. Howell and M. Best. On-line pid tuning for engine idle-speed control using continuous action reinforcement

learning automata. *Control Engineering Practice*, 8(2):147 – 154, 2000.

[6] M. N. Howell, G. P. Frost, T. J. Gordon, and Q. H. Wu. Continuous action reinforcement learning applied to vehicle suspension control. *Mechatronics*, 7(3):263 – 276, 1997.

[7] S. Kapetanakis, D. Kudenko, and M. Strens. Learning to coordinate using commitment sequences in cooperative multiagent-systems. In *in Proceedings of the Third Symposium on Adaptive Agents and Multi-agent Systems (AAMAS-03*, page 2004, 2003.

[8] E. Parzen. *Modern Probability Theory And Its Applications*. Wiley-interscience, wiley classics edition edition, December 1960.

[9] A. Rodríguez, R. Grau, and A. Nowé. Continuous action reinforcement learning automata. performance and convergence. In J. Filipe and A. Fred, editors, *In Proceedings of the Third International Conference on Agents and Artificial Intelligence*, pages 473–478. SciTePress, January 2011.

[10] M. Thathachar and P. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.

[11] M. Tsetlin. The behavior of finite automata in random media. *Avtomatika i Telemekhanika*, pages 1345–1354, 1961.

[12] M. Tsetlin. The behavior of finite automata in random media. *Avtomatika i Telemekhanika*, pages 1210–1219, 1962.

[13] Y. Z. Tsypkin. *Adaptation and learning in automatic systems*. New York: Academic Press, 1971.

[14] Y. Z. Tsypkin. *Foundations of the theory of learning systems*. New York: Academic Press, 1973.

[15] M. Veelen and P. Spreij. Evolution in games with a continuous action space. *Economic Theory*, 39(3):355–376, June 2009.

[16] K. Verbeeck. *Coordinated Exploration in Multi-Agent Reiforcement Learning*. PhD thesis, Vrije Universiteit Brussel, Faculteit Wetenschappen, DINF, Computational Modeling Lab, September 2004.

[17] J. von Neumann and O. Morgenstern. Theory of games and economic behavior. 1944.

APPENDIX

A. CARLA CONVERGENCE IN CONTINUOUS ACTION GAMES

The analysis will be performed for normalized reward signals $\beta : \Re \to [0,1]$ – no generality is lost because any closed interval can mapped to this interval by a linear transformation. The final goal of this analysis is to find the necessary restrictions to guarantee convergence to local optima.

The sequence of PDF updates is a Markovian process, where for each time-step t an action $a_t^{(i)} \in A^{(i)}$ is selected and a new $f_t^{(i)}$ is returned for every learner i. The expected value $\bar{f}_{t+1}^{(i)}$ of $f_{t+1}^{(i)}$ can be computed following equation (8).

$$\bar{f}_{t+1}^{(i)}(a) = \int_{-\infty}^{+\infty} f_t^{(i)}(z) f_{t+1}^{(i)}\left(a \mid a_t^{(i)} = z\right) dz \tag{8}$$

Let $\bar{\gamma}_t^{(i)}(z) = \bar{\gamma}_t^{(i)} | a_t = z$ be the expected value for $\gamma_t^{(i)}$ if $a_t^{(i)} = z$ taking into account the other agent policies, $\bar{\bar{\gamma}}_t^{(i)}$ its expected value and $\bar{\beta}_t^{(i)}(z)$ the expected value for $\beta_t^{(i)}(z)$ as shown in (9). Then (8) could be rewritten as (10).

$$\begin{aligned}
\bar{\gamma}_t^{(i)}(z) = & \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} f_t^{(0)}\left(z^{(0)}\right) \cdots f_t^{(i-1)}\left(z^{(i-1)}\right) \\
& f_t^{(i+1)}\left(z^{(i+1)}\right) \cdots f_t^{(n)}\left(z^{(n)}\right) \\
& \left(\gamma_t^{(i)} | a_t^{(i)} = z, a_t^{(0)} = z^{(0)}, \cdots a_t^{(n)} = z^{(n)}\right) dz^{(n)} \cdots dz^{(0)} \\
\bar{\beta}_t^{(i)}(z) = & \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} f_t^{(0)}\left(z^{(0)}\right) \cdots f_t^{(i-1)}\left(z^{(i-1)}\right) \\
& f_t^{(i+1)}\left(z^{(i+1)}\right) \cdots f_t^{(n)}\left(z^{(n)}\right) \\
& \beta_t^{(i)}\left(z^{(0)}, \cdots, z^{(i-1)}, z, z^{(i+1)}, \cdots, z^{(n)}\right) dz^{(n)} \cdots dz^{(0)} \\
\bar{\bar{\gamma}}_t^{(i)} = & \int_{-\infty}^{+\infty} f_t^{(i)}(z) \bar{\gamma}_t^{(i)}(z) dz
\end{aligned} \tag{9}$$

$$\begin{aligned}
\bar{f}_{t+1}^{(i)}(a) = & \int_{-\infty}^{+\infty} f_t^{(i)}(z) \bar{\gamma}_t^{(i)}(z) \\
& \left(f_t^{(i)}(a) + \alpha^{(i)} \bar{\beta}_t^{(i)}(z) e^{-\frac{1}{2}\left(\frac{a-z}{\lambda_t^{(i)}}\right)^2} \right) dz \\
= & f_t^{(i)}(a) \bar{\bar{\gamma}}_t^{(i)} + \\
& \alpha^{(i)} \int_{-\infty}^{+\infty} f_t^{(i)}(z) \bar{\gamma}_t^{(i)}(z) \bar{\beta}_t^{(i)}(z) e^{-\frac{1}{2}\left(\frac{a-z}{\lambda_t^{(i)}}\right)^2} dz
\end{aligned} \tag{10}$$

Let us have a look at the right member of the integral. $f_t^{(i)}(z)$ is multiplied by the factor composed by the normalization factor given that $a_t^{(i)} = z$, the feedback signal $\bar{\beta}_t^{(i)}(z)$ and the distance measure $e^{-\frac{1}{2}\left(\frac{a-z}{\lambda_t^{(i)}}\right)^2}$ which can be interpreted as the strength of the relation of actions a and z, the higher the value of this product, the bigger the relation of these actions. Let us call this composed factor $G_t^{(i)}(a,z)$ and $\bar{G}_t^{(i)}(a)$ its expected value at time-step t with respect to z. Then equation (10) could be finally formulated as (11).

$$\begin{aligned}
\bar{f}_{t+1}^{(i)}(a) &= f_t^{(i)}(a) \bar{\bar{\gamma}}_t^{(i)} + \alpha^{(i)} \bar{G}_t^{(i)}(a) \\
&= f_t^{(i)}(a) \left(\bar{\bar{\gamma}}_t^{(i)} + \frac{\alpha^{(i)} \bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)} \right)
\end{aligned} \tag{11}$$

The sign of the first derivative of $f_t^{(i)}$ depends on the factor $\bar{\bar{\gamma}}_t^{(i)} + \frac{\alpha^{(i)} \bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)}$ of expression (11) so it behaves as shown in (12).

$$\frac{\partial f_t^{(i)}}{\partial t} \begin{cases} < 0 & \bar{\bar{\gamma}}_t^{(i)} + \frac{\alpha^{(i)} \bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)} < 1 \\ = 0 & \bar{\bar{\gamma}}_t^{(i)} + \frac{\alpha^{(i)} \bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)} = 1 \\ > 0 & \bar{\bar{\gamma}}_t^{(i)} + \frac{\alpha^{(i)} \bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)} > 1 \end{cases} \tag{12}$$

Notice $\bar{\bar{\gamma}}_t^{(i)}$ is a constant for all $a \in A^{(i)}$ and $\int_{-\infty}^{+\infty} f_t^{(i)}(z) dz = 1$ so:

$$\begin{aligned}
& \exists_{b_1^{(i)}, b_2^{(i)} \in A^{(i)}} : \frac{\bar{G}_t^{(i)}\left(b_1^{(i)}\right)}{f_t^{(i)}\left(b_1^{(i)}\right)} \neq \frac{\bar{G}_t^{(i)}\left(b_2^{(i)}\right)}{f_t^{(i)}\left(b_2^{(i)}\right)} \Longrightarrow \\
& \exists_{A_+^{(i)}, A_-^{(i)} \subset A^{(i)}, A_+^{(i)} \cap A_-^{(i)} = \emptyset}, \forall_{a_+^{(i)} \in A_+^{(i)}, a_-^{(i)} \in A_-^{(i)}} : \\
& \left(\frac{\partial f_t^{(i)}\left(a_+^{(i)}\right)}{\partial t} > 0 \right) \wedge \left(\frac{\partial f_t^{(i)}\left(a_-^{(i)}\right)}{\partial t} < 0 \right)
\end{aligned} \tag{13}$$

From logical implication (13) it can be assured that the sign of $\frac{\partial f_t^{(i)}(a)}{\partial t}$ will be determined by the ratio $\frac{\bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)}$. Notice subsets $A_+^{(i)}$ and $A_-^{(i)}$ are

composed by the elements of $A^{(i)}$ that have not reached their value for the probability density function in equilibrium with $\bar{G}_t^{(i)}(a)$. That is, the $A_+^{(i)}$ subset is composed by all $a \in A^{(i)}$ having a value of probability density function which is too small with respect to $\bar{G}_t^{(i)}(a)$ and vice versa for $A_-^{(i)}$.

Let $a_*^{(i)} \in A^{(i)}$ be the actions yielding the highest value for every agent i at expression $\int_{-\infty}^{+\infty} \bar{\beta}_t^{(i)}(z)\, e^{-\frac{1}{2}\left(\frac{a-z}{\lambda_t^{(i)}}\right)^2} \mathrm{d}z$ as shown in (14). It is important to stress that $a_*^{(i)}$ are not the optimum of $\bar{\beta}_t^{(i)}$ given the set $f_t^{(j)}$ but the points yielding the optimal vicinity around it and defined by $e^{-\frac{1}{2}\left(\frac{a_*^{(i)}-z}{\lambda_t^{(i)}}\right)^2}$ which depends on $\lambda_t^{(i)}$.

$$\forall_{a \in A^{(i)}} : \int_{-\infty}^{+\infty} \bar{\beta}_t^{(i)}(z)\, e^{-\frac{1}{2}\left(\frac{a_*^{(i)}-z}{\lambda_t^{(i)}}\right)^2} \mathrm{d}z \geq \int_{-\infty}^{+\infty} \bar{\beta}_t^{(i)}(z)\, e^{-\frac{1}{2}\left(\frac{a-z}{\lambda_t^{(i)}}\right)^2} \mathrm{d}z \tag{14}$$

For symmetric games, that is common interest games ($\forall_{i,j} : \beta_t^{(i)} = \beta_t^{(j)}$), all $a_*^{(i)}$ will correspond initially to the same maximum in the joint-action space, if all learner start with the same parameters and an uniform PDF each one. Then (15) holds.

$$\begin{aligned}
&\forall_t : f_t^{(i)}\left(a_*^{(i)}\right) \geq f_t^{(i)}\left(a^{(i)}\right) \Rightarrow f_{t+1}^{(i)}\left(a_*^{(i)}\right) \geq f_{t+1}^{(i)}\left(a^{(i)}\right) \\
&\forall_t : \bar{\beta}_t^{(i)}\left(a_*^{(i)}\right) \geq \bar{\beta}_t^{(i)}\left(a^{(i)}\right) \Rightarrow \bar{\beta}_{t+1}^{(i)}\left(a_*^{(i)}\right) \geq \bar{\beta}_{t+1}^{(i)}\left(a^{(i)}\right) \\
&\forall_t : \bar{G}_t^{(i)}\left(a_*^{(i)}\right) \geq \bar{G}_t^{(i)}\left(a^{(i)}\right) \Rightarrow \bar{G}_{t+1}^{(i)}\left(a_*^{(i)}\right) \geq \bar{G}_{t+1}^{(i)}\left(a^{(i)}\right)
\end{aligned} \tag{15}$$

Since the first derivative of the PDF depends on $\frac{\bar{G}_t^{(i)}(a)}{f_t^{(i)}(a)}$ the value necessary for keeping $\frac{\partial f_t^{(i)}(a)}{\partial t} = 0$ is also the highest for every $a_*^{(i)}$.

Notice that the maximum update that $f_t^{(i)}\left(a_*^{(i)}\right)$ may receive is obtained when $a_t^{(i)} = a_*^{(i)}$ – centering the bell at $a_*^{(i)}$ –, then if $f_t^{(i)}\left(a_*^{(i)}\right)$ reaches the value $\frac{1}{\lambda_t^{(i)}\sqrt{2\pi}}$, its first derivative will not be higher than 0 as shows (16) since $\beta : \Re \to [0,1]$.

$$\begin{aligned}
f_{t+1}^{(i)}\left(a_*^{(i)}\right) &= \bar{\gamma}_t^{(i)}\left(a_t^{(i)}\right) \\
&\left(f_*^{(i)}\left(a_t^{(i)}\right) + \beta_t^{(i)}\left(a_t^{(i)}\right) \alpha^{(i)} e^{-\frac{1}{2}\left(\frac{a_*^{(i)}-a_t^{(i)}}{\lambda_t^{(i)}}\right)^2}\right) \\
&\leq \frac{1}{1+\alpha^{(i)}\lambda_t^{(i)}\sqrt{2\pi}}\left(\frac{1}{\lambda_t^{(i)}\sqrt{2\pi}} + \alpha^{(i)}\right) \\
&\leq \frac{1}{1+\alpha^{(i)}\lambda_t^{(i)}\sqrt{2\pi}}\left(\frac{1+\alpha^{(i)}\lambda_t^{(i)}\sqrt{2\pi}}{\lambda_t^{(i)}\sqrt{2\pi}}\right) \\
&\leq \frac{1}{\lambda_t^{(i)}\sqrt{2\pi}}
\end{aligned} \tag{16}$$

Then the equilibrium point of $f_t^{(i)}\left(a_*^{(i)}\right)$ has the higher bound $\frac{1}{\lambda_t^{(i)}\sqrt{2\pi}}$. Notice that the closer the $\beta_t^{(i)}\left(a_*^{(i)}\right)$ to 1, the closer the equilibrium point of $f_t^{(i)}\left(a_*^{(i)}\right)$ to its higher bound.

$$\frac{\partial f_t^{(i)}\left(a_*^{(i)}\right)}{\partial t} \begin{cases} < 0 & f_t^{(i)}\left(a_*^{(i)}\right) > \frac{1}{\lambda_t^{(i)}\sqrt{2\pi}} \\ = 0 & f_t^{(i)}\left(a_*^{(i)}\right) = \frac{1}{\lambda_t^{(i)}\sqrt{2\pi}} \\ > 0 & f_t^{(i)}\left(a_*^{(i)}\right) < \frac{1}{\lambda_t^{(i)}\sqrt{2\pi}} \end{cases} \tag{17}$$

We can conclude from (17) that the highest value for $f^{(i)}$ will be achieved at $a_*^{(i)}$ as shown in (18) which has the higher bound $\frac{1}{\lambda_t^{(i)}\sqrt{2\pi}}$.

$$\begin{aligned}
\forall_{a \in A^{(i)} \setminus \{a_*^{(i)}\}} &: lim_{t\to\infty} f_t^{(i)}(a) < f_t^{(i)}\left(a_*^{(i)}\right) \\
&lim_{t\to\infty} f_t^{(i)}\left(a_*^{(i)}\right) \leq \frac{1}{\lambda_t^{(i)}\sqrt{2\pi}}
\end{aligned} \tag{18}$$

Finally

$$\begin{aligned}
&lim_{\lambda^{(i)}\downarrow 0} lim_{t\to\infty} f_t^{(i)}\left(a_*^{(i)}\right) = \infty \\
\forall_{a \neq a_*^{(i)}} &: lim_{\lambda^{(i)}\downarrow 0} lim_{t\to\infty} f_t^{(i)}(a) = 0
\end{aligned} \tag{19}$$

We can safety conclude that the set of CARLA will converge to an equilibrium in the joint-action space as long as the parameter α is low enough for the learners to match their expected policies through time. Which of the multiple equilibria is selected will depend on the initialization of the parameter λ.

B. ANALYTICAL EXPRESSION OF REWARDS

$$\begin{aligned}
\beta\left(a^{(1)}, a^{(2)}\right) &= \mathrm{Bell}\left(a^{(1)}, a^{(2)}, 0.5, 0.5, 0.084\right) \\
&\bigcup 0.63\,\mathrm{Bell}\left(a^{(1)}, a^{(2)}, 0.1, 0.9, 0.187\right) \\
&\bigcup 0.63\,\mathrm{Bell}\left(a^{(1)}, a^{(2)}, 0.9, 0.1, 0.187\right) \\
\beta^{i}\left(a_t^{(1)}, a_t^{(2)}\right) &= 0.3125\left(1+\sin(9a_t^{(1)}a_t^{(2)})\right)^2 \frac{a_t^{(1)}+a_t^{(2)}}{2} \\
\beta^{ii}\left(a_t^{(1)}, a_t^{(2)}\right) &= \mathrm{Bell}\left(\left(a_t^{(1)}\right)^2, a_t^{(2)}, 0.5, 0.5, 0.002\right) \\
&\bigcup 0.7\,\mathrm{Bell}\left(\left(a_t^{(1)}\right)^2, a_t^{(2)}, 0.1, 0.1, 0.2\right) \\
&\bigcup 0.7\,\mathrm{Bell}\left(\left(a_t^{(1)}\right)^2, a_t^{(2)}, 0.9, 0.9, 0.2\right) \\
&\bigcup 0.7\,\mathrm{Bell}\left(\left(a_t^{(1)}\right)^2, a_t^{(2)}, 0.9, 0.1, 0.2\right) \\
&\bigcup 0.7\,\mathrm{Bell}\left(\left(a_t^{(1)}\right)^2, a_t^{(2)}, 0.1, 0.9, 0.2\right) \\
\beta_1(a) &= \bigcup_{i=1}^{n} f_i\, b_i(a) \\
\beta_2(a) &= \bigcup_{i=1}^{n} (r_i\, f_i\, b_i(a) + \mathrm{rand}(1-r_i)) \\
\beta_3(a) &= pick\left\{\frac{r_i}{\sum_1^n r_i}, f_i\, b_i(a)\right\} \\
\beta_4\left(a^{(1)}, a^{(2)}\right) &= \bigcup_{i=1}^{n} f_i \mathrm{Bell}_i\left(a^{(1)}, a^{(2)}\right) \\
b(a) &= e^{-\frac{(a-\mu)^2}{2\sigma^2}} \\
\mathrm{Bell}\left(a^{(1)}, a^{(2)}, \mu^{(1)}, \mu^{(2)}, \sigma\right) &= e^{-\frac{\left(a^{(1)}-\mu^{(1)}\right)^2+\left(a^{(2)}-\mu^{(2)}\right)^2}{2\sigma^2}} \\
a \bigcup b &= a+b-ab
\end{aligned} \tag{20}$$

Designing Environment-Agnostic Agents

Olivier L. Georgeon
Université de Lyon
CNRS, LIRIS, UMR5205
F-69622, France
+33 4 72 04 63 30
olivier.georgeon@liris.cnrs.fr

Ilias Sakellariou
Department of Applied Informatics
University of Macedonia of Economic and
Social Sciences, Thessaloniki, Greece
+30 2310-891-858
iliass@uom.gr

ABSTRACT
We present autonomous agents that are designed without encoding strategies or knowledge of the environment in the agent. The design approach focuses on the notion of *sensorimotor patterns of interaction* between the agent and the environment rather than separating *perception* from *action*. The agent's motivational system is also interaction-centered in that the agent has inborn proclivities to enact certain sensorimotor patterns and to avoid others. Such motivations result in the agent autonomously discovering, learning, and exploiting regularities of interaction afforded by the environment, and constructing operative knowledge of the environment. Because such agents have no predefined goals, we propose a set of behavioral criteria to both judge and demonstrate the agents' capacities, rather than performance measurement. A design platform based on NetLogo is presented. Results show that these agents demonstrate interesting behavioral properties such as hedonistic temperance, active perception, and individuation.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning. I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent agents.*

General Terms
Algorithms, Measurement, Design, Experimentation.

Keywords
Intrinsic motivation; Autonomous learning; Cognitive development; Enactive cognition; Affordances; Constructivism.

1. INTRODUCTION
This paper presents an argumentation in favor of the concept of *environmental agnosticism* as an original and useful concept to think about autonomous agents. We introduce this concept to capture the idea that we do not encode our knowledge of the environment in the agent's decisional mechanism. A typical example of such encoded knowledge would consist of a set of logical rules that specify the agent's behavior in response to specific information received from the environment (sensory input). Another example would consist of modeling the agent's environment as a predefined *problem space* in which reward values would be associated with predefined *problem states* and such values propagated across states [15][1]. Instead, we expect environment-agnostic agents to learn the semantics of sensorimotor information and the ontological structure of their world by themselves.

Our motivation for studying environment-agnostic agents is both theoretical and practical. On the theoretical level, we believe that this study can shed some light on how knowledge emerges in artificial agents and becomes meaningful to them. This question relates to the symbol grounding problem [13] and the study of developmental cognition (e.g., [28]). On the practical level, environment-agnostic agents will facilitate the development of agent-based simulations by unburdening the modeler from encoding knowledge in the agent.

As we intend to show, designing an environment-agnostic agent raises the crucial question of defining the agent's *drives*. We employ the term drive to emphasize the difference from traditional approaches that employ the terms task or goal. Indeed, we argue that programming an agent to perform a given task or to reach a given goal generally implies specifying how the agent interprets its world. For example, behavioral rules presuppose the semantics of input data, and traditional reward values presuppose goal assessment criteria. Such presuppositions conflict with the environment-agnosticism principle because environment-agnostic agents are precisely expected to learn by themselves how to interpret their world.

An intuitive distinction between drives and goals may be expressed by the fact that drives enforce a bottom-up approach starting from inborn behavioral tendencies toward the possible construction of higher-level goals, while goals follow a top-down approach through the decomposition of a problem into sub-goals. This conception of drives can also be related to the concept of *intrinsic motivation* (e.g., [2, 19]) in that the motivation does not come from an external reward. Yet, we acknowledge that the nuance may still seem vague and better pertaining to the domain of philosophy than computer science. At a philosophical level, let us only note here that we find some resonance with Dennett's *inversion of reasoning* argument [6]. The purpose of this paper is not to pursue this philosophical discussion any further but to show that this shift of viewpoint is not mere jargon and rhetoric but can have a strong impact on the way we develop autonomous agents.

Our intuition in developing environment-agnostic agents is to put the focus on the interaction between the agent and the environment rather than only on the agent. When designing

[1] The Soar architecture offers an emblematic illustration of these two types of examples as a rule-based system extended with reinforcement learning (Soar-RL). We credit the Soar team for acknowledging this knowledge-oriented bias in both cases.

autonomous agents, we, indeed, must presuppose the possible range of interactions between the agent and the environment. For example, in the case of robots, engineers define the robot's interactions when designing sensors and effectors. In the case of natural organisms, phylogenetic evolution selected the organism's sensorimotor system. In a similar manner, we predefine environment-agnostic agents' interactions in a given environment. We, however, neither presuppose nor specify how the agent should interpret such interactions. Instead, the agent has behavioral drives that define preferred courses of action in the world. The agent learns regularities of interactions and exploits such regularities in turn to better fulfill its drives. This mechanism is explained in more detail in Section 2.

The fact that the agent has no predefined goals raises the question of how to assess its behavior. This question is discussed in Section 3, and illustrated by experiments in Section 4. Section 5 presents the design/simulation platform employed for implementing environment-agnostic agents. Finally, the conclusion discusses the implications and limitations of the concept of environmental agnosticism for future work.

2. ENVIRONMENTAL AGNOSTICISM

Fundamentally, we believe designing environment-agnostic agents entails considering individual *sensorimotor patterns* as the atomic elements of cognition, without making an initial distinction between perception and action. This assumption is supported by many theories in cognitive science that argue that perception, cognition, and motion are entangled (e.g., [5, 14, 18, 21]). Specifically, Piaget [22], proposed the term *scheme* to refer to sensorimotor patterns. In the rest of this paper, we simply refer to sensorimotor patterns by the term elementary *interactions*.

Figure 1 illustrates the elementary interactions along the *interaction timeline*. Elementary interactions are represented by letters (A, B, C, A, D, A …). As introduced in Section 1, these interactions have no associated semantics implemented in the agent. Because of the absence of implemented semantics, these interactions can correspond to anything in the environment, which is why we characterize the agent as environmentally agnostic. As opposed to traditional artificial agents, this agent has no channel by which it would directly "perceive" its environment, but can only discover the structure of the environment through the regularities experienced while enacting these interactions.

2.1 Intrinsic drives

In addition to focusing on elementary interactions, we propose implementing inborn *proclivity values* associated with such interactions (in parenthesis in Figure 1). Accordingly, we provide the agent with a mechanism that tends to seek interactions with positive proclivity values, and to avoid interactions with negative proclivity values. Such a mechanism results in the implementation of primitive intrinsic drives because, subjectively, the agent seems to simply enjoy interactions that have positive proclivity values, and to dislike interactions that have negative proclivity values. As we present later, the difficulty resides in that the agent needs to learn what interactions will result from its choices.

Proclivity values are related to the notions of intrinsic reward (e.g., [25]) and value systems (e.g., [21, 26, 27]) in reinforcement learning. Because of this relation, our approach can be considered a form of *discrete time decision process* consisting of learning a *policy function* that tends to maximize a *reward function* over time. Traditional algorithms of discrete time decision processes, however, require assumptions incompatible with the agnosticism principle. For example, Markov Decision Processes (MDP) algorithms require that the temporal dependency be known a priori, and that the environment be modeled in the form of states that the agent can directly recognize. Partially Observable Markov Decision Processes (POMDP) [1] algorithms offer a way toward eliminating these hypotheses but they require a *state evaluation function* to assesses a *believed state* from observational data. The

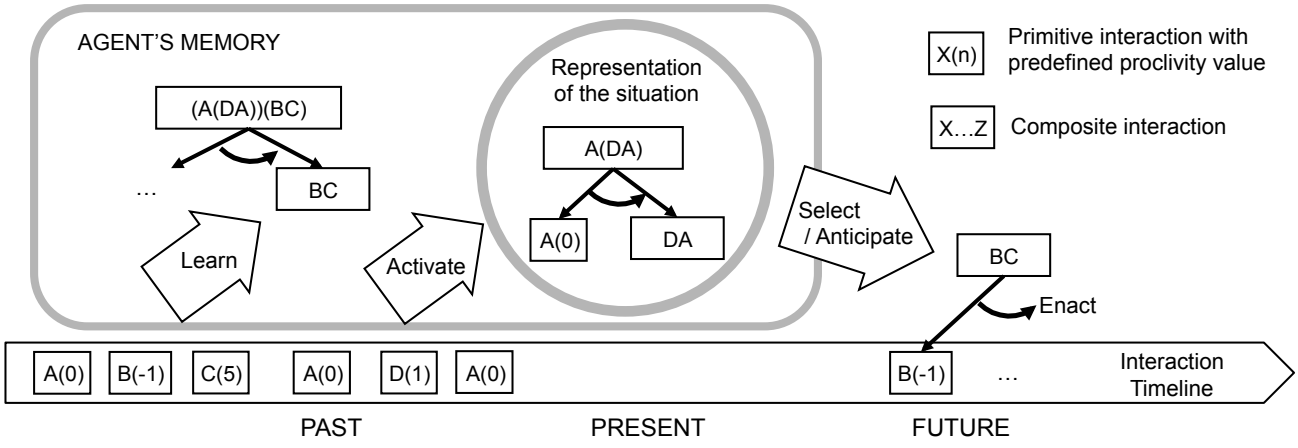


Figure 1. Learning and decisional mechanism of an environment-agnostic agent.

The agent's activity is represented along the *interaction timeline*. Letters (A, B, C, etc.) represent *primitive interactions*. Primitive interactions are associated with proclivity values pre-defined by the designer (e.g., (0), (-1), (5), etc.). Through its activity, the agent learns hierarchical sequences of interactions (*composite interactions*) that capture hierarchical regularities of interaction with a given environment (e.g., (A(DA))(BC)). The agent represents its current situation in the form of these hierarchical composite interactions. Over time, learned composite interactions allow the agent to predict the consequences of its choices depending on the current situation, and, therefore to select interactions that have the best chance to maximize the agent's proclivity in a given environment (e.g., BC). The agent can learn to enact unsatisfying interactions (e.g., B(-1)) to reach situations where it can enact even more satisfying interactions (e.g., C(5)).

state evaluation function needs to be known a priori, which remains incompatible with the agnosticism principle.

Here, we propose the term proclivity because this term conveys the idea that interactions are enacted for their own sake, while the term reward suggests that interactions are enacted for the sake of their outcome. Enacting interactions for their own sake removes the need for a pre-assumed model of the world. With proclivity, knowledge follows from action, while, with reward, action follows from knowledge. Our approach based on proclivity values relates more to continuous case-based reasoning [23] or trace-based reasoning [17] (but unsupervised, with no presupposed knowledge) than to traditional reinforcement learning, as we will develop in Section 2.2.

Notably, we consider this proclivity value mechanism only as an initial value system. In the future, we imagine implementing more elaborated drives, for example drives that may vary according to the agent's internal status (e.g., simulating a form of hunger varying over time).

2.2 Knowledge representation

An agent's stream of interaction with a given environment depends both on its decisions and on the unfolding of the environment. That is, the agent may well decide to try to enact an interaction with a high value, but this attempt may result in an actual interaction with a low value, due to unexpected environmental conditions (e.g., the agent may try to enact an interaction consisting of moving forward, but this attempt may result in bumping into a wall). This effect will be further explained in Section 4 and Figure 2. Because of this, the agent needs to learn to predict the interactions that result from its decisions. This raises the question of how to encode the knowledge that the agent learns.

We recommend representing the agent's knowledge as sequences of interactions. This view conforms with Gibson's [11] notion of *affordance*. Gibson suggests that the world is not known "objectively" but is rather known in terms of possibilities of interaction, called affordances. Encoding the agent's knowledge as sequences of interactions also follows from the fact that our agent has no other source of information about the environment anyway.

We devised an algorithm in compliance with these principles, called the *Intrinsically Motivated Schema mechanism* (IMOS) [10]. A significant difference from most existing decision process algorithms is that IMOS does not require Markov's hypothesis that the duration of the temporal dependency be known a priori. It can learn arbitrarily long episodes of interest by autonomously finding their beginning and end points. As opposed to partially observable Markov decision process (POMDP), IMOS does not require that the set of possible hidden states be defined a priori. IMOS recursively organizes episodes in a hierarchy of sub-episodes, a higher-level episode being a sequence of lower-level episodes. The proclivity value of a higher-level episode is set equal to the sum of the proclivity values of its sub-episodes, all the way down to predefined primitive proclivity values. The ability to perform such learning stems from the fact that, in the environmentally agnostic approach, the criteria for selecting interesting episodes are incorporated within the learning mechanism in the form of proclivity values. The agent can test hypothetical episodes and progressively select the most satisfying episodes with regard to their value.

Figure 1 illustrates this principle by representing learned hierarchical episodes of interaction in the agent's memory ((A(DA))(BC)). The activity at hand reactivates previously learned episodes that match the current situation (A(DA)) which in turn, triggers the subsequent interactions that are the most likely to result in satisfying interactions (as far as the agent can tell at this point in its development). This matching is possible because of the consistency between the representation of the situation and the representation of agent's procedural experience (a form of *homoiconicity*).

Again, we consider this mode of knowledge representation only as a starting point from which more elaborated representational structures can be derived. In particular, this mode of representation can be coupled with additional structures proposed by other researchers in Piagetian mechanisms, such as synthetic elements (e.g., [20]) or bare schemas (e.g., [12]). When implementing such structures, however, the assumptions these structures make about the environment should be explicitly stated. Notably, the purely sequential representations that we suggest here have the advantage of remaining compliant with the principle of environmental agnosticism because the agent has no *a priori* knowledge of how to represent the world "as such". The agent only knows and learns interactions.

3. EXPERIMENTAL PARADIGM

As introduced in Section 1, environment-agnostic agents are not designed to improve their performance over time with regard to a predefined problem set, task, or goal. Performance, as traditionally defined, is, therefore, not a property that appropriately accounts for the expected qualities of such agents. Instead, environment-agnostic agents are developed for their qualitative behavioral properties or for their theoretical implications on the study of developmental cognition. To study these agents, researchers need to agree on such expected properties.

This section proposes an initial list of expected properties based on our experience implementing environment-agnostic agents and on existing literature. In particular, Oudeyer, Kaplan, and Hafner [19] noted similar needs to characterize the properties of intrinsically motivated robots. These authors distinguished between three categories of criteria: (a) evolution of internal variables that account for the robot's learning (e.g., accuracy of anticipation or level of detail of learned categories); (b) evolution of external variables that characterize the robot's behavior (e.g., efficiency in the interaction with the environment); (c) evidence of reaching certain well-known developmental stages with regard to psychological or ethological theories.

3.1 Internal evaluation criteria

Category (a), the evolution of internal variables, has the advantage of objectivity because these criteria are based on variables implemented in the system. The drawback, however, is that each system has its specific variables, which complicates the comparison across systems. With these criteria, the authors need to clearly explain the significance of the variables.

A typical internal criterion is the growth of the variable that represents the system's *satisfaction*, as measured by its value system. We can formulate this criterion as:

a.1 Principle of objective hedonism.

For example, with the value system introduced in Section 2.1, the agent's objective hedonism is demonstrated by the agent's increasing ability to perform interactions with high values, and avoid interactions with negative values.

Notably, the principle of objective hedonism does not require the agent to reach the optimum value but only good-enough values (notion of bounded rationality [24]). Because good-enough values cannot be precisely defined, this principle should be complemented with qualitative principles that reflect the agent's decisional mechanisms more precisely. In particular, the agent should not simply react towards the highest immediate value. This can be expressed in the form of the corollary principle:

a.2 Principle of hedonistic temperance.

The agent should learn to enact negative interactions when such interactions can lead to even more positive interactions. Conversely, the agent should learn to refrain from enacting positive interactions when such interactions would lead to more negative interactions.

3.2 Behavioral criteria

Category (b), behavioral criteria, has the advantage of supporting comparisons across systems, because these criteria are based on the external observation of the system's behavior. The expected behavior can, however, vary across studies, raising the need for defining general principles. Assessing the agent's development with regard to general principles is the point of the third category listed by Oudeyer and coauthors (c). Yet, principles proposed by theories in psychology and ethology appear too vague, and out of reach for current artificial systems [12]. In the current state of the art, we need precise behavioral principles that account for the very beginning of the developmental process.

Surveys in developmental robotics (e.g. [2, 16, 28]) suggest three widely acknowledged principles:

b.1 Principle of situational categorization.

The agent should exhibit the capacity to categorize aspects of its situation and to adjust its behavior according to such categories.

b.2 Principle of situational disambiguation.

The agent should distinguish between different situations that generate the same sensory stimuli (perceptual aliasing, [3]).

b.3 Principle of graceful readaptation.

The agent should readapt gracefully to novel situations rather than experiencing catastrophic forgetting [7].

Moreover, the interaction-centered approach inspires two additional principles:

b.4 Principle of active perception.

The agent should learn to enact interactions not only because of their direct proclivity but also to update its representation of the current situation so as to take better decisions.

b.5 Principle of individuation.

Different executions of the same system should possibly lead to individualized instances exhibiting different habits. This principle accounts for the intuitive distinction between drives and goals according to which drives should leave room to individual *choices*. Individuation can occur through an "en habitus deposition" (De Loor [4] citing Husserl).

In summary, the criteria to assess environment-agnostic agents generally involve temporal analysis—either quantitative (category a) or qualitative (category b). This indicates a need for implementation platforms that generate activity traces and support activity trace analysis. The next section illustrates these principles by presenting example experiments.

4. EXAMPLE IMPLEMENTATIONS

When implementing an environment-agnostic agent in a specific environment, the designer chooses the meaning that he or she assigns to the primitive interactions. For example, he or she may design a two-dimensional grid where interactions may consist of moving, turning, bumping into obstacles, touching objects, etc. An agnostic agent, however, may even ignore that its world has two dimensions. The designer implements the execution of the interactions in an interface layer, as depicted in Figure 2.

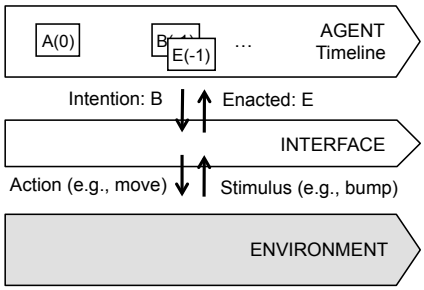


Figure 2. The interface agnostic agent/environment.

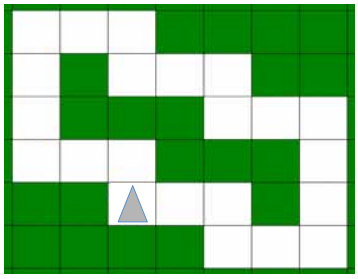
The interface between the agent and the environment generates the actions in the environment (e.g., try to move) from the intended interactions (e.g., B: move), and either confirms the intended interaction (B) or generates a contradictory interaction (e.g., E: bump). The meaning of the interaction is not implemented in the agent's decisional algorithm.

The designer also chooses the primitive interactions' proclivity values to generate interesting behaviors. For instance, if a negative value is associated with the interaction *turn*, and a positive value with the interaction *move forward*, then the agent tends to move forward, and turns only to ensure even more moving forward or to avoid subsequent even more negative interactions (such as bumping a wall). Conversely, with positive values associated with turning, the agent would learn to spin in place.

Interesting behaviors come from that an agent's possibilities of interaction and their proclivity values are adapted to a specific environment. In the case of natural organism, we assume that such adaptation results from phylogenetic evolution. Notably, this motivational mechanism is neither purely extrinsic (as would be a reward associated with an object in the environment), neither purely intrinsic (as *curiosity* in [2, 19]). Instead, it is *interaction-centered* and results from the pairing of an agent with an environment.

4.1 Simple sequential environment

We first demonstrated the intrinsically motivated schema mechanism in the experiment shown in Figure 3 [10]. In this case,



Primitive interactions		Val.
Move forward	→	10
Bump wall	→	-10
Turn 90° right toward empty square	↘	0
Turn 90° right toward wall	↘	-5
Turn 90° left toward empty square	↗	0
Turn 90° left toward wall	↗	-5
Touch wall ahead	–	0
Touch empty square ahead	–	-1
Touch wall on the right	\	0
Touch empty square on the right	\	-1
Touch wall on the left	/	0
Touch empty square on the left	/	-1

Trace:
1→ 2\ 3→! 4\ 5\ 6/! 7\ 8/ 9/! 10↘! 11↗! 12→! 13/! 14/ 15→! 16↗ 17↘! 18↘! 19→! 20/! 21/ 22↘ 23→ 24↘! 25\! 26/ 27→! 28↗! 29- 30→! 31→! 32- 33↘! 34- 35↗! 36\ 37↗ 38-! 39→ 40/ 41↘! 42/! 43/ 44- 45/! 46/ 47/ 48/ 49↗ 50\ 51-! 52→ 53- 54/! 55\! 56↗! 57↗ 58→ 59↗! 60/! 61/ 62(//) 63/ 64(//) 65↗ 66→ 67→! 68↘ 69→ 70- 71- 72(//) 73↗ 74→ 75/ 76\! 77-! 78→ 79\ 80- 81\ 82- 83(//) 84↗ 85→ 86/ 87↗! 88/ 89↗ 90→ 91\! 92- 93/! 94- 95\! 96↘ 97→ 98\ 99-! 100→ 101- 102- 103/ 104\! 105↘ 106→ 107\ 108- 109(//) 110↗ 111→ 112-! 113→ 114- 115/! 116/ 117- 118\! 119↘ 120→ …

Figure 3. Example environment-agnostic agent in a sequential environment (adapted from [10]).

Left: The agent (triangle) in the environment. Filled cells are walls which the agent can bump into. Center: List of the 12 primitive interactions with their proclivity values. Right: Activity trace of an example run. Steps 116 through 120: the agent has learned how to recognize and deal with a corner: touch the wall on the left – touch the wall in front – touch the empty square on the right – turn right – move forward.

The agent had 6 possible choices: try to move forward, turn left, turn right, touch in front, touch left, and touch right. The environment generated a single bit in return, which resulted in the 12 possible primitive interactions listed in Figure 3 (center). Again, the agent had no other way of apprehending its environment than interaction (no traditional "perceptual system"). This experiment can be seen online[2].

An analysis of this experiment based on the criteria listed in Section 3 leads to the following findings. This agent met the principle of objective hedonism (a.1) by learning to enact interactions of higher value, in particular by learning to avoid bumping into walls. Yet, the agent demonstrated temperance (a.2) because it learned to enact turn and touch interactions to ensure safer moves. The agent also demonstrated its capacity to identify and discriminate between situations (b.1, b.2) by representing the situation in the form of sequences rather than with the current feedback received from the environment (a single bit at each single point in time). The agent showed active perception (b.4) by adopting the habit of touching ahead before moving forward, and not moving forward if it touched a wall. This result is original because nothing initially differentiated perceptive interactions from motion interaction from the agent's viewpoint, except their cost (value). In essence, the agent learned to use cheap interactions to gain a better representation of the situation to ensure safer high-value interactions, which grounded the meaning of the constructed perception in the agent's activity.

This agent, however, had difficulties when the environment was not shaped as a linear route but was rather an open space, because the agent's sequential mechanism had trouble capturing spatial regularities. To address this difficulty, we implemented the experiment reported next.

4.2 Simple open space environment

In the experiment reported in Figure 4, we implemented interactions that were sensitive to remote properties of the environment [8]. This implementation was inspired by the visual system of an archaic arthropod, the limulus (horseshoe crab). In particular, two notable properties of the limulus were reproduced: (a) sensitivity to movement: the limulus's eye responds to movement, and the limulus has to move to "see" immobile things; (b) behavioral proclivity toward targets: male limulus move toward females, based on vision.

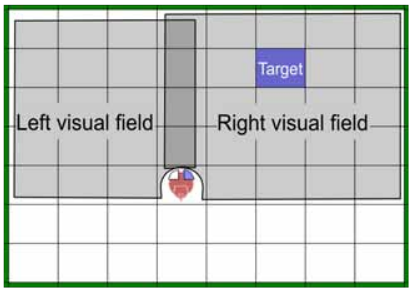
To replicate these behaviors, we simply implemented an interface that sent dynamic visual features to the agent. In this case, the primitive interactions were generated from the agent's action in the environment (move and turn), associated with the dynamic features resulting from the changes in the agent's visual field (target appear, closer, reached, and disappear). The behavioral proclivity toward targets was implemented by giving positive values to interactions in which the target appeared or enlarged in the visual field, and negative values when the target disappeared from the visual field (Figure 4, center). Note that the agent had no way to accurately predict when the target would disappear from its visual field as it moved forward.

This experiment demonstrated hedonist temperance (a.2) and active vision (b.4). In some instances, the agent learned to move forward until the target disappeared from its visual field as the agent passed it (active vision, b.4), then made a U-turn (hedonist temperance, a.2) and turned to realign itself with the target (in bold in Figure 4, right). In other instances, the agent learned to move toward the target in stair steps until it aligned itself with the target (trail shown in Figure 5). Once learned, an agent instance kept the same strategy when new targets were introduced in the board. This demonstrates the principle of individuation (b.5). Videos of these behaviors are available online[3]. These behaviors can also be reproduced in the simulation platform described in Section 5.

In this experiment, the agent exhibited adaptation to a spatial environment by simply capturing sequential regularities, but without capturing spatial regularities. For example, the agent did not learn the persistence of objects in space: it stopped pursuing a target in configurations where the target became hidden behind walls. Additionally, this agent was unable to adapt its behavior to different objects in the environment, for example, seeking food when hungry and water when thirsty. Such issues of spatial regularity learning and object persistence learning constitute a topic of research that we are currently exploring [9].

[2] http://e-ernest.blogspot.com/2010/12/java-ernest-72-in-vacuum.html

[3] http://e-ernest.blogspot.com/2011/01/tengential-strategy.html



Primitive interactions		Values
Move forward	(>)	0
Bump wall	[>]	-8
Turn 90° right to empty square	(v)	0
Turn 90° right to adjacent wall	[v]	-5
Turn 90° left to empty square	(^)	0
Turn 90° left to adjacent wall	[^]	-5
Additional value for each eye		
Appear	*	15
Closer	+	10
Reached	x	15
Disappear	o	-15

Trace:
1 2(> |+) 3(> |+) 4(> |+) 5(> |+) 6(> |+) 7(> |o) 8(v |*) 9(v*|o) 10(>+|) 11(^ |*) 12(^o|) 13(^ |o) 14(^*|) 15(>o|) 16(>) 17(>) 18(^*|) 19(vo|) 20(v) 21(^) 22(>) 23(^*|) 24(^o|*) 25(^ |o) 26(^) 27(v) 28(v |*) 29(^ |o) 30(^) 31(>) 32(^*|) 33(^o|*) 34(v*|o) 35(>+|) 36(^o|*) 37(v*|o) 38(>+|) 39(^ |*) 40(v |o) 41(>o|) 42(>) 43(^*|) 44(^o|*) 45(> |+) 46(> |+) 47(> |o) 48(v |*) 49(v*|o) 50(>+|) 51(^ |*) 52(>+|+) 53(v |o) 54(vo|) 55(v |*) 56(v*|) 57(v |o) 58(^ |*) 59(>+|+) 60(>+|+) 61(>+|+) 62(>x|x) 63(>o|o) 64(v) 65(v) 66(v) 67[v] 68(^) 69[v] 70(v) 71(v) 72(v) 73[v] 74[>] 75(^*|) 76(^o|*) **77(> |+) 78(> |+) 79(> |+) 80(> |+) 81(> |+) 82(> |o) 83(v |*) 84(v*|o) 85(>+|) 86(^ |*) 87(>+|+) 88(>x|x)** 89(^o|o) ...

Figure 4. Example environment-agnostic agent in an open space environment (adapted from [8]).

Left: The agent in the environment. The agent's visual system is made of two pixels that can only detect blue cells (targets). Each pixel covers a 90° span. Center: primitive interactions: move, or turn 90°, plus dynamic features possibly returned by each eye: appear, closer, reached, disappear. Right: Activity trace. An interaction consists of associating the agent's action with the signal sent by the eyes, separated by the symbol "|". Step 62: the agent finds and "eats" the first target. Step 75: a second target is inserted in the environment. Steps 77 to 88 (bold) demonstrate the learned behavior: Steps 77-81: the agent goes on a straight line with the target enlarging in its right eye's field. Step 82: the target disappears from the agent's right eye's field as the agent passes it. Steps 83-86: the agent makes a U-turn, returns back one step, and turns left towards the target. Step 87: the agent is aligned with the target and moves forward. Step 88: the agent reaches the target. Once learned, this "strategy" is repeated to reach other targets that the experimenter randomly introduces in the environment.

5. DESIGN/SIMULATION PLATFORM

Evaluating agnostic agents using the principles and criteria described in Sections 2 and 3 demands a flexible agent simulation platform to easily set up and run experiments. Such a platform should provide the means to parameterize experiments as well as offer mechanisms to detect patterns of agent behaviors in order to assess the agent's performance. We decided to use NetLogo [29] because we find it one of the simplest, most powerful, and most widely used agent simulation platforms available.

In a typical NetLogo simulation, the world consists of patches, i.e. components of a grid, and turtles that are agents that "live" and interact in that world. Modeling complex agents and environments is greatly facilitated by the fact that each entity participating in the simulation can carry its own state, stored in a set of system and/or

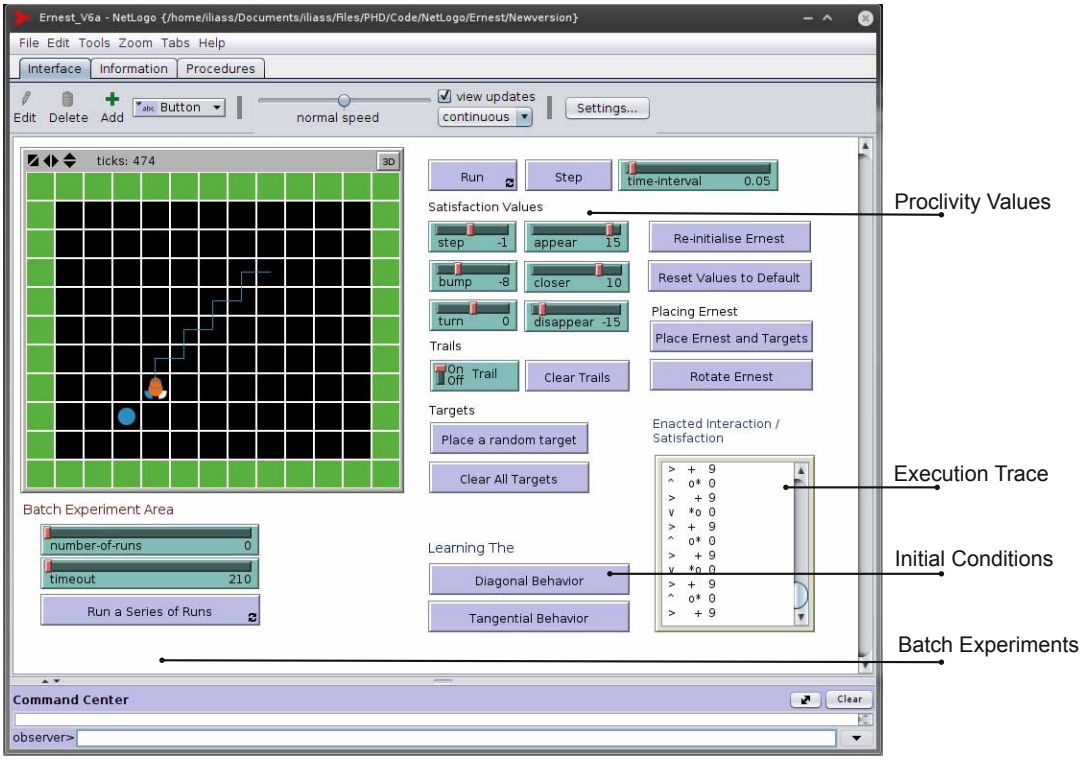


Figure 5: The NetLogo implementation of the IMOS agnostic agent. The platform allows the user to easily draw qualitative results. In the image, the diagonal strategy is shown by the trail left behind by the agent. The user can also set up a series of experimental runs (batch experiments) to observe the agent's long-term behavior.

user defined variables, allowing great flexibility in the simulations that can be implemented. Moreover, a simulation domain specific programming language permits implementation of any "sensors" or "actuators" that the experiment requires. Thus, a large range of worlds and a variety of agents can be simulated in order to evaluate design choices and test ideas in agent systems.

The intrinsically motivated schema mechanism [10] was integrated in NetLogo through an appropriate Java module, that we called the IMOS extension (Interaction MOtivation System). Thus, the environment and the interface (as defined in Section 4 and illustrated in Figure 2) are implemented in standard NetLogo, whereas the agents' decision/learning mechanism is handled by the IMOS extension. This architecture facilitates implementation of an unlimited number of different experimental settings by simply encoding the agent/environment related models, as well as effortlessly changing the agent parameters (e.g. proclivity values) in order to see how these parameters affect the agent's behavior. Finally, the NetLogo primitives offer an easy visualization of the agent's motion in the world (trails) that leads to a clear detection of patterns of interaction in the agent's behavior (Figure 5). Although, all the above could be implemented in any programming language, the simulation Domain Specific Language (DSL) that NetLogo offers makes the task of encoding simulation environments including controls, monitors etc., simple.

The implementation of the open space environment described in Section 4.2 is depicted in figure 5. Through the interface, the user can change the full set of experimental parameters in order to investigate their impact on the learned behavior. Such experimentation can lead to rather interesting observations. For example, using the platform, we were able to investigate how proclivity values affect the agent's interaction with the environment and also to investigate the impact of initial conditions, i.e. the initial locations of the agent and the target, on the different "strategies" learned by the agent in order to approach the target.

In the experiment in Section 4.2, the agent had a rather limited way of navigating through the environment, simply by moving between grid positions and turning by 90 degrees each time. In order to assess the algorithm's robustness and performance in a higher resolution environment, we simply changed some of the sensors and effectors (turning angle, cone of vision and movement capabilities) of the agent in the NetLogo part of the simulation, with proclivity values and the learning algorithm remaining the same as in the original experiment. As seen in figure 6, the agent managed to learn a similar strategy to approach the target, supporting further the argument regarding the robustness of the learning algorithm and the approach.

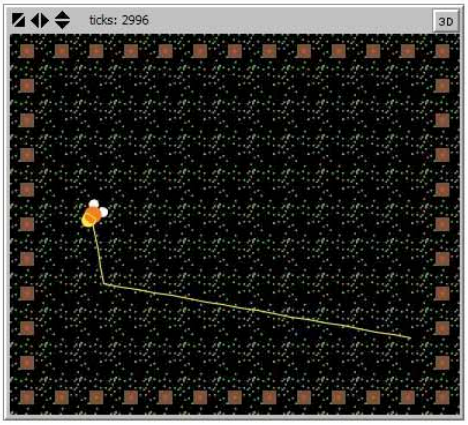


Figure 6: Learning to reach the target in a continuous environment

Thus, in more than one respect, NetLogo proved to be an excellent platform for evaluation and testing of the agnostic agent approach presented in this paper. The whole experimental platform, together with the Java library module, is available online[4].

6. CONCLUSION

We propose an approach to designing autonomous agents with minimal preconception of their environment. We characterize such agents as environmentally agnostic. This approach focuses on the notion of *sensorimotor patterns of interaction* between the agent and the environment rather than on the usual notions of *perception* and *action*.

The sensorimotor approach allows the implementation of an *interaction-centered* motivational mechanism. With this mechanism, the agent has inborn proclivities to enact sensorimotor patterns with high values and to avoid sensorimotor patterns with negative values. To do so, the agent needs to discover, learn, and exploit regularities of interaction afforded by the environment, which results in the autonomous construction of operative knowledge.

The interaction-centered motivational mechanism contrasts with traditional problem-solving or reinforcement-learning approaches in that it implements *drives* rather than *goals*. We argue that the fulfillment of drives should be assessed through activity analysis rather than performance measurement. We propose a list of developmental principles that should be observed in the agent's activity, in particular: objective hedonism, hedonistic temperance, active perception, and individuation.

Experiments show agents that meet the developmental principles in rudimentary settings. We provide the algorithm as a NetLogo extension to demonstrate that the agent's decisional process is independent from the environment that the designer chooses to implement. These experiments constitute an initial investigation of the principles of environment-agnosticism but the resulting behaviors are still rudimentary and many issues remain. In particular, we are now studying how environment-agnostic agents can learn spatial regularities and knowledge of persistent objects in the environment.

7. ACKNOWLEDGMENTS

This work was supported by the *Agence Nationale de la Recherche* (ANR) contract ANR-10-PDOC-007-01. We gratefully thank Jonathan Morgan and James Marshall for their review of this article.

8. REFERENCES

[1] Aström, K. 1965. "Optimal control of Markov processes with incomplete state information". Journal of Mathematical Analysis and Applications (10). 174-205.

[4] http://users.uom.gr/~iliass/projects/NetLogo/Ernest/index.html

[2] Blank, D.S., Kumar, D., Meeden, L. and Marshall, J. 2005. "Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture". Cybernetics and Systems, 32 (2). 125-150.
[3] Crook, P. and Hayes, G. 2003. Learning in a state of confusion: Perceptual aliasing in grid world navigation. In Proceedings of Towards Intelligent Mobile Robots (Bristol), UWE.
[4] De Loor, P., Manac'h, K. and Tisseau, J. 2010. "Enaction-based artificial intelligence: Toward co-evolution with humans in the loop". Minds and Machine, 19. 319-343.
[5] Dennett, D. 1991. Consciousness explained. Penguin.
[6] Dennett, D. 2009. "Darwin's ''strange inversion of reasoning''". PNAS, 106. 10061-10065.
[7] French, R. 1999. "Catastrophic forgetting in connectionist networks: Causes, consequences and solutions". Trends in Cognitive Sciences, 3 (4). 138-135.
[8] Georgeon, O.L., Cohen, M. and Cordier, A. 2011. A Model and simulation of Early-Stage Vision as a Developmental Sensorimotor Process. In Proceedings of Artificial Intelligence Applications and Innovations (Corfu, Greece, 2011), 11-16.
[9] Georgeon, O.L., Marshall, J. and Ronot, P.-Y. 2011. Early-Stage Vision of Composite Scenes for Spatial Learning and Navigation. In Proceedings of First Joint IEEE Conference on Development and Learning and on Epigenetic Robotics (Frankfurt, Germany, 2011), 224-229.
[10] Georgeon, O.L. and Ritter, F.E. 2012. "An Intrinsically-Motivated Schema Mechanism to Model and Simulate Emergent Cognition". Cognitive Systems Research, 15-16. 73-92.
[11] Gibson, J.J. 1979. The ecological approach to visual perception. Houghton-Mifflin, Boston.
[12] Guerin, F. and McKenzie, D. 2008. A Piagetian model of early sensorimotor development. In Proceedings of Eighth International Conference on Epigenetic Robotics (Brighton, UK).
[13] Harnad, S. 1990. "The symbol grounding problem". Physica D, 42. 335-346.
[14] Hurley, S. 1998. Consciousness in action. Harvard University Press, Cambridge, MA.
[15] Laird, J.E. and Congdon, C.B. The Soar User's Manual Version 9.1, University of Michigan, 2009.
[16] Lungarella, M., Metta, G., Pfeifer, R. and Sandini, G. 2003. "Developmental robotics: a survey". Connection Science, 15 (4). 151-190.
[17] Mille, A. 2006. "From case-based reasoning to traces-based reasoning". Annual Reviews in Control, 30 (2). 223-232.
[18] O'Regan, J.K. and Noë, A. 2001. "A sensorimotor account of vision and visual consciousness". Behavioral and Brain Sciences, 24 (5). 939–1031.
[19] Oudeyer, P.-Y., Kaplan, F. and Hafner, V. 2007. "Intrinsic motivation systems for autonomous mental development". IEEE Transactions on Evolutionary Computation, 11 (2). 265-286.
[20] Perotto, F., Buisson, J. and Alvares, L. 2007. Constructivist anticipatory learning mechanism (CALM): Dealing with partially deterministic and partially observable environments. In Proceedings of Seventh International Conference on Epigenetic Robotics (Rutgers, NJ, 2007).
[21] Pfeifer, R. and Scheier, C. 1994. From perception to action: The right direction? In From Perception to Action, Gaussier, P. and Nicoud, J.-D. eds. IEEE Computer Society Press, 1-11.
[22] Piaget, J. 1970. L'épistémologie génétique. PUF, Paris.
[23] Ram, A. and Santamaria, J.C. 1997. "Continuous case-based reasoning". Artificial Intelligence, 90 (1-2). 25-77.
[24] Simon, H. 1955. "A behavioral model of rational choice". Quaterly Journal of Economics, 69. 99-118.
[25] Singh, S., Barto, A.G. and Chentanez, N. 2005. Intrinsically motivated reinforcement learning. In Advances in Neural Information Processing Systems, Saul, L.K., Weiss, Y. and Bottou, L. eds. MIT Press, Cambridge, MA, 1281-1288.
[26] Sporns, O. 2003. Embodied cognition. In MIT Handbook of Brain Theory and Neural Networks, Arbib, M. ed. MIT Press, Cambridge, MA.
[27] Sutton, R.S., Modayil, J., Delp, M., Degris, T., Pilarski, P.M., White, A. and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In Proceedings of Tenth International Conference on Autonomous Agents and Multiagent Systems (Taipei, Taiwan).
[28] Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M. and Thelen, E. 2001. "Artificial intelligence - Autonomous mental development by robots and animals". Science, 291 (5504). 599-600.
[29] Wilensky, U. 1999. NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. http://ccl.northwestern.edu/netlogo/.

Feature Selection Based on Reinforcement Learning for Object Recognition

Monica Piñol
Computer Science Dept.
Computer Vision Center
Universitat Autònoma de Barcelona
08193-Bellaterra
Barcelona, Spain
mpinyol@cvc.uab.es

Angel D. Sappa
Computer Vision Center
Campus UAB
08193-Bellaterra
Barcelona, Spain
asappa@cvc.uab.es

Angeles López
Computer Science and Engineering Dept.
Universitat Jaume I
12071-Castellón de la Plana
Castellón, Spain
lopeza@icc.uji.es

Ricardo Toledo
Computer Science Dept.
Computer Vision Center
Universitat Autònoma de Barcelona
08193-Bellaterra
Barcelona, Spain
ricard@cvc.uab.es

ABSTRACT

This paper presents a novel method that allows learning the best feature that describes a given image. It is intended to be used in object recognition. The proposed approach is based on the use of a Reinforcement Learning procedure that selects the best descriptor for every image from a given set. In order to do this, we introduce a new architecture joining a Reinforcement Learning technique with a Visual Object Recognition framework. Furthermore, for the Reinforcement Learning, a new convergence and a new strategy for the exploration-exploitation trade-off is proposed. Comparisons show that the performance of the proposed method improves by about 6.67% with respect to a scheme based on a single feature descriptor. Improvements in the convergence speed have been also obtained using the proposed exploration-exploitation trade-off.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents*; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis —*Object recognition*

General Terms

Algorithms, Performance, Experimentation

Keywords

Reinforcement Learning, Object Recognition, Q-Learning, Visual Feature Descriptors

1. BACKGROUND

Usually, in Visual Object Recognition (VOR), the scenes are represented in feature spaces where the classification and/or recognition tasks can be done more efficiently. There are several representations with different levels of performance. Often, the feature space is built around "interest points" of the scene. The interest points are obtained with detectors, then, for each "interest point" a descriptor is computed.

Currently, researchers are beginning to work with reinforcement learning in computer vision. For instance, several approaches have been proposed in the image segmentation field [19, 21, 22]. In all these cases, the reinforcement learning has been used to tackle the threshold tuning reaching a similar performance than the state of the art approaches.

There are also some works in the face recognition problem. For instance, in [8] the authors proposes to learn the set of dominant features for each image to recognize the faces using a reinforcement learning based technique.

In the VOR domain, [13] presents a method using bottom-up and top-down strategies joining Reinforcement Learning and Ordinal Conditional Functions. A similar approach has been proposed in [6], which joins Reinforcement Learning with First Order Logic. In [10] and [11], a new method for extracting image features is presented. It is based on "interest points" detection, and uses reinforcement learning and aliasing to distinguish the classes. Finally, in [2] the reinforcement learning method is used to select the best classification in a Bag of Features approach. On the contrary to previous works, in the current work we propose the use of a reinforcement learning based method to learn the best descriptor for each image.

One of the most recent and widely used approach for VOR is Bag of Features (BoF) [4, 5]. Figure 2($left$) shows an illustration of the BoF architecture. In general, a BoF approach consists of four steps detailed next. In the first step, feature extraction is carried out by image feature descriptors. In [15] the performance of descriptors is analysed. The second step is the creation of a dictionary using visual words from a training set; and the third step is the image representation. Both steps are solved with a Vocabulary Tree (VT)

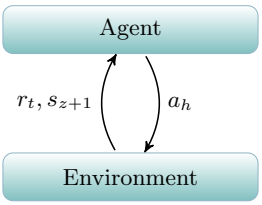


Figure 1: Scheme of interaction between the agent and the environment

[17]. The VT defines a hierarchical tree by k-means clustering. In the algorithm, k defines the branch factor (number of children of each tree-node) and the process is recursive until reaching a maximum depth. At any tree level, each cluster is defined by a dictionary of visual words. The fourth step is the classification.

The BoF approach does not specify the descriptors and the classification algorithms. One solution for determining the descriptors is concatenating all of them, but this solution introduces noise as well as increase the CPU time. Furthermore, since each image could have different characteristics of color, texture, edges, etc.; images from the same database could behave differently when we apply the same descriptor.

This paper proposes a new approach for enhancing the BoF. The key idea is the introduction of the Reinforcement Learning technique [20] to learn the best descriptor for each image. Then, these descriptors are used in a VT scheme, which is used by a Support Vector Machine (SVM) algorithm for the classification. The reminder of the paper is organized as follows. Section 2 summarizes the Reinforcement Learning technique. Then, Section 3 presents the proposed method. Experimental results are provided in Section 4. Finally, conclusions and future work are given in Section 5.

2. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a learning method based on trial and error, where the agent does not have a prior knowledge about which is the correct action to take. The underlying model that RL learns is a Markov Decision Process (MDP). A MDP is defined as a tuple $\langle S, A, \delta, \tau \rangle$, where: S is the set of states; A is the set of actions; δ is a transition function δ:Sx$A \rightarrow S$; and τ is a reward/punishment function τ:Sx$A \rightarrow \Re$.

The agent interacts with the environment and selects an action. Applying the action (a_h) at state(s_z), the environment gives a new state (s_{z+1}) and a reward/punishment (r_t) (see Fig. 1). In order to maximize the expected reward, the agent selects the best action a_h based on the $\tau(s_z, a_h)$ provided by τ:Sx$A \rightarrow \Re$.

Different methods have been proposed in the literature for solving the RL problem: dynamic programming, Monte Carlo methods, and temporal difference learning. In the current work a temporal difference learning based method has been selected since it does not require a model and it is fully incremental [23]. In concrete, our framework is based on the *Q-learning* algorithm [25]. In this case, the agent learns the action policy π:$S \longrightarrow A$, where π maps the current state s_z into an optimal action a_h to maximize the expected long term reward.

The Q-learning proposes a strategy to learn an optimal policy π^*, when the δ function (δ:Sx$A \rightarrow S$) and τ function (τ:Sx$A \rightarrow \Re$) are not known a priori. The optimal policy is $\pi^* = argmax_{a'}\mathbf{Q}(s, a')$, where $\mathbf{Q}$ is the evaluation function the agent is learning. Note that since δ and/or τ are nondeterministic, the environment should be represented in a nondeterministic way. Hence, we can write the $\mathbf{Q}$ in a nondeterministic environment as follows:

$$\mathbf{Q}_n(s_z, a_h) \longleftarrow (1-\alpha_n)\mathbf{Q}_{n-1}(s_z, a_h) + \alpha_n[r + \gamma \max_{a'} \mathbf{Q}_{n-1}(s_{z+1}, a')], \quad (1)$$

$$\alpha_n = \frac{1}{1 + \mathbf{visits}_n(s_z, a_h)}, \quad (2)$$

where $0 \leq \gamma < 1$ is a discount factor for future reinforcements. The Eq. (2) is the value α_n for a nondeterministic world and **visits** is the number of iterations visiting the **Q**-table at the tuple (s_z, a_h) [16].

Since, in this problem is not feasible to update the Q-table using the Eq. (1), because the current state (s_z) is only affected by the previous visits (first order MDP), the expression in Eq. (1) is modified by Eq. (3).

$$\mathbf{Q}_n(s_z, a_h) \longleftarrow (1-\alpha_n)\mathbf{Q}_{n-1}(s_z, a_h) + \alpha_n[r + \gamma \max_{a'} \mathbf{Q}_{n-1}(s_z, a')], \quad (3)$$

where $0 \leq \gamma < 1$ and α_n is defined as in Eq. (2).

3. PROPOSED METHOD

This paper proposes a new method where the agent learns the best descriptor for each image. Figure 2 shows an illustration of the proposed scheme. The most important elements of the proposed approach are detailed in this section. First, the elements defining the tuple are introduced. Then, the proposed convergence strategy is presented. Next, the exploration-exploitation trade-off to select the actions is introduced. Finally, the training stage is summarized. Note the proposed method does not follow a classical RL based scheme where an action (a_h) and a reward/punishment (r_z), for a given state, produce a new state. In our case, after applying a given action, the **Q**-table is updated but it does not result in a new state, hence a new image is considered. The different stages are detailed next.

3.1 Tuple definition

The tuple that defines our MDP is $\langle S, A, \delta, \tau \rangle$. The variables of the tuple are:

3.1.1 State definition, S

In our case, the state is a representation of the image. Many different image features can be used as a state. In our experiments, we convert the color image to grey scale image. Then, we extract mean, standard deviation and median values from that image (Fig. 3(a)). After that, the process is applied again by splitting the image into four equally sized squared blocks, and, for each sub-image, we extract again the mean, standard deviation and median values (as depicted in Fig. 3(b)). Additionally, the number of corners (Fig. 3(c)) and the number of blobs (Fig. 3(d)) using the whole grey scale image (Fig. 3(a)) are extracted. The corners are obtained using a Harris corner detector [9], while blobs are obtained converting grey scale image to black and

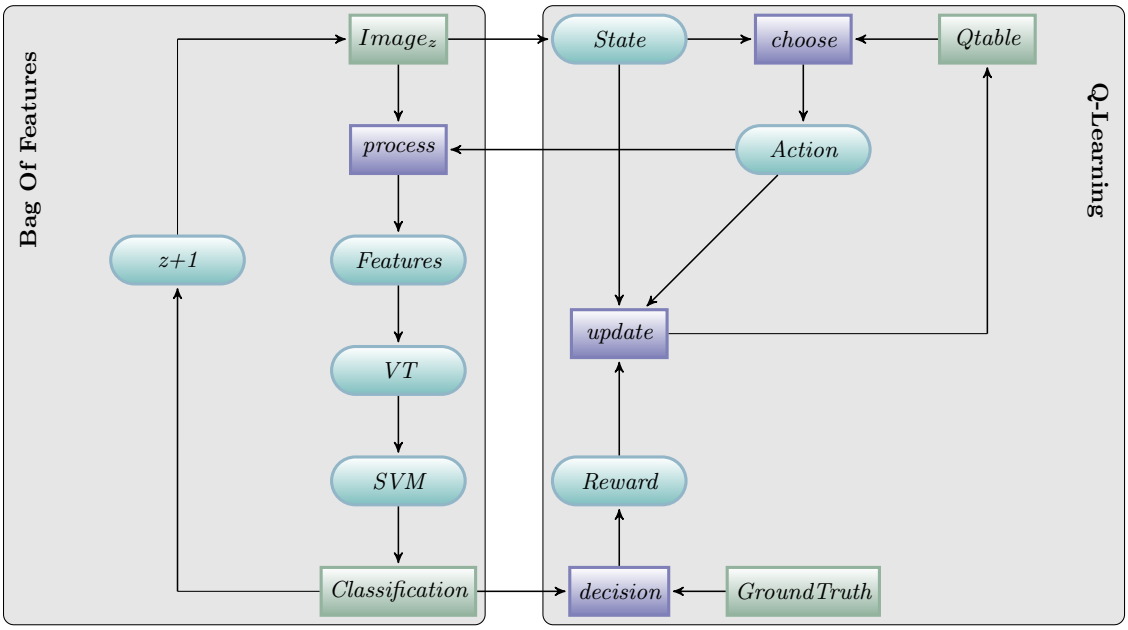


Figure 2: Illustration of the proposed scheme for image classification using Q-learning

white using OTSU threshold [18] and then, the labelling algorithm (bwlabel) with a connectivity of 8 neighbours [7] is applied. The result gives a vector of 17 dimensions with the following shape:

$\langle\ mean_{L_1},\ std_{L_1},\ median_{L_1},\ mean_{(1,1)L_2},\ std_{(1,1)L_2},\ \cdots,\ median_{(2,2)L_2}$, ncorners, nblobs $\rangle$.

This state definition is highly discriminative among images. Since the database could contain thousand of images, the size of the **Q**-table could be huge. To solve this problem, we propose a k-means clustering of the vectors and use the centroid of each cluster as a state ($S = \{s_z\}_{0<z<ncentroids}$), instead of using all the vectors' components. The size of the **Q**-table is determined by the number of clusters.

3.1.2 Action definition, A

In the current work, the actions $A = \{\ a_h\}_{0<h<u}$ are descriptors, where u is the size of the descriptor set. In other words, our agent learns which descriptor gives the best information for each image. Our architecture is flexible and does not depend on a particular set of descriptors. In the current work, the descriptors used as actions are based on gradients, blobs and patterns, although other combinations could be also used. The four descriptors selected to test the proposed architecture are as follow:

- SIFT (Scale-Invariant Feature Transform): finds scale invariant regions using the magnitude of the gradient [14].
- Spin: the descriptor makes a histogram of quantized pixels locations and intensity values. This descriptor finds textures [12].
- SURF (Speeded Up Robust Feature): is based on sums of 2D Haar wavelet responses and is efficiently computed using integral images [1].

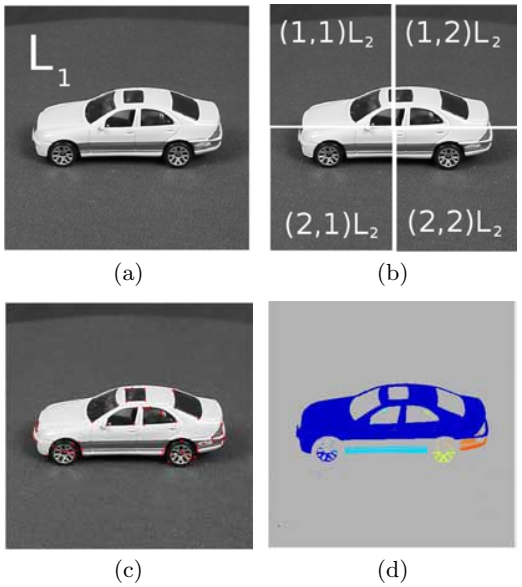


Figure 3: Illustration of an image of the database. (a) Original gray level image. (b) Image split up into four equally sized squared blocks. (c) The corners detected in the image. (d) The blobs detected in the image.

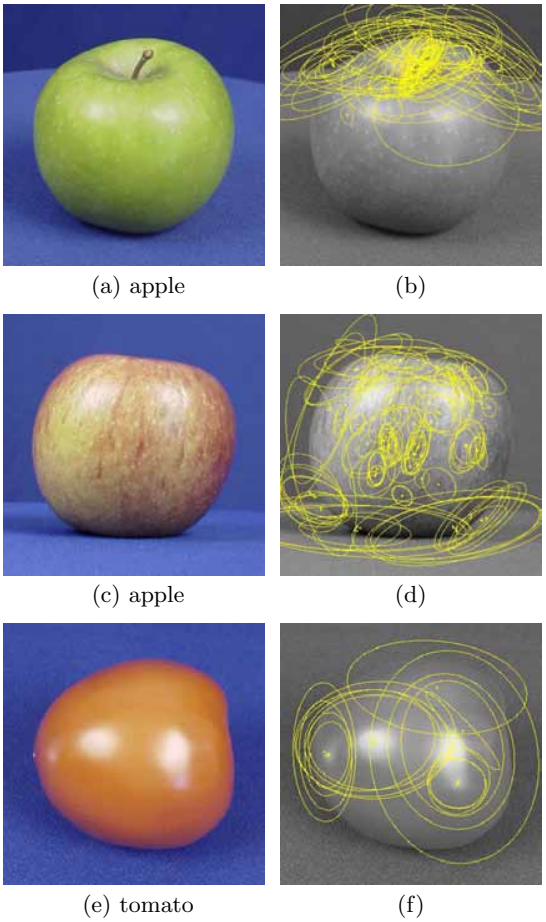
- PHOW (Pyramid Histogram Of visual Words): is a variant of dense SIFT descriptors, extracted at multiple scales [3].

3.1.3 δ function

The classical Q-learning formulation involves a δ function, which for a given state (s_z) and an action (a_h), it returns a

new state ($\delta : SxA \rightarrow S$).

In our case, given an image from the given data set (I_{s_z}), the δ function does not give a new state, instead, the output is a new representation of the image (I'_{s_z}). Also, two different images at the same state (centroid) and applying the same action, usually, leads to different representation (see the example in Fig. 4). For this reason, the δ function is nondeterministic.



(a) apple (b)

(c) apple (d)

(e) tomato (f)

Figure 4: ($left-column$) Illustration of three images from the database. Although these images belong to different classes, their centroid lies in the same position. Hence, they are classified into the same cluster. ($right-column$) The image patches obtained applying the same action: SIFT.

Once δ function is applied the learning process continues with the classification step. The BoF uses the new representation of the image(I'_{s_z}) to classify the object through the VT and SVM. Once the object has been classified, the iteration is finished and the process continues through a new image as a new iteration.

3.1.4 τ function

The agent decides the class of the image. The τ function returns a reward when the decision of the agent matches the ground truth, and, when the decision of the agent differs, the function returns a punishment. The τ function is also a nondeterministic function. During the process, we cannot ensure that two images at the same state (s_z) and doing the same action (a_h) lead to the same $\tau(s_z, a_h)$ [16].

For example, in Fig. 4 using the action SIFT over the three images gives the same reward because the VT correctly classifies all of them. But, if we repeat the same example changing the action SIFT by SURF, the first image (Fig. 4(a)) is within the class apple but the VT returns tomato. For the second image (Fig. 4(c)), the VT hits the class. Finally, the third image (Fig. 4(e)) is a tomato but the VT returns apple. Hence, using SURF, for the image (Fig. 4(c)) results in a reward but in the other two images (Fig. 4(a) and Fig. 4(e)) the process gives a punishment. In the current implementation τ is defined as ($+1000$) when the image is correctly classified and (-1000) when it is wrongly classified.

3.2 Convergence

The theorem "Convergence of Q-learning for nondeterministic Markov decision processes" [16] shows that a nondeterministic MDP converges when there is a bounded reward ($\forall(s,a), |r(s,a)| \leq c$ and $0 < \alpha_n \leq 1$). Eq. (4) is true in the ith iteration when $n \longrightarrow \infty$ with probability 1.

$$\sum_{i}^{\infty} \alpha_n(i,s,a) = \infty, \qquad \sum_{i}^{\infty} [\alpha_n(i,s,a)]^2 \leq \infty. \tag{4}$$

In our framework, the agent interacts with a nondeterministic environment, so, the convergence is very expensive in time. For example, in [16] we can see that the Tesauro's TD-GAMMON needs for training 1.5 million of backgammon games iterations and each of them contains tens of state-action transitions. As the convergence can last for weeks, we need some criteria to stop the training. Thus, in the current work it is proposed to stop the training when the following criterion is achieved:

$$\sum_{i=n-w}^{w} |\mathbf{Q}_{n+1}(s,a) - \mathbf{Q}_n(s,a)|_i < \theta, \tag{5}$$

the sliding window provides a restricted convergence. Where, w is the size of the sliding window and θ is the admitted error.

3.3 Exploration-exploitation trade-off

This section presents the action selection; in general, it is referred to as exploration-exploitation trade-off. If the agent uses only the exploration strategy it could fall into a local maximum. To avoid this problem, the RL learns some steps with an exploitation strategy.

An ε-greedy scheme is generally used as an exploitation strategy. In this paper, we propose a method to compute ε adaptively. We propose the measurement of the error as the parameter for switching the strategy. We define the error as $e = f(it)$, where f is defined as: $f(it) = |Q_{it} - Q_{it-1}|$; and, for each iteration we store this error. The ideal process reduces the error for each iteration, but sometimes the error increases (see Fig. 5).

We propose to calculate e_{it} and use this value as a switching indicator: $e_{it} - e_{it-1} > threshold$. However, to avoid switches when the error is a small spike, we propose to consider also the error in the neighborhood of current iteration.

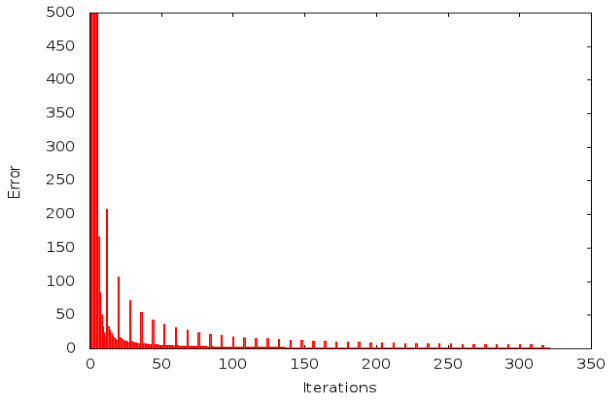


Figure 5: Behavior of e_{it}, till convergence is reached, for different iterations.

Hence, a sliding windows (w') is used as indicated in Eq. (6). In other words, if the following condition is fulfilled current strategy is switched to exploitation trade-off.

$$\sum_{i=0}^{w'} e_{it-i} > \sum_{i=1}^{w'+1} e_{it-i}, \tag{6}$$

where, like in Eq. (5), w' is the size of the sliding windows (in the current implementation $w' = 5$).

3.4 Training

In order to train and test our approach, we have considered an image database and a set of descriptors widely used in the computer vision literature [1, 3, 12, 14]. The image database is split up into three sets: vocabulary tree training set (VTTS), **Q**-table training set (QTTS), and testing set.

The first training is using the VTTS, a tree is built for each descriptor (T_{a_h}) using a BoF approach with VT and SVM [24].

The second training starts initializing the **Q**-table and the method is depicted in Fig. 2. Given an image from QTTS, the agent extracts the state (s_z) and using the **Q**-table and the exploration-exploitation strategy decides the action (a_h). The agent extracts the features using the descriptor (a_h) and classifies the image into one class. To do the classification, we use the pair (T_{a_h}, a_h). When the agent obtains a class, the agent compares this class with the ground truth and obtains the reward/punishment r_t through $\tau(s_z, a_h)$. Finally, the agent computes the convergence (Eq. (5)) and then, the agent updates the **Q**-table using Eq. (3).

4. RESULTS

The ETH database has been used to evaluate the proposed approach. Nine classes from that database have been selected: apple, car, cow, cowcup, cup, dog, horse, pear and tomato. Figure 6 shows some images of the database.

We have used 45 images per class, which were split into three sets: 15 images for training VT, 15 images for training the **Q**-table and finally, 15 images for testing. We have repeated the experiments fifteen times, using this three image sets. The process of testing starts when the **Q**-table achieves the convergence (Eq. (5), with $\theta = 0.4$).

Figure 6: Illustration from ETH database

Given a testing image, the process extracts the state and selects the action. To select the best action, the process searches the maximum value in the **Q**-table for this state. But sometimes, the **Q**-table does not have an unique maximum, to solve this problem, we introduce a vector with weights. The vector of weights is multiplied by the **Q**-table to obtain more distance between the actions.

In order to compare the results, we have measured the performance of each action (Table 1). Firstly, using as an action always the same descriptor. Secondly, the descriptor resulting from Q-learning is used. Note that other strategies could be considered for the comparisons, for instance considering as an action all the descriptors at once. However, this kind of strategy will introduce noise as well as will increase the CPU time.

The best result using a single descriptor as an action for the ETH database is PHOW with a performance of 74.81%. The proposed Q-learning scheme has been evaluated using

Table 1: Performance for each action and using the Q-table (Performance: percentage of success during the classification)

Action	Performance
Spin	60.00%
SIFT	61.48%
SURF	62.96%
PHOW	74.81%
Q-learning	81.48%

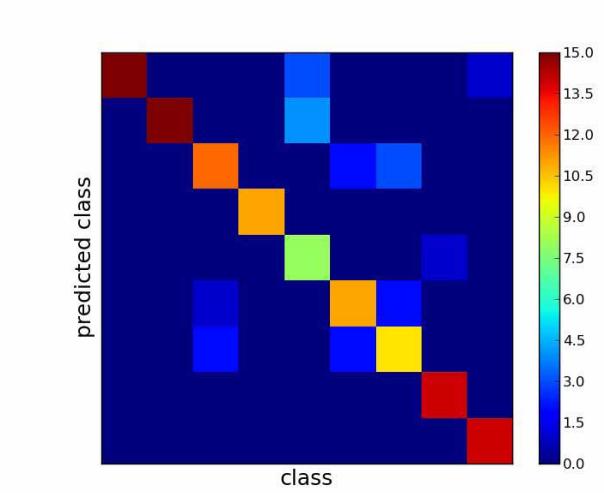


Figure 7: Confusion matrix with $\varepsilon = 0.5$ exploration-exploitation strategy; it achieves a performance of 81.48%

two different exploration-exploitation strategies. Firstly, an exploration-exploitation trade-off with $\varepsilon = 0.5$ has been used. It reaches a performance of 81.48%. The value ε has been set empirically; actually, other values have been tested (e.g., 0.2) reaching the similar results but increasing the number of iterations and consequently the CPU time. Figure 7 shows the confusion matrix.

The second experiment was done with the exploration-exploitation strategy presented in Section 3.3; in this case we also reach the same performance (81.48%). However, it should be noticed that the proposed approach converges in less iterations as depicted in Table 2, which shows the number of iterations needed for each exploration-exploitation strategy. The usual strategy needs more than 40.000 iterations to arrive at the same convergence state.

Table 2: Number of iterations for each strategy using $\theta = 0.4$

Strategy	Number of iterations
$\varepsilon = 0.5$	168.419
History of the error	127.710

5. CONCLUSIONS AND FUTURE WORK

In this paper a novel method to learn the best descriptor for each image in a database has been presented. A new architecture joining the Reinforcement Learning and Bag of Features is proposed. Additionally, a new exploration-exploitation strategy is introduced. The proposed approach has been validated using the ETH database. Its performance has been compared with respect to a single descriptor scheme. The best descriptor for the ETH database is PHOW with 74.81% of performance, while the proposed method reaches 81.48%. Therefore, the results are improved in almost 7% using the proposed method. Additionally, a strategy for exploration-exploitation is proposed that makes the convergence faster than using random switches.

Future work will be focused on the exploration of other state definition, and also, on the increase of the set of descriptors (e.g., H-mat, GIST, Centrist, etc. are some of the descriptors to be considered). The proposed method will be tested with databases containing different backgrounds. Finally, the possibility of selecting the actions without random values will be studied.

6. ACKNOWLEDGMENTS

This work was partially supported by the Spanish Government under Research Program Consolider Ingenio 2010: MIPRCV (CSD2007-00018) and Project TIN2011-25606. Monica Piñol was supported by Universitat Autònoma de Barcelona grant PIF 471-01-8/09.

7. REFERENCES

[1] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. *Proc. the European Conference on Computer Vision*, pages 404–417, 2006.

[2] R. Bianchi, A. Ramisa, and R. de Mántaras. Automatic Selection of Object Recognition Methods using Reinforcement Learning. *Advances in Machine Learning I*, pages 421–439, 2010.

[3] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. *Proc. International Conference on Computer Vision*, 2007.

[4] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. *In Workshop on Statistical Learning in Computer Vision, Proc. the European Conference on Computer Vision*, pages 1–22, 2004.

[5] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 524–531, 2005.

[6] K. Häming and G. Peters. Learning scan paths for object recognition with relational reinforcement learning. *Signal Processing, Pattern Recognition and Applications*, 2010.

[7] R. Haralick and L. Shapiro. *Computer and robot vision*, volume 1. Addison-Wesley, 1992.

[8] M. T. Harandi, M. N. Ahmadabadi, and B. N. Araabi. Face recognition using reinforcement learning. *Proc. IEEE International Conference on Image Processing*, IV:2709–2712, 2004.

[9] C. Harris and M. J. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, -:147 – 152, 1988.

[10] S. Jodogne. Reinforcement learning of perceptual classes using q learning updates. *Proc. of the 23rd IASTED International Multi-Conference on Artificial Intelligence and Applications*, pages 445–450, 2005.

[11] S. Jodogne and J. H. Piater. Interactive selection of visual features through reinforcement learning. *24th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 285–298, 2004.

[12] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using local affine regions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278, 2005.

[13] T. Leopold, G. Kern-Isberner, and G. Peters. Belief revision with reinforcement learning for interactive object recognition. *Proc. the European Conference on Artificial Intelligence*, pages 65–69, 2008.

[14] D. Lowe. Distinctive image features from scale invariant keypoints. *International Journal on Computer Vision* , 2:91–110, 2004.

[15] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[16] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[17] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2:2161 – 2168, 2006.

[18] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:62–69, 1979.

[19] J. Peng, , J. Peng, and B. Bhanu. Local reinforcement learning for object recognition. *Pattern Recognition* , 1:272–274, 1998.

[20] S. Russell, P. Norvig, J. Canny, J. Malik, and D. Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, NJ, 1995.

[21] F. Sahba, H. R. Tizhoosh, and M. Salama. Aplication of opposition-based reinforcement learning in image segmentation. *IEEE Computational Intelligence in Image and Signal Processing*, pages 246 – 251, 2007.

[22] M. Shokri and H. R.Tizhoosh. A reinforcement agent for threshold fusion. *Applied Soft Computing*, 8:174 – 181, 2008.

[23] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 28. Cambridge Univ Press, 1998.

[24] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms, 2008. http://www.vlfeat.org/.

[25] C. J. C. H. Watkins. *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge UK, 1989.

Help an Agent Out: Student/Teacher Learning in Sequential Decision Tasks

Lisa Torrey
St. Lawrence University
ltorrey@stlawu.edu

Matthew E. Taylor
Lafayette College
taylorm@lafayette.edu

ABSTRACT

Research on agents has led to the development of algorithms for learning from experience, accepting guidance from humans, and imitating experts. This paper explores a new direction for agents: the ability to teach other agents. In particular, we focus on situations where the teacher has limited expertise and instructs the student through action advice. The paper proposes and evaluates several teaching algorithms based on providing advice at a gradually decreasing rate. A crucial component of these algorithms is the ability of an agent to estimate its confidence in a state. We also contribute a student/teacher framework for implementing teaching strategies, which we hope will spur additional development in this relatively unexplored area.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Reinforcement Learning, Inter-agent teaching, Transfer Learning

1. INTRODUCTION

Agents are becoming increasingly common in industry, education, and domestic environments. Significant advances have been made in autonomous learning and learning with human guidance. However, less attention has been paid to the question of how agents could best teach each other. For instance, an existing robot in a factory should be able to instruct a newly arriving robot, even if it is from a different manufacturer, has a different knowledge representation, or is not optimal itself.

This paper investigates a variety of teaching methods in sequential decision tasks. In particular, we consider a reinforcement learning student that must learn from autonomous exploration of the environment under the guidance of another agent. In order to minimize inter-operability requirements, the teacher and student are presumed not to know each others' internal workings; teachers can only help students by suggesting actions. Furthermore, the teacher may have limited expertise in the student's task, so it should be careful not to over-advise the student. The primary question we address is: how should the teacher decide when to give advice?

The teaching context is related to the more well-studied problem of *transfer learning* [16, 17, 18], in which an agent uses knowledge from a source task to aid its learning in a target task. However, transfer algorithms often assume that agents can directly access the internal knowledge representation from the source task, which is too strong an assumption to make in teaching.

Another related area is *learning from experts* [4, 12], where agents may imitate experts or ask for their advice. This setting puts the student in charge of the process; our work gives more control to the teacher, because the teacher is the agent with more initial knowledge. Our setting also focuses on non-expert teachers whose knowledge may not be complete.

We introduce a family of teaching methods and conduct a set of experiments to evaluate and compare them. Experiments take place in both discrete and continuous tasks, and with varying levels of teacher expertise.

Empirical results from this study highlight a number of insights into inter-agent teaching. First, teachers can make better decisions about when to give advice if they are able to judge their own confidence in their knowledge. Second, teachers can also make better decisions if they are able to ask about the student's confidence and compare it to their own. Third, there are multiple factors that affect the relative performance of different teaching algorithms: the domain, the teacher's level of expertise, and even the student's exploration strategy.

The primary contributions of this paper are to suggest effective algorithms for teaching, to highlight interesting empirical observations, and to provide an open-source framework for evaluating teaching methods. Our hope is that this paper enables and inspires the agents community to develop further methods by which agents can teach other agents.

2. BACKGROUND AND RELATED WORK

This section provides a summary of important background information, and a survey of existing work in relevant areas.

2.1 Reinforcement Learning

This paper focuses on *reinforcement learning* (RL), a popular formulation for sequential decision tasks [7, 14]. RL tasks are typically framed as Markov decision processes (MDPs) and defined by the 4-tuple of state set, action set, transition function, and reward function: $\{S, A, T, R\}$. A learner chooses which action to take in a state via a policy, $\pi : S \mapsto A$, which is modified over time. A better policy gives the learner better performance, which is defined as the expected (discounted) total reward. To learn an optimal policy, agents need to balance exploration and exploitation.

Many RL algorithms are based on building an action-value function, $Q : S \times A \mapsto \mathbb{R}$, which maps state-action pairs to their expected return. In large or continuous state spaces, agents typically factor the state into features: $s = \langle x_1, x_2, \ldots, x_n \rangle$. In such cases, RL methods typically use function approximation to represent the Q-function, which produces state space abstraction.

2.2 Humans Teaching Agents

Some methods for humans teaching agents are relevant to the current work. For instance, *Learning from Demonstration* (LfD) [2] includes a broad category of work that focuses on agents learning to mimic a human demonstrator. Much of the existing LfD research focuses on compensating for variance/errors in actions demonstrated by humans and determining where the estimate of the human action is accurate. LfD methods are naturally centered on the student, whereas the current work centers on a (non-human) teacher.

Inverse reinforcement learning [1] is another increasingly popular paradigm. In this setting, an agent must act in an MDP without a reward signal. The agent observes a human, tries to infer the human's reward function, and then maximizes this function. In contrast, in this work we assume the MDP does have a reward signal, and we do not assume that the student will do best by imitating the teacher.

There is a wide range of other work on providing human help to an agent, such as giving high-level programmatic advice [10]. These methods may inform future work with agent teachers, but they would require teachers and students to be able to communicate more than just immediate action advice.

2.3 Agents Teaching Agents

In *transfer learning* (TL), an agent uses knowledge from a source task to aid its learning in a target task [16, 17, 18]. Often, the agent is allowed to copy source-task knowledge directly, and then continues to learn from that starting point. In contrast, this work assumes that a student agent cannot directly transfer all knowledge completely and immediately from its teacher, because their internal knowledge representations are not known *a priori*, or are even incompatible.

Probabilistic policy reuse (PPR) is a TL technique in which the agent uses a transferred policy with probability ψ, explores with probability ϵ, and exploits the current policy with probability $1 - \psi - \epsilon$ [5, 6]. By decaying ψ over time, the agent can initially leverage transferred knowledge, and then learn to improve upon it. The PPR framework will be used later in this paper for balancing the need to exploit the teacher's knowledge with the need for the student to learn autonomously.

Ask For Help is a method for agents to learn from other expert agents [3, 4]. In this system, an agent asks for advice when its confidence in a state is low. Our work builds upon the idea of making decisions based upon confidence in a state, but we consider the teacher's confidence as well as the student's. Another difference is that we have the teacher make the advice decisions, since it is more knowledgeable than the student.

Experience replay [9] has been successfully used to share recorded experiences between agents in Q-Learning and in batch reinforcement settings [8], but it requires the teacher to store experience samples, and the student must have an identical state representation. Tan [15] extends this idea, allowing agents to share episodes and entire policies, but is also restricted to Q-learning agents with identical representations.

Learning by watching [19] is another example of experience sharing, improved upon by *imitation learning*, which allows agents to have different action sets but still requires them to have the same state representation [12].

Others, including Nunes and Oliveira [11], have considered groups of agents simultaneously learning in a single environment, where agents share experience among themselves. In contrast, this work focuses on teaching rather than cooperative learning.

3. METHODS

Our goal in this paper is to explore ways that agent teachers can help agent students to learn sequential decision tasks. A core assumption is that agents cannot necessarily understand each others' internal workings and thus are limited to teaching via communication, rather than other forms of knowledge transfer (e.g., directly copying a Q-function). This, of course, is the case when humans teach each other as well.

Humans often teach skills by guiding students' actions completely at first, then reducing guidance gradually as students become more capable. We propose a similar approach for agent teachers, and we begin by applying Probabilistic Policy Reuse [5, 6].

PPR allows an agent to learn a task faster by taking advantage of an existing policy. The PPR method, described in Algorithm 1, changes only the action selection step of the learning process. With probability ψ, the agent exploits an old policy; the rest of the time, it uses normal ϵ-greedy action selection. The value of ψ decays over time according to a decay rate v so that the agent makes less use of old policies as it improves its own.

Method 1 Probabilistic Policy Reuse: learning a new policy with assistance from an old policy

```
Load old policy π_old
Initialize new policy π_new
Set ψ = 1
for each training episode do
    for each state s do
        if random(0,1) < ψ then
            Take action π_old(s)
        else if random(0,1) < ε then
            Take a random action
        else
            Take action π_new(s)
        end if
        Update policy
    end for
    Decay ψ = ψv
end for
```

PPR has some good characteristics for teaching. It allows a teacher to give advice frequently at first and less frequently over time. However, recall another core assumption we make: teachers do not always have complete expertise. They may have more knowledge in some states than others and could even have some incorrect knowledge. This can also be true in human teaching as well, though we may be less likely to admit it.

For a teacher with limited expertise, PPR may not be the optimal teaching algorithm. A PPR teacher provides action advice with a global probability ψ that is uniform across all states. If the teacher is more confident in some states than others, it makes more sense for advice probabilities to be higher in some states than others. We therefore propose several new algorithms that use *teacher confidence* to make advice probabilities state-specific.

We will consider the student's confidence eventually as well. However, we focus first on the teacher's confidence, because of the issue of limited expertise.

3.1 Measuring Confidence

To support these new algorithms, we need a way to estimate an agent's confidence in a state. A common approach to this problem is *Q-value interval estimation*, where confidence is measured by the difference between the highest and lowest Q-values in a

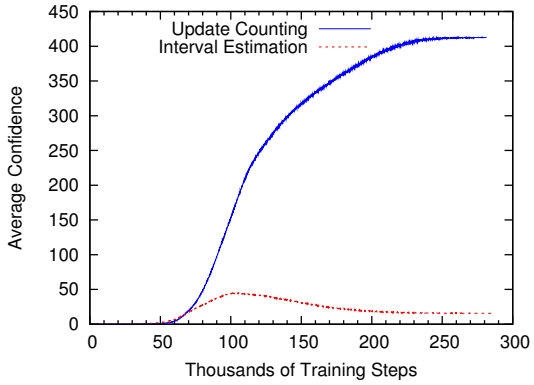


Figure 1: This graph shows how an agent's average confidence per state changes while learning to navigate a maze, using two different confidence measures.

state [4]. However, the interval-estimation measure produces counterintuitive results in some domains.

For example, consider an agent learning to navigate a maze. The agent gains confidence as it first discovers paths to the goal state, but after a while it begins to *lose* confidence, as shown in Figure 1. This is because maze states are highly connected — the true Q-values of actions in these states are not very different. Interval estimation is therefore not a stable confidence measure for maze-like domains. Since such domains are used in our experiments, we do not use interval estimation.

Instead, we introduce *visit counting*, an approach that measures confidence in terms of the number of times an agent has visited a state. It is easily implemented in discrete settings by keeping a table of state visit-counts. When an agent visits a state s, it increments the visit-count $v(s)$, and its confidence in that state is $v(s)$. However, the visit-counting approach is also adaptable to continuous settings. For instance, when using tile-coding, we can use *tile* visit-counts, rather than state visit-counts. When an agent visits a state with active tiles $\langle t_1, t_2, \ldots, t_n \rangle$, it increments the tile visit-counts $v(t_1), v(t_2), ..., v(t_n)$, and its confidence in that state is $\sum_i v(t_i)/n$.

However, we note that visiting a state does not always give an agent more knowledge about it. It seems clear that knowledge is gained only if the agent makes a non-zero Q-value update. It may be possible to be even more restrictive than this, depending on the domain and the learning algorithm. If small negative rewards are received at each step by default, true knowledge gains may be represented only by *positive* Q-value updates. Or, if an agent uses optimistic Q-value initialization, knowledge gains may be represented only by *negative* Q-value updates.

We therefore suggest a metric that we label *update counting*, in which confidence is measured by the number of times an agent has made a non-trivial Q-value update in a state. The definition of "non-trivial" is necessarily domain-dependent and agent-dependent. Update counts extend to continuous settings with tile coding the same way that visit-counts do. This measure has more intuitive behavior than interval estimation, as Figure 1 shows.

3.2 State-Specific Probabilities

We now describe several new teaching algorithms that extend PPR for use with non-expert teachers. These algorithms all make use of a global probability ψ, but they also use confidence measures to compute state-specific advice probabilities.

A state-specific advice probability should have several general properties. First, it should decay over time. Second, it should be higher in states where the teacher is more confident. Third, teachers with lower confidence levels should give less advice overall than teachers with higher confidence levels. Fourth, for high-performing teachers, the state-specific probabilities should all start to converge to ψ.

Note that we calculate an advice *probability* in each state, not just a binary decision on whether or not to give advice. This allows the teacher to smoothly decrease its guidance over time, and cleanly integrates with the PPR framework.

We propose three algorithms that meet these specifications in different ways. In a state s, a teacher must compute a probability of giving advice $p(s)$. Let $c_t(s)$ represent the teacher's confidence in that state.

Our first algorithm computes:

$$p(s) = \begin{cases} 0 & \text{if } c_t(s) < 1 \\ \psi & \text{if } c_t(s) \geq 1 \end{cases}$$

We label this algorithm *conditional-PPR*, since its advice is conditional upon having at least some confidence. We use the threshold $c_t(s) = 1$ as the confidence cutoff in order to draw the line clearly between *no knowledge* and *some knowledge*. This approach provides a simple but logical alternative to PPR that allows a teacher to avoid giving advice in unfamiliar states. For a teacher who is confident in all states, this algorithm becomes equivalent to regular PPR.

Our second algorithm computes:

$$p(s) = \begin{cases} 0 & \text{if } c_t(s) < 1 \\ \psi \frac{c_t(s)+f}{\max(c_t)+f} & \text{if } c_t(s) \geq 1 \end{cases}$$

Here $\max(c_t)$ represents the maximum confidence level the teacher has experienced in any state during its training.

We label this algorithm *proportional-PPR*, since its advice probability is proportional to confidence. In states with higher confidence, it gives advice with a higher probability, up to a maximum of ψ. This approach allows the teacher to scale its level of guidance more precisely in different states.

The floor parameter $f \geq 0$ can be used to shrink the range of probabilities this function produces. As f increases, the minimum advice probability for states with non-zero confidence rises. In the limit, this algorithm becomes equivalent to conditional-PPR.

Our third algorithm computes:

$$p(s) = \begin{cases} 0 & \text{if } c_t(s) < 1 \\ \min\left(1 - \frac{c_s(s)}{c_t(s)+d}, \psi\right) & \text{if } c_t(s) \geq 1 \end{cases}$$

Here $c_s(s)$ represents the *student's* confidence in state s.

We label this algorithm *relative-PPR*, since its advice probability depends on the relationship between the student's confidence and the teacher's confidence. In states where the teacher has much higher confidence than the student, it gives advice with a higher probability, up to a maximum of ψ. As the student's confidence in a state grows, the advice probability decreases, and eventually it stops altogether.

The delay parameter $d \geq 0$ can be used to slow down this process. As d increases, teachers require students to reach higher levels of confidence before they stop giving advice. In the limit, this algorithm also becomes equivalent to conditional-PPR.

This approach combines the probability concepts of PPR, the confidence concepts of Ask-For-Help, and the additional concept of comparing student and teacher confidence. Of our three algorithms, it determines advice probabilities in the most sophisticated way.

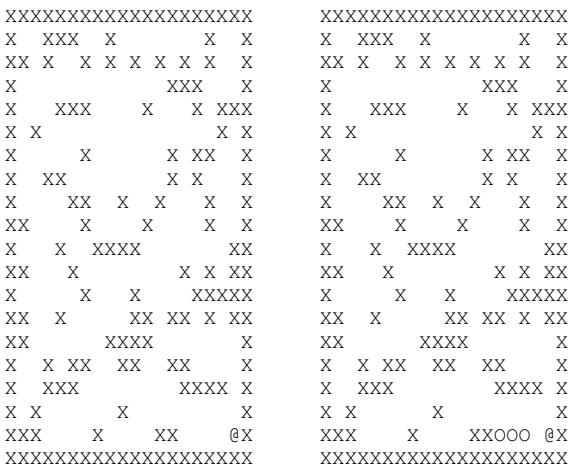


Figure 2: This figure shows the maze world (left) and the cliff world (right). Wall cells are labeled X, cliff cells are labeled O, and the @ is the goal cell.

4. EVALUATION

To evaluate and compare these teaching algorithms, we perform teaching experiments in three domains: maze world, cliff world, and mountain car. The goal in all of these domains is to complete episodes using a minimum number of steps.

Our maze world comes from the Ask-For-Help work [4]. It is a 20x20 grid in which each open cell is a different state, and the agent starts in a random state. Actions corresponding to the four cardinal directions allow the agent to move between cells. The reward function is 0 for every transition except one in which the agent enters the goal cell. Reaching the goal ends the episode, and there is no limit to the number of steps an agent can take.

Our cliff world is identical to the maze world, except that it contains cells that represent pits, arranged in a row like a cliff. If an agent enters a cliff cell, it gets reset back to its starting position. Figure 2 shows an illustration of the maze and cliff environments.

Mountain Car is a benchmark RL task, commonly used to test algorithms in a simple continuous state task [13]. The car starts near the bottom of the hill with zero velocity, and must drive to the top of the hill. However, its motor is underpowered, so the car must build up enough energy to reach the goal by moving back and forth. Figure 3 shows an illustration of this environment.

States in this domain are described by two continuous features: the car's position and velocity. The agent has three actions: accelerate in the +x direction, accelerate in the -x direction, or do not accelerate at all. The transition function is a simple physics simulation involving position, velocity, acceleration, and gravity. The car is given a reward of -1 at every step until it reaches the goal location. Reaching the goal ends the episode, but there is also a limit of 1000 steps per episode.

We train agents in these worlds with learning algorithms that are appropriate to their setting. In the maze world, we use simple tabular Q-learning. In the cliff world, we use tabular Sarsa, since the Sarsa variant of Q-learning is better for handling state spaces where exploration can be extremly costly [14]. In mountain car, we use Sarsa(λ) with 16 tile codings to handle the continuous features.

In each learning algorithm, we use default parameters for these domains that produce reasonable learning curves for independent agents. For the maze and cliff, these are $\alpha = 0.15$, $\epsilon = 0.05$, and $\gamma = 0.99$. For mountain car, they are $\alpha = 0.25$, $\epsilon = 0.05$, $\gamma = 1.0$, and $\lambda = 0.25$.

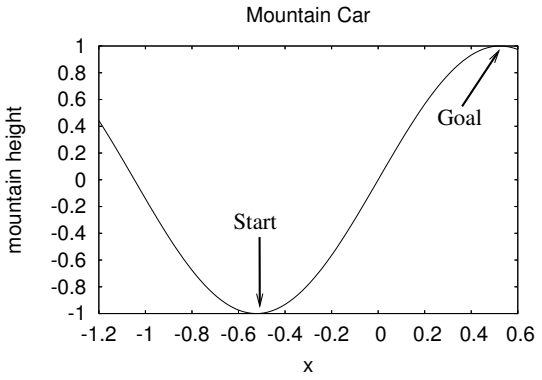


Figure 3: This figure shows the traditional mountain car environment, where x is the agent's location.

To produce teachers with limited expertise, we do not allow them to train until their policies converge. Instead, we train teachers for only 10, 20, or 30 episodes. In all the domains, we allow teacher episodes to be as long as necessary for the teacher to reach a goal state. Otherwise, two teachers who have trained for 10 episodes could have very different confidence levels, which would be a large source of unnecessary variability in the experiments.

Each teacher then gives advice to students using the four teaching algorithms we have discussed: regular PPR, conditional-PPR, proportional-PPR, and relative-PPR. We use the update-count metric for estimating confidence. For the maze and cliff worlds, we use state visit-counts that are incremented when a visit to a state produces a non-zero Q-value update. For mountain car, we use tile visit-counts that are incremented when a visit to a state produces a positive Q-value update.

Each teaching algorithm also has parameters that must be set. They all share the decay-rate parameter v; proportional-PPR also has the floor parameter f and relative-PPR has the delay parameter d. Since appropriate values for these parameters are domain-dependent, we determine a set of 3 reasonable choices for each parameter in each domain and select the best of those settings.

To produce smooth learning curves for the students, we average the performance of 100 students for each experiment. To lend perspective to the student learning curves, we also show curves for *independent* agents and *direct-transfer* agents. Independent agents learn without a teacher, and should therefore learn more slowly than any student. Direct-transfer agents copy the teacher's entire Q-function, and should therefore learn more quickly than all other students. In fact, because they have direct access to their teacher's brains, direct-transfer agents should represent a bound for student performance.

To compare the algorithms quantitatively as well as visually, we compute areas under learning curves and perform paired t-tests to check for significant differences between algorithms.

4.1 Maze World Results

Figures 4, 5, and 6 show how students learn from teachers with varying levels of expertise in the maze world. As expected, teachers with more training are more helpful to their students. However, the PPR algorithm provides some benefits to students at all levels of training, and our state-specific versions produce similar results.

At the lowest training level, regular PPR is more beneficial than the rest, and this difference is statistically significant. This result is surprising, since it means that giving advice in unfamiliar maze

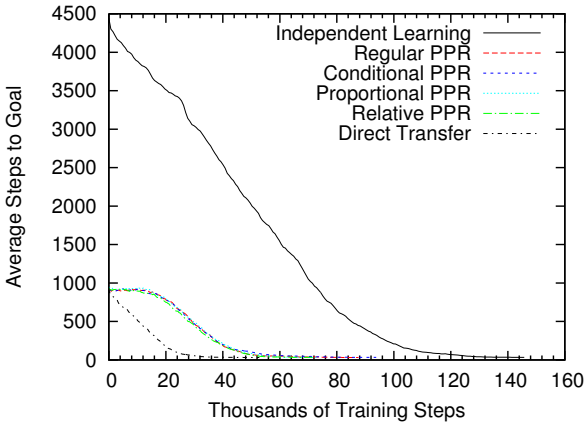


Figure 4: Maze world performance of students whose teachers had 30 episodes of training

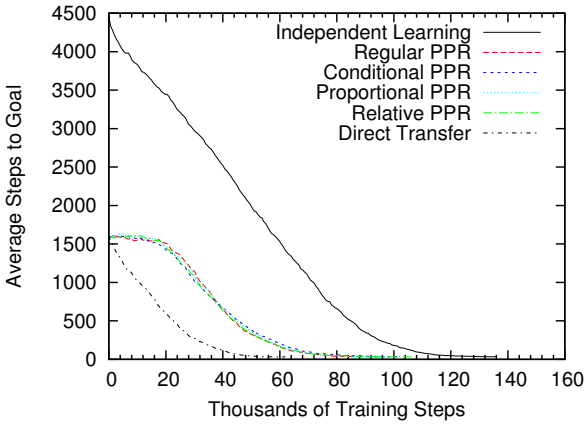


Figure 5: Maze world performance of students whose teachers had 20 episodes of training

states is somehow helpful. Further inspection reveals that advice in these states is random, and thus equivalent to exploration. This changes the student's exploration rate from constant ϵ to something correlated with decaying ψ. If we let all the students use $\epsilon = \psi$, the difference between regular PPR and the other approaches disappears. This indicates that the student's exploration rate can have a significant impact on the effectiveness of a teaching algorithm.

We find that reasonable parameter settings in the maze world are $v \in \{0.9, 0.99, 0.999\}$, $f \in \{10, 100, 1000\}$, and $d \in \{0, 10, 100\}$. Table 1 shows the best settings for each algorithm. Note that the floor parameters for proportional-PPR tend to be the higher ones, which makes that approach comparable to conditional-PPR. This means a teacher in the maze world does best using even small amounts of knowledge to their fullest degree. The delay rate d for relative-PPR is not particularly important; all the choices produce similar results. This means relative-PPR is not very sensitive to parameter selection in this domain.

One other important difference between the teaching algorithms in the maze world is the amount of advice they provide to students. Table 2 shows the average number of times each algorithm gives advice while training the students. The state-specific approaches give drastically less advice than regular PPR. They may therefore

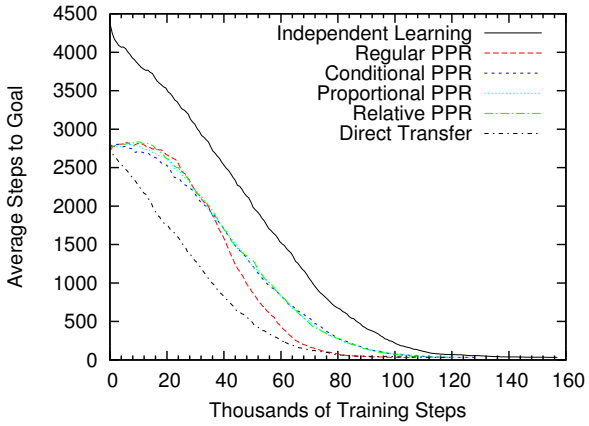


Figure 6: Maze world performance of students whose teachers had 10 episodes of training

	Training = 30	Training = 20	Training = 10
PPR	$v = 0.99$	$v = 0.99$	$v = 0.99$
c-PPR	$v = 0.99$	$v = 0.999$	$v = 0.99$
p-PPR	$v, f = 0.999, 100$	$v, f = 0.99, 1000$	$v, f = 0.99, 1000$
r-PPR	$v, d = 0.999, 0$	$v, d = 0.99, 0$	$v, d = 0.9, 100$

Table 1: Best parameter settings in the maze world

be preferable in this domain simply for their advice efficiency.

The maze world contains no critical decision points; there is no state in which a particular action is crucial to take or avoid. Teachers can therefore give some non-optimal advice in this domain without causing harm. This is the reason that regular PPR performs comparably to state-specific versions in the maze world. Our next experiments in the cliff world will tell a different story.

4.2 Cliff World Results

Figures 7, 8, and 9 show how students learn from teachers with varying levels of expertise in the cliff world. In these experiments, all of the state-specific algorithms are more beneficial than regular PPR, and these differences are statistically significant.

The reason for this change is that cliffs make it dangerous to give advice in unfamiliar states, as is done by regular PPR. States near cliffs are critical decision points. Some teachers become familiar with the states near the cliff during their limited training, but others do not. Our algorithms allow only the confident teachers to give advice in those states. The more critical decision points are in a domain, the more superior state-specific approaches should be for teachers with limited expertise.

Reasonable parameter settings in the cliff world are the same as in the maze world. Table 3 shows the best settings for each algorithm. The amounts of advice given by teachers in the cliff world are shown in Table 4. As in the maze world, the state-specific approaches give drastically less advice than regular PPR.

	Training = 30	Training = 20	Training = 10
PPR	55,728	112,505	209,512
conditional-PPR	1,275	2,025	617
proportional-PPR	2,496	1,121	623
relative-PPR	592	353	225

Table 2: Amounts of advice given by teachers while training the students in the maze world

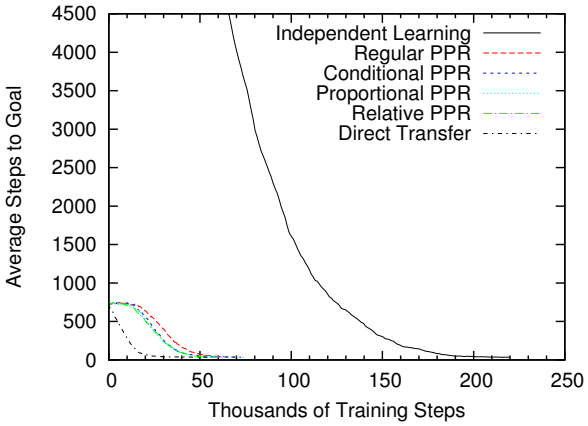


Figure 7: Cliff world performance of students whose teachers had 30 episodes of training

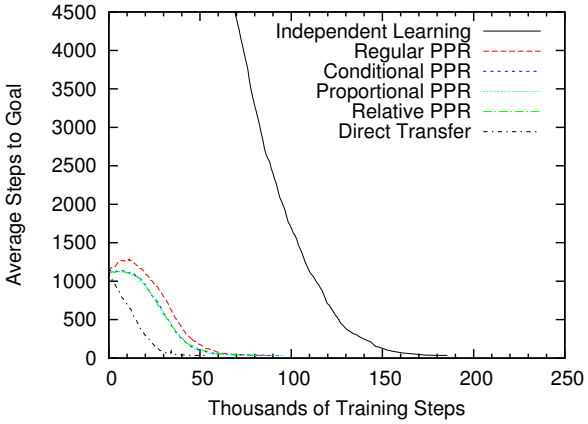


Figure 8: Cliff world performance of students whose teachers had 20 episodes of training

4.3 Mountain Car Results

Figures 10, 11, and 12 show how students learn from teachers with varying levels of expertise in mountain car. In these experiments, conditional-PPR and proportional-PPR produce similar results to regular PPR. However, relative-PPR is more beneficial, and these differences are statistically significant.

These results can be best understood by looking at the confidence mechanics in mountain car. Since it is a tile-coding domain, update counts are assigned to tiles rather than states, and an agent's confidence in a state is the average update-count of the state's active tiles. One effect of spreading confidence across tiles is that confidence tends to grow quickly throughout the state space. Unseen states can still receive confidence if their component tiles have been seen (due to visiting nearby states). It does not take long for an agent to have at least some confidence in nearly all states.

In this situation, conditional-PPR and proportional-PPR should approach equivalency with regular PPR, and their performance reflects this. Relative-PPR is better equipped to handle these confidence mechanics, because it takes both teacher and student confidence into account. It backs off quickly in states where the student gains confidence quickly, but keeps giving advice in the less common states.

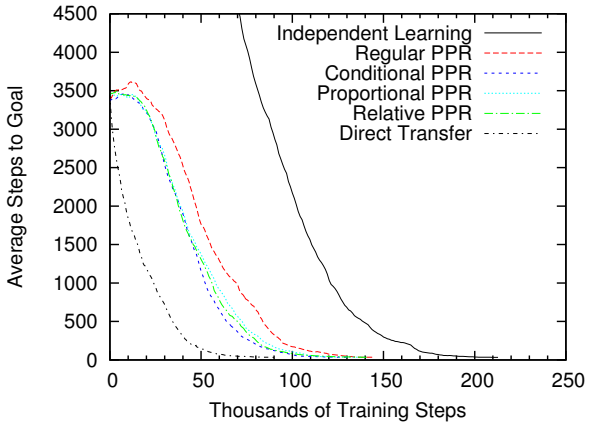


Figure 9: Cliff world performance of students whose teachers had 10 episodes of training

	Training = 30	Training = 20	Training = 10
PPR	$v = 0.99$	$v = 0.9$	$v = 0.9$
c-PPR	$v = 0.99$	$v = 0.999$	$v = 0.99$
p-PPR	$v, f = 0.999, 1000$	$v, f = 0.999, 1000$	$v, f = 0.999, 1000$
r-PPR	$v, d = 0.999, 0$	$v, d = 0.999, 100$	$v, d = 0.99, 10$

Table 3: Best parameter settings in the cliff world

We find that reasonable parameter settings in mountain car are $v \in \{0.99, 0.97, 0.95\}$, $f \in \{100, 1000, 10000\}$, and finally $d \in \{10, 100, 1000\}$. Table 5 shows the best settings for each algorithm. Note that the parameters for relative-PPR are more important in this domain; the performance of the approach varies more than it does in the maze and cliff worlds.

The amounts of advice given by teachers in mountain car are shown in Table 6. The state-specific approaches use only slightly less advice than regular PPR. Since teachers tend to have at least some confidence in nearly all states, they give substantially more advice.

5. FUTURE WORK AND CONCLUSIONS

The literature on learning agents naturally focuses on algorithms that agents can use to learn. This paper contributes an initial study of algorithms that agents can use to *teach*. It focuses on agents teaching other agents in sequential decision tasks. We assume a broadly applicable setting, in which teachers and students interact through action advice and in which teachers have limited expertise.

This paper contributes a family of teaching methods for advising students. The algorithms are based on Probabilistic Policy Reuse, but they use the concept of agent confidence to make advice probabilities state-specific. Empirical results show that state-specific methods, particularly one that takes both teacher and student confidence levels into account, are effective in the teaching setting.

	Training = 30	Training = 20	Training = 10
PPR	53,509	30,489	114,102
conditional-PPR	1,562	2,547	962
proportional-PPR	3,060	2,527	1,664
relative-PPR	728	1,255	416

Table 4: Amounts of advice given by teachers while training the students in the cliff world

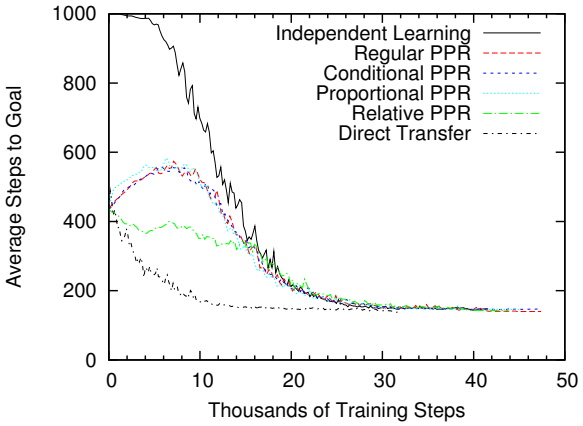


Figure 10: Mountain car performance of students whose teachers had 30 episodes of training

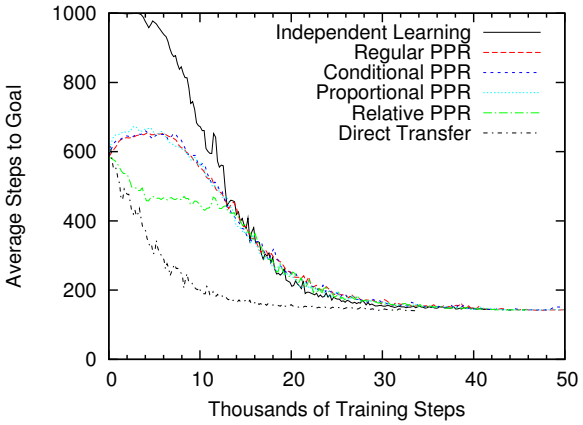


Figure 11: Mountain car performance of students whose teachers had 20 episodes of training

There are many potential directions for future work in this area. Teachers could explicitly reason about the expense of communication versus the expected gain, which would be appropriate in domains where communication has a non-zero cost. Teachers could use student experience to adjust their confidence levels. There could also be multiple teachers, with different areas of expertise, that must coordinate with each other.

Students could also be given more active roles than they currently have in this work. For instance, a student could estimate its teacher's performance and decide whether or not to use the advice it receives in a certain area of the state space, or select which advice to follow when multiple teachers are present.

Finally, algorithms that allow agents to teach each other may also inform strategies for agents to teach humans. Agents could make particularly patient teachers, and using some of the ideas in this paper, they could also be responsive to student learning. For instance, an agent teacher could attempt to estimate a human's confidence through visit counts, reaction times, and other non-verbal cues, and then use this estimate to decide whether to provide advice.

We hope that this paper encourages others to continue studying inter-agent teaching, as well as providing a set of algorithms and results to serve as benchmarks.

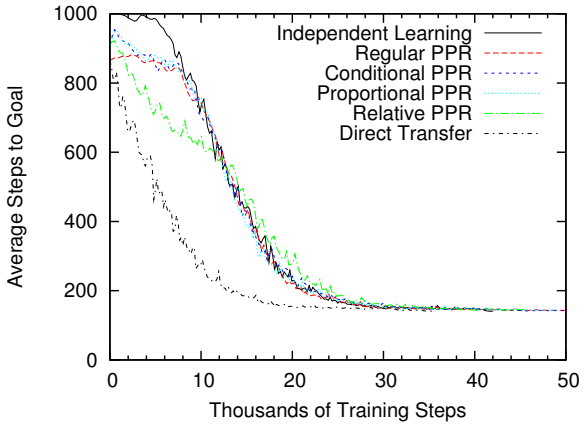


Figure 12: Mountain car performance of students whose teachers had 10 episodes of training

	Training = 30	Training = 20	Training = 10
PPR	$v = 0.97$	$v = 0.97$	$v = 0.95$
c-PPR	$v = 0.97$	$v = 0.97$	$v = 0.95$
p-PPR	$v, f = 0.97, 1000$	$v, f = 0.97, 1000$	$v, f = 0.95, 1000$
r-PPR	$v, d = 0.99, 100$	$v, d = 0.99, 100$	$v, d = 0.99, 100$

Table 5: Best parameter settings in mountain car

6. REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 2004.

[2] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[3] J. A. Clouse. An introspection approach to querying a trainer. Technical Report 96-13, University of Massachusetts, Amherst, MA, USA, 1996.

[4] J. A. Clouse. *On integrating apprentice learning and reinforcement learning*. PhD thesis, University of Massachusetts, 1996.

[5] F. Fernández, J. García, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010. Advances in Autonomous Robots for Service and Entertainment.

[6] F. Fernandez and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*, 2006.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[8] S. Kalyanakrishnan and P. Stone. Batch reinforcement

	Training = 30	Training = 20	Training = 10
PPR	21,911	25,426	31,769
conditional-PPR	21,840	23,983	21,951
proportional-PPR	21,745	23,782	21,857
relative-PPR	18,859	19,806	20,687

Table 6: Amounts of advice given by teachers while training the students in mountain car

learning in a complex domain. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 650–657, May 2007.

[9] L. J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[10] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3):251–281, 1996.

[11] L. Nunes and E. Oliveira. Learning from multiple sources. In *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 1106–1113, 2004.

[12] B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[13] S. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[14] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[15] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

[16] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[17] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[18] L. Torrey, T. Walker, J. W. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 412–424, 2005.

[19] S. D. Whitehead. A complexity analisys of cooperative mechanisms in reinforcement learning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 607–613, 1991.

Plan-Based Reward Shaping for Multi-Agent Reinforcement Learning

Sam Devlin
Department of Computer Science,
University of York, UK
devlin@cs.york.ac.uk

Daniel Kudenko
Department of Computer Science,
University of York, UK
kudenko@cs.york.ac.uk

ABSTRACT

Recent theoretical results have justified the use of potential-based reward shaping as a way to improve the performance of multi-agent reinforcement learning (MARL). However, the question remains of how to generate a useful potential function.

Previous research demonstrated the use of STRIPS operator knowledge to automatically generate a potential function for single-agent reinforcement learning. Following up on this work, we investigate the use of STRIPS planning knowledge in the context of MARL.

Our results show that a potential function based on joint or individual plan knowledge can significantly improve MARL performance compared with no shaping. In addition, we investigate the limitations of individual plan knowledge as a source of reward shaping in cases where the combination of individual agent plans causes conflict.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Experimentation

Keywords

Reinforcement Learning, Reward Shaping

1. INTRODUCTION

Using reinforcement learning agents in multi-agent systems (MAS) is often considered impractical due to an exponential increase in the state space with each additional agent. Whilst assuming other agents' actions to be part of the environment can save from having to calculate the value of all joint-policies, the time taken to learn a suitable policy can become impractical as the environment now appears stochastic.

One method, explored extensively in the single-agent literature, to reduce the time to convergence is reward shaping. Reward shaping is the process of providing prior knowledge to an agent through additional rewards. These rewards help direct an agent's exploration, minimising the number of sub-optimal steps it takes and so directing it towards the optimal policy quicker.

Recent work has justified the use of these methods in multi-agent reinforcement learning and so now our interest shifts towards how to encode knowledge commonly available. Previous research, again from the single-agent literature, translated knowledge encoded as STRIPS operators into a potential function for reward shaping [12]. In this paper we will discuss our attempts to use this approach in MAS with either coordinated plans made together or greedy plans made individually. Both are beneficial to agents but the former more so. However, planning together will not always be possible in practice and, therefore, we also present a subsequent investigation into how to overcome conflicted knowledge in individual plans.

The next section begins by introducing the relevant background material and existing work in multi-agent reinforcement learning, reward shaping and planning. Section 3 goes on then to describe our novel combination of these tools. The bulk of experimentation and analysis is in Sections 4, 5 and 6. Finally, in the closing section we conclude with remarks on the outcomes of this study and relevant future directions.

2. BACKGROUND

In this section we introduce all relevant existing work upon which this investigation is based.

2.1 Multi-Agent Reinforcement Learning

Reinforcement learning is a paradigm which allows agents to learn by reward and punishment from interactions with the environment [25]. The numeric feedback received from the environment is used to improve the agent's actions. The majority of work in the area of reinforcement learning applies a Markov Decision Process (MDP) as a mathematical model [19].

An MDP is a tuple $\langle S, A, T, R \rangle$, where S is the state space, A is the action space, $T(s, a, s') = Pr(s'|s, a)$ is the probability that action a in state s will lead to state s', and $R(s, a, s')$ is the immediate reward r received when action a taken in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and reward function) are available, this task can be solved using dynamic programming [3].

When the environment dynamics are not available, as with most real problem domains, dynamic programming cannot be used. However, the concept of an iterative approach remains the backbone of the majority of reinforcement learn-

ing algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states, $V(s)$, or state-action pairs, $Q(s, a)$ [24]. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. The SARSA algorithm is such a method [25]. After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \quad (1)$$

where α is the rate of learning and γ is the discount factor. It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'.

It is important whilst learning in an environment to balance exploration of new state-action pairs with exploitation of those which are already known to receive high rewards. A common method of doing so is $\epsilon-$greedy. When using this method the agent explores, with probability ϵ, by choosing a random action or exploits its current knowledge, with probability $1 - \epsilon$, by choosing the highest value action for the current state [25].

Temporal-difference algorithms, such as SARSA, only update the single latest state-action pair. In environments where rewards are sparse, many episodes may be required for the true value of a policy to propagate sufficiently. To speed up this process, a method known as eligibility traces keeps a record of previous state-action pairs that have occurred and are therefore eligible for update when a reward is received. The eligibility of the latest state-action pair is set to 1 and all other state-action pairs' eligibility is multiplied by λ (where $\lambda \leq 1$). When an action is completed all state-action pairs are updated by the temporal difference multiplied by their eligibility and so Q-values propagate quicker [25].

Applications of reinforcement learning to MAS typically take one of two approaches; multiple individual learners or joint action learners [4]. The latter is a group of multi-agent specific algorithms designed to consider the existence of other agents. The former is the deployment of multiple agents each using a single-agent reinforcement learning algorithm.

Multiple individual learners assume any other agents to be a part of the environment and so, as the others simultaneously learn, the environment appears to be dynamic as the probability of transition when taking action a in state s changes over time. To overcome the appearance of a dynamic environment, joint action learners were developed that extend their value function to consider for each state the value of each possible combination of actions by all agents.

Learning by joint action, however, breaks a common fundamental concept of MAS. Specifically, each agent in a MAS is self-motivated and so may not consent to the broadcasting of their action choices as required by joint action learners. Furthermore, the consideration of the joint action causes an exponential increase in the number of values that must be calculated with each additional agent added to the system. For these reasons, this work will focus on multiple individual learners and not joint action learners. However, it is expected that the application of these approaches to joint action learners would have similar benefits.

Typically, reinforcement learning agents, whether alone or sharing an environment, are deployed with no prior knowledge. The assumption is that the developer has no knowledge of how the agent(s) should behave. However, more often than not, this is not the case. As a group we are interested in knowledge-based reinforcement learning, an area where this assumption is removed and informed agents can benefit from prior knowledge. One common method of imparting knowledge to a reinforcement learning agent is reward shaping, a topic we will discuss in more detail in the next subsection.

2.2 Multi-Agent and Plan-Based Reward Shaping

The idea of reward shaping is to provide an additional reward representative of prior knowledge to reduce the number of suboptimal actions made and so reduce the time needed to learn [17, 20]. This concept can be represented by the following formula for the SARSA algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, s') + \gamma Q(s', a') - Q(s, a)] \quad (2)$$

where $F(s, s')$ is the general form of any state-based shaping reward.

Even though reward shaping has been powerful in many experiments it quickly became apparent that, when used improperly, it can change the optimal policy [20]. To deal with such problems, potential-based reward shaping was proposed [17] as the difference of some potential function Φ defined over a source s and a destination state s':

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \quad (3)$$

where γ must be the same discount factor as used in the agent's update rule (see Equation 1).

Ng et al. [17] proved that potential-based reward shaping, defined according to Equation 3, does not alter the optimal policy of a single agent in both infinite- and finite- state MDPs.

However, in multi-agent reinforcement learning the goal is no longer the single agent's optimal policy. Instead some compromise must be made and so agents are typically designed instead to learn a Nash equilibrium [16, 22]. For such problem domains, it has been proven that the Nash equilibria of a MAS are not altered by any number of agents receiving additional rewards provided they are of the form given in Equation 3 [8].

Recent theoretical work has extended both the single-agent guarantee of policy invariance and the multi-agent guarantee of consistent Nash equilibria to cases where the potential function is dynamic [9].

Reward shaping is typically implemented bespoke for each new environment using domain-specific heuristic knowledge [2, 7, 20] but some attempts have been made to automate [11, 15] and semi-automate [12] the encoding of knowledge into a reward signal. Automating the process requires no previous knowledge and can be applied generally to any problem domain. The results are typically better than without shaping but less than agents shaped by prior knowledge. Semi-automated methods require prior knowledge to be put in but then automate the transformation of this knowledge into a potential function.

Plan-based reward shaping, an established semi-automated method in single-agent reinforcement learning, uses a STRIPS planner to generate high-level plans. These plans are encoded into a potential function where states later in the plan receive a higher potential than those lower or not in

the plan. This potential function is then used by potential-based reward shaping to encourage the agent to follow the plan without altering the agent's goal. The process of learning the low-level actions necessary to execute a high-level plan is significantly easier than learning the low-level actions to maximise reward in an unknown environment and so with this knowledge agents tend to learn the optimal policy quicker. Furthermore, as many developers are already familiar with STRIPS planners, the process of implementing potential-based reward shaping is now more accessible and less domain specific. [12]

In this investigation we explore how multi-agent planning, introduced in the following subsection, can be combined with this semi-automatic method of reward shaping.

2.3 Multi-Agent Planning

The generation of multi-agent plans can occur within one centralised agent or spread amongst a number of agents [21, 26].

The centralised approach benefits from full observation making it able to, where possible, satisfy all agents' goals without conflict. However, much like joint-action learning, this approach requires sharing of information, such as goals and abilities, that agents in a MAS often will not want to share.

The alternative approach, allowing each agent to make their own plans, will tend to generate conflicting plans. Many methods of coordination have been attempted including, amongst others, social laws [23], negotiation [26] and contingency planning [18] but still this remains an ongoing area of active research [6].

In the next section we will discuss how plans generated by both of these methods can be used with plan-based reward shaping to aid multiple individual learners.

3. MULTI-AGENT, PLAN-BASED REWARD SHAPING

Based on the two opposing methods of multi-agent planning, centralised and decentralised, we propose two methods of extending plan-based reward shaping to multi-agent reinforcement learning.

The first, joint-plan based reward shaping, employs the concept of centralised planning and so generates where possible plans without conflict. This shaping is expected to outperform the alternative but may not be possible in competitive environments where agents are unwilling to cooperate.

Alternatively, individual-plan-based reward shaping, requires no cooperation as each agent plans as if it is alone in the environment.

Unfortunately, the application of individual-plan-based reward shaping to multi-agent problem domains is not as simple in practice as it may seem. The knowledge given by multiple individual plans will often be conflicted and agents may need to deviate significantly from this prior knowledge when acting in their common environment. Our aim is to allow them to do so. Reward shaping only encourages a path of exploration, it does not enforce a joint-policy. Therefore, it may be possible that reinforcement learning agents, given conflicted plans initially, can learn to overcome their conflicts and eventually follow coordinated policies.

For both methods, the STRIPS plan of each agent is translated into a list of states so that, whilst acting, an agent's current state can be compared to all plan steps. The potential of the agent's current state then becomes:

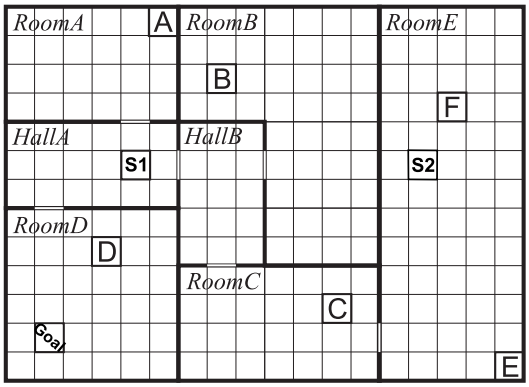


Figure 1: Multi-Agent, Flag-Collecting Problem Domain

$$\Phi(s) = CurrentStepInPlan * \omega \quad (4)$$

where $CurrentStepInPlan$ is the corresponding state in the state-based representation of the agent's plan (for example see the left hand column of Listing 5) and ω is a scaling factor.

If the current state is not in the state-based representation of the agent's plan, then the potential used is that of the last state experienced that was in the plan. This was implemented in the original work to not discourage exploration off of the plan and is now more relevant as we know, in the case of individual greedy plans, that strict adherence to the plan by every agent will not be possible. This feature of the potential function makes plan-based reward shaping an instance of dynamic potential-based reward shaping [9].

To preserve the theoretical guarantees of potential-based reward shaping, the potential of all goal states is set to zero so that it equals the initial state of all agents in the next episode.

These potentials are then used as in Equation 3 to calculate the additional reward given to the agent.

In the next section we will introduce a problem domain and the specific implementations of both our proposed methods in that domain.

4. INITIAL STUDY

Our chosen problem for this study is a grid-world, flag collecting domain with two agents attempting to collect six flags spread across seven rooms. An overview of this world is illustrated in Figure 1 with the goal location labeled as such, each agent's starting location labeled Si where i is their unique id and the remaining labeled grid locations being flags and their unique id.

At each time step an agent can move up, down, left or right and will deterministically complete their move provided they do not collide with a wall or the other agent. Once an agent reaches the goal state their episode is over regardless of the number of flags collected. The entire episode is completed when both agents reach the goal state. At this time both agents receive a reward equal to one hundred times the number of flags they have collected in combination. No other re-

wards are given at any other time. To encourage the agents to learn short paths, the discount factor γ is set to less than one.[1]

Additionally, as each agent can only perceive its own location and the flags it has already picked up, the problem is a DEC-POMDP.

Given this domain, the plans of agent 1 and agent 2 with joint-plan based reward shaping are documented in Listings 1 and 2. It is important to note that these plans are coordinated with no conflicting actions.

Listing 1: Joint-Plan for Agent 1 Starting in HallA

```
MOVE(hallA,roomA)
TAKE(flagA,roomA)
MOVE(roomA,hallA)
MOVE(hallA,hallB)
MOVE(hallB,roomB)
TAKE(flagB,roomB)
MOVE(roomB,hallB)
MOVE(hallB,hallA)
MOVE(hallA,roomD)
```

Listing 2: Joint-Plan for Agent 2 Starting in RoomE

```
TAKE(flagF,roomE)
TAKE(flagE,roomE)
MOVE(roomE,roomC)
TAKE(flagC,roomC)
MOVE(roomC,hallB)
MOVE(hallB,hallA)
MOVE(hallA,roomD)
TAKE(flagD,roomD)
```

Alternatively, Listings 3 and 4 document the plans used to shape agent 1 and agent 2 respectively when receiving individual-plan-based reward shaping. However, now both plans cannot be completed as each intends to collect all flags. How, or if, the agents can learn to overcome this conflicting knowledge is the focus of this investigation.

Listing 3: Individual Plan for Agent 1 Starting in HallA

```
MOVE(hallA,hallB)
MOVE(hallB,roomC)
TAKE(flagC,roomC)
MOVE(roomC,roomE)
TAKE(flagE,roomE)
TAKE(flagF,roomE)
MOVE(roomE,roomC)
MOVE(roomC,hallB)
MOVE(hallB,roomB)
TAKE(flagB,roomB)
MOVE(roomB,hallB)
MOVE(hallB,hallA)
MOVE(hallA,roomA)
TAKE(flagA,roomA)
MOVE(roomA,hallA)
MOVE(hallA,roomD)
TAKE(flagD,roomD)
```

Listing 4: Individual Plan for Agent 2 Starting in RoomE

```
TAKE(flagF,roomE)
TAKE(flagE,roomE)
MOVE(roomE,roomC)
TAKE(flagC,roomC)
MOVE(roomC,hallB)
MOVE(hallB,roomB)
TAKE(flagB,roomB)
MOVE(roomB,hallB)
MOVE(hallB,hallA)
MOVE(hallA,roomA)
TAKE(flagA,roomA)
MOVE(roomA,hallA)
MOVE(hallA,roomD)
TAKE(flagD,roomD)
```

As mentioned in Section 3, these plans must be translated into state-based knowledge. Listing 5 shows this transformation for the joint-plan starting in hallA (listed in Listing 1) and the corresponding value of ω.

In all our experiments, regardless of knowledge used, we have set the scaling factor ω so that the maximum potential of a state is the maximum reward of the environment. As the scaling factor affects how likely the agent is to follow the heuristic knowledge [10], maintaining a constant maximum across all heuristics compared ensures a fair comparison. For enivronments with an unknown maximum reward the scaling factor ω can be set experimentally or based on the designer's confidence in the heuristic.

Listing 5: State-Based Joint-Plan for Agent 1 Starting in HallA

```
0 robot-in_hallA
1 robot-in_roomA
2 robot-in_roomA taken_flagA
3 robot-in_hallA taken_flagA
4 robot-in_hallB taken_flagA
5 robot-in_roomB taken_flagA
6 robot-in_roomB taken_flagA taken_flagB
7 robot-in_hallB taken_flagA taken_flagB
8 robot-in_hallA taken_flagA taken_flagB
9 robot-in_roomD taken_flagA taken_flagB
```

$\omega = MaxReward/NumStepsInPlan = 600/9$

For comparison, we have implemented a team of agents with no prior knowledge/shaping and a team with the domain-specific knowledge that collecting flags is beneficial. These flag-based agents value a state's potential equal to one hundred times the number of flags it alone has collected. This again ensures that the maximum potential of any state is equal to the maximum reward of the environment.

We have also considered the combination of this flag-based heuristic with the general methods of joint-plan-based and individual-plan-based shaping. These combined agents value the potential of a state to be:

$$\begin{aligned} \Phi(s) = &\ (CurrentStepInPlan + NumFlagsCollected) * \omega \\ \omega = &\ MaxReward/(NumStepsInPlan \\ &\ + NumFlagsInWorld) \end{aligned} \quad (5)$$

where $NumFlagsCollected$ is the number of flags the agent has collected itself, $NumStepsInPlan$ is the number of steps in its state-based plan and $NumFlagsInWorld$ is the total number of flags in the world (i.e. for this domain - 6).

All agents, regardless of shaping, implemented SARSA with $\epsilon-$greedy action selection and eligibility traces. For all experiments, the agents' parameters were set such that $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon = 0.1$ and $\lambda = 0.4$. For these experiments, all initial Q-values were zero.

These methods, however, do not require the use of SARSA, $\epsilon-$greedy action selection or eligibility traces. Potential-based reward shaping has previously been proven with Q-learning and RMax [1] and in our own experience works with many multi-agent specific algorithms (including both temporal difference and policy iteration algorithms) except for Distributed-Q [14]. Furthermore, it has been shown before without (but never before to our knowledge with) eligibility traces [17, 1, 7] and proven for any action selection method that chooses actions based on relative difference and not absolute magnitude [1, 8].

All experiments have been repeated thirty times with the mean discounted reward per episode presented in the following graphs. All claims of significant differences are supported by two-tailed, two sample t-tests with significance $p < 0.05$

[1]Experiments with a negative reward on each time step and $\gamma = 1$ made no significant change in the behaviour of the agents.

(unless stated otherwise).

4.1 Results and Conclusions

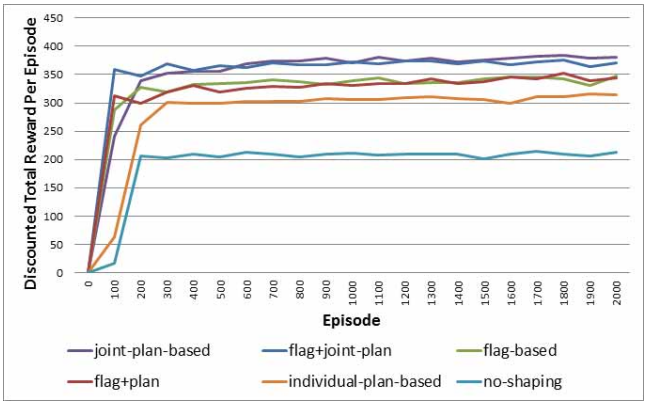


Figure 2: Initial Results

Figure 2 shows all agents, regardless of shaping, learn quickly within the first 300 episodes. In all cases, some knowledge significantly improves the final performance of the agents as shown by all shaped agents out-performing the base agent with no reward shaping.

Agents shaped by knowledge of the optimal joint-plan (both alone or combined with the flag-based heuristic) significantly outperform all other agents, consistently learning to collect all six flags[2]. Figure 3 illustrates the typical behaviour learnt by these agents. Note that in these examples the agents have learnt the low level implementation of the high level plan provided.

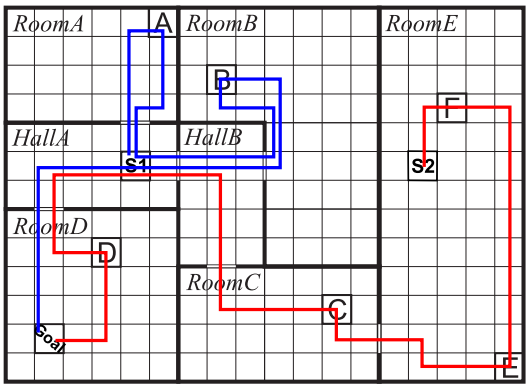


Figure 3: Example Behaviour of Joint-Plan-Based Agents

The individual-plan-based agents are unable to reach the same performance as they are given no explicit knowledge of how to coordinate.

Figure 4 illustrates the typical behaviour learnt by these agents. This time we note that agent 1 has opted out of receiving its shaping reward by moving directly to the goal and not following its given plan. The resultant behaviour allows the agents to receive the maximum goal reward from collecting all flags, but at a longer time delay and, therefore, a significantly greater discount.

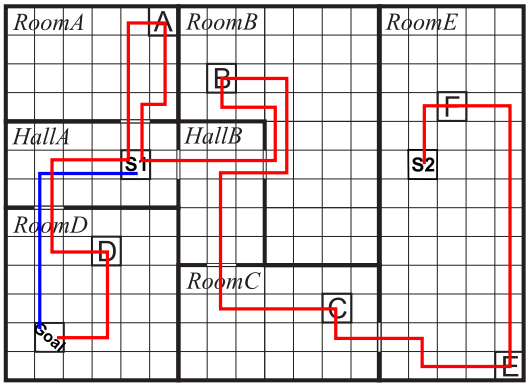


Figure 4: Example Behaviour of Individual-Plan-Based Agents

Occasionally the agents coordinate better with agent 1 collecting flag D or, even rarer, flags D and A. Whilst this is the exception, it is interesting to note that the agent not following its plan will always choose actions that take away from the end of the other agent's plan rather than follow the first steps of their own plan.

The flag based heuristic can be seen to improve coordination slightly in the agents receiving combined shaping from both types of knowledge, but not sufficiently to overcome the conflicts in the two individual plans.

To conclude, some knowledge, regardless of the number of conflicts, is better than no knowledge but coordinated knowledge is more beneficial if available.

Given these initial results, our aim in the following experiments was to try to close the difference in final performance between individual-plan-based agents and joint-plan-based agents by overcoming the conflicted knowledge.

5. OVERCOMING CONFLICTED KNOWLEDGE

In this section we explore options for closing the difference in final performance caused by conflicted knowledge in the individual plans.

One plausible option would be to introduce communication between the agents. Another may be to combine individual-plan-based reward shaping with FCQ-learning [5] to switch to a joint-action representation in states where coordination is required. However, as both multiple individual learners and individual-plan-based reward shaping were designed to avoid sharing information amongst agents, we have not explored these options.

Without sharing information, agent 1 could be encouraged not to opt out of following its plan by switching to a competitive reward function. However, although this closed the gap between individual-plan-based and joint-plan-based agents, the change was detrimental to the team performance of all agents regardless of shaping.

Specifically, individual-plan-based agent 1 did, as expected, start to participate and collect some flags but collectively they would not collect all flags. Both agents would follow their plans to the first two or three flags but then head to

[2]Please note the joint-plan-based agents' illustrated performance in Figure 2 does not reach 600 as the value presented is discounted by the time it takes the agents to complete the episode.

the goal as the next flag would not reliably be there. For similar reasons joint-plan-based agents would also no longer collect all flags. Therefore, the reduction in the gap between individual-plan-based and joint-plan-based agents was at the cost of no longer finding all flags. We considered this an undesirable compromise and so will not cover this approach further.

Instead, in the following subsections we will discuss two approaches that lessened the gap by improving the performance of the individual-plan-based agents.

The first of these approaches is increasing exploration in the hope that the agents will experience and learn from policies that coordinate better than those encouraged by their individual plans. The second approach was to improve the individual plans by reducing the number of conflicts or increasing the time until conflict.

Both methods enjoy some success and provide useful insight in to how future solutions may overcome incorrect or conflicted knowledge. Where successful, these approaches provide solutions where multiple agents can be deployed without sharing their goals, broadcasting their actions or communicating to coordinate.

5.1 Increasing Exploration

Setting all initial Q-values to zero, as was mentioned in Section 4, is a pessimistic initialisation given that no negative rewards are received in this problem domain. Agents given pessimistic initial beliefs tend to explore less as any positive reward, however small, once received specifies the greedy policy and other policies will only be followed if randomly selected by the exploration steps [25].

With reward shaping and pessimistic initialisation an agent becomes more sensitive to the quality of knowledge they are shaped by. If encouraged to follow the optimal policy they can quickly learn to do so, as is the case in the initial study with the joint-plan-based agents. However, if encouraged to follow incorrect knowledge, such as the conflicted plans of the individual-plan-based agent, they may converge to a sub-optimal policy.

The opposing possibility is to instead initialise optimistically by setting all Q-values to start at the maximum possible reward. In this approach agents explore more as any action gaining less than the maximum reward becomes valued less than actions yet to be tried [25].

In Figure 5 we show the outcome of optimistically initialising the agents with Q-values of 600, the maximum reward agents can receive in this problem domain.

As would be expected, increased exploration causes the agents to take longer to learn a suitable policy. However, all agents (except for those receiving flag-based or combined-flag+joint-plan shaping) learn significantly better policies than their pessimistic equivalents[3]. This reduces the gap in final performance between all agents and the joint-plan-based agents, however, the difference that remains is still significant.

Despite that, the typical behaviour learnt by optimistic individual-plan-based agents is the same as the behaviour illustrated in Figure 3. However, it occurs less often in these agents than it occurred in the pessimistic joint-plan-based agents. This illustrates that conflicts can be overcome by optimistic initialisation but it cannot be guaranteed, by this

[3]For individual-plan-based agents $p = 0.064$, for all others $p < 0.05$.

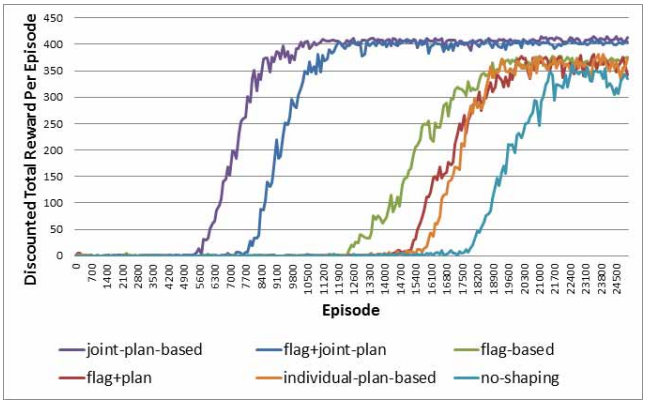


Figure 5: Optimistic Initialisation

method alone, that the optimal joint-plan will be learnt.

Furthermore, it takes time for the individual-plan-based agents to learn how to overcome the conflicts in their plans. However, this time is still less than it takes the agents with no prior knowledge to learn. Therefore, given optimistic initialisation, the benefit of reward shaping is now more important in the time to convergence instead of the final performance.

To conclude, these experiments demonstrate that some conflicted knowledge can be overcome given sufficient exploration.

5.2 Improving Knowledge

An alternative approach to overcoming conflicted knowledge would be to improve the knowledge. The results in this section illustrate that if the amount of the plan that can be followed increases then the time to convergence decreases (when optimistically initialised) or the final performance increases (when pessimistically initialised).

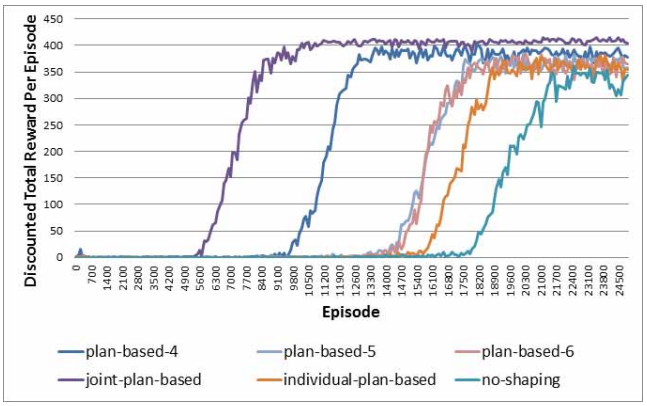


Figure 6: Optimistic Partial Plans

The individual-plan-based agents received shaping based on plans to both collect all six flags. If these plans are followed the agents will collide at their second planned flag to collect. The agent that does not pick up the flag will no longer be able to follow their plan and will therefore receive no further shaping rewards. Instead, we now consider three groups of agents that are shaped by less conflicted plans.

Specifically, plan-based-6 agents still both plan to collect all six flags, but the initial conflict is delayed until the second or third flag. The comparison of these agents to the individual-plan-based agents will show whether the timing of the conflict affects performance.

Plan-based-5 agents plan to collect just five flags each, reducing the number of conflicted flags to 4. Comparing this to both previous agents and subsequent agents will show whether the number of conflicts affects performance. These agents also experience their first conflict on the second or third flag.

Plan-based-4 agents plan to collect four flags each, reducing the number of conflicted flags to two and delaying the first conflict until the third flag. This agent will contribute to conclusions both on timing of conflicts and amount of.

As can be seen in Figure 6, both the timing of the conflict and the amount of conflict affect the agents' time to convergence. Little difference in final performance is evident in these results as the agents are still benefiting from optimistic initialisation.

If we return to pessimistic initialisation, as illustrated by Figure 7, reducing the amount of incorrect knowledge can also affect the final performance of the agents.

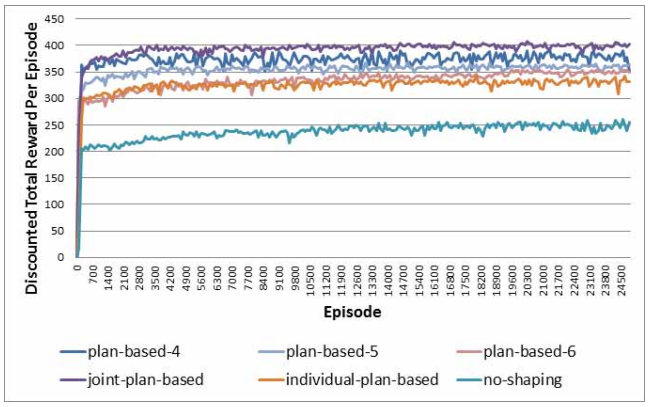


Figure 7: Pessimistic Partial Plans

However, to make plans with only partial overlaps, agents require some coordination or joint-knowledge that would not typically be available to multiple individual learners. If the process of improving knowledge could be automated, for instance with an agent starting an episode shaped by its individual plan and then refining the plan as it notices conflicts (i.e. plan steps that never occur), the agent may benefit from the improved knowledge and so alter its final performance without the need for optimistic initialisation.

6. SCALING UP

To further test multi-agent, plan-based reward shaping and our two approaches to handling incorrect knowledge we extended the problem domain by adding six extra flags[4](as illustrated in Figure 8.)

The results for pessimistic initialisation were effectively the same as those in the original domain except for a slightly longer time to convergence as would be expected due to the larger state space.

[4]Consequently $MaxReward$ now equals 1200.

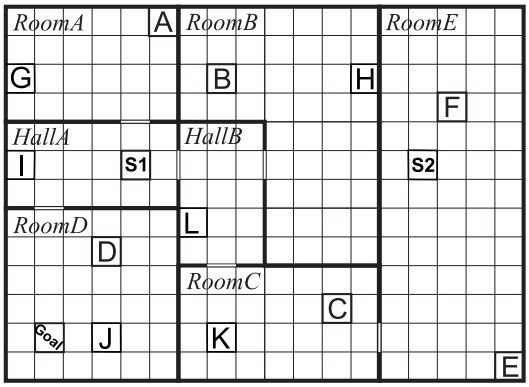


Figure 8: Scaled Up Problem Domain

The results for optimistic initialisation, however, took significantly longer. Figure 9 illustrates the results of just one complete run for this setting as performing any repeats would be impractical.

Whilst these results may be obtained quicker using function approximation or existing methods of improving optimistic exploration [13], they highlight the poor ability of optimistic initialisation to scale to large domains. Therefore, these experiments further support that automating the reduction of incorrect knowledge by an explicit belief revision mechanism would be more preferable than increasing exploration by optimistic initialisation as the latter method does not direct exploration sufficiently. Instead optimistic initialisation encourages exploration to all states randomly taking considerable time to complete. A gradual refining of the plan used to shape an agent would encourage initially a conflicted joint-policy, which is still better than no prior knowledge, and then on each update exploration would be directed towards a more coordinated joint-plan.

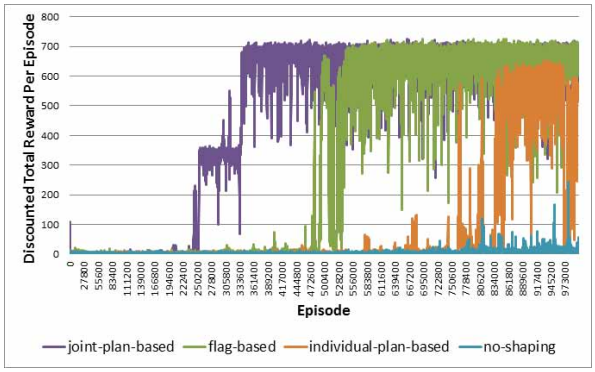


Figure 9: Scaling Up Optimistic Initialisation

7. CLOSING REMARKS AND FUTURE WORK

In conclusion, we have demonstrated two approaches to using plan-based reward shaping in multi-agent reinforcement learning. Ideally, plans are devised and coordinated centrally so each agent starts with prior knowledge of its own task allocation and the group can quickly converge to an optimal joint-policy.

Where this is not possible, due to agents unwilling to share information, plans made individually can shape the agent. Despite conflicts in the simultaneous execution of these plans, agents receiving individual-plan-based reward shaping outperformed those without any prior knowledge in all experiments.

Overcoming conflicts in the multiple individual plans by reinforcement learning can occur if shaping is combined with domain specific knowledge (i.e. flag-based reward shaping), the agent is initialised optimistically or the amount of conflicted knowledge is reduced. The first of these approaches requires a bespoke encoding of knowledge for any new problem domain and the second, optimistic initialisation, becomes impractical in larger domains.

Therefore, we are motivated to pursue in ongoing work the approach of automatically improving knowledge by an explicit belief revision mechanism. Where successful, this approach would provide a semi-automatic method of incorporating partial-knowledge in reinforcement learning agents that benefit from the correct knowledge provided and can overcome the conflicted knowledge.

8. REFERENCES

[1] J. Asmuth, M. Littman, and R. Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 604–609, 2008.

[2] M. Babes, E. de Cote, and M. Littman. Social reward shaping in the prisoner's dilemma. In *Proceedings of The Seventh Annual International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1389–1392, 2008.

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control (2 Vol Set)*. Athena Scientific, 3rd edition, 2007.

[4] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 746–752, 1998.

[5] Y. De Hauwere, P. Vrancx, and A. Nowé. Solving delayed coordination problems in mas (extended abstract). In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1115–1116. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[6] M. De Weerdt, A. Ter Mors, and C. Witteveen. Multi-agent planning: An introduction to planning and coordination. *Handouts of the European Agent Summer*, 2005.

[7] S. Devlin, M. Grześ, and D. Kudenko. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 2011.

[8] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of The Tenth Annual International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[9] S. Devlin and D. Kudenko. Dynamic potential-based reward shaping. In *Proceedings of The Eleventh Annual International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[10] M. Grześ. Improving exploration in reinforcement learning through domain knowledge and parameter analysis. Technical report, University of York, 2010.

[11] M. Grześ and D. Kudenko. Multigrid Reinforcement Learning with Reward Shaping. *Artificial Neural Networks-ICANN 2008*, pages 357–366, 2008.

[12] M. Grześ and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *Proceedings of the 4th IEEE International Conference on Intelligent Systems (IS'08)*, pages 22–29. IEEE, 2008.

[13] M. Grześ and D. Kudenko. Improving optimistic exploration in model-free reinforcement learning. *Adaptive and Natural Computing Algorithms*, pages 360–369, 2009.

[14] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.

[15] B. Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th International Conference on Machine learning*, page 608. ACM, 2007.

[16] J. Nash. Non-cooperative games. *Annals of mathematics*, 54(2):286–295, 1951.

[17] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, 1999.

[18] M. Peot and D. Smith. Conditional nonlinear planning. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, page 189. Morgan Kaufmann Pub, 1992.

[19] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., New York, NY, USA, 1994.

[20] J. Randløv and P. Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pages 463–471, 1998.

[21] J. Rosenschein. Synchronization of multi-agent plans. In *Proceedings of the National Conference on Artificial Intelligence*, pages 115–119, 1982.

[22] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[23] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[24] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, 1984.

[25] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[26] V. Ziparo. Multi-Agent Planning. Technical report, University of Rome, 2005.

Learning Teammate Models for Ad Hoc Teamwork

Samuel Barrett
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
sbarrett@cs.utexas.edu

Peter Stone
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
pstone@cs.utexas.edu

Sarit Kraus
Dept. of Computer Science
Bar-Ilan University
Ramat Gan, 52900 Israel
sarit@cs.biu.ac.il

Avi Rosenfeld
Jerusalem College of Technology
Jerusalem, 91160 Israel
rosenfa@jct.ac.il

ABSTRACT

Robust autonomous agents should be able to cooperate with new teammates effectively by employing *ad hoc teamwork*. Reasoning about ad hoc teamwork allows agents to perform joint tasks while cooperating with a variety of teammates. As the teammates may not share a communication or coordination algorithm, the ad hoc team agent adapts to its teammates just by observing them. Whereas most past work on ad hoc teamwork considers the case where the ad hoc team agent has a prior model of its teammate, this paper is the first to introduce an agent that learns models of its teammates autonomously. In addition, this paper presents a new transfer learning algorithm that can be used when the ad hoc agent only has limited observations about potential teammates.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligience**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Experimentation

Keywords

Ad Hoc Teams, Multiagent Systems, Teamwork

1. INTRODUCTION

For autonomous agents to perform effectively in society, they should be able cooperate with other agents. One way for this to happen is for all autonomous agents to share a communication protocol or a coordination algorithm. However, agents may be developed by many sources, making it difficult to ensure that all the agents share the same information. In addition, if agents stay deployed for a long time, it is likely that new agents will come along with new and different protocols. Therefore, it is important that agents be capable of adapting to previously unseen teammates to cooperate in accomplishing their tasks.

For example, after a disaster, robots developed in many different labs and companies may be deployed to search the area and rescue victims. As these robots come from multiple sources, they may not share a coordination protocol. If there was no need for immediate action, the developers could program methods for communication or split up the task into parts by hand. If instead some of the robots are designed to cooperate with ad hoc teams, they will be able to quickly adapt to their teammates and swiftly map the area and rescue victims.

Stone et al. [17] introduced the *ad hoc team setting* as a problem in which team coordination strategies cannot be specified a priori. They presented an algorithm for evaluating ad hoc team agents based on their ability to cooperate with a set of teammates to accomplish a set of possible tasks. They argue that while previous research has focused on theoretical results, the ad hoc teamwork problem is "ultimately an empirical challenge."

This paper addresses this challenge by introducing an ad hoc agent that explicitly builds models of its teammates and plans its behavior using a sample-based method. Then, the paper evaluates the ad hoc agent's ability to cooperate with a wide variety of unknown teammates, showing that ad hoc agents can learn models for their teammates and evaluate which models are helpful, despite their current teammates differing qualitatively from the observed ones. Finally, this paper introduces a new method for transfer learning that considers data coming from multiple sources and then evaluates its effectiveness for ad hoc teams when the ad hoc agent has a small number of observations of the current agent.

2. PROBLEM DESCRIPTION

Consider the case where an ad hoc agent is trying to cooperate with a set of teammates it has never seen before. If the ad hoc agent and its teammates share a communication protocol or a method for coordination, it can directly cooperate with them. However, if its communication is limited and there is no pre-arranged coordination method, the ad hoc agent must observe its teammates and try to adapt to their behaviors. If the ad hoc agent has previously observed agents similar to its current teammates, it should try to leverage its prior experiences in order to cooperate with them more effectively.

To clarify the problem, consider the concrete example of

a team of robots trying to surround and disable an intruder. The original team of robots was designed to coordinate their actions to capture the intruder quickly and reliably, but one of the original robots has since broken and the others have been damaged from constant wear and tear. Therefore, a new robot is deployed to join the team, but it does not know the specific make and model of its teammates. Fortunately, the new robot has observed similar teammates in other situations, so it has some understanding of how to cooperate with its new teammates. The problem it faces is to quickly identify which previous observations are applicable to these teammates and adapt to any differences.

2.1 Pursuit Domain

The pursuit domain is a popular problem in multiagent systems literature because it requires all of the teammates to cooperate to capture the prey [18]. The details vary, but the pursuit domain revolves around a set of agents called predators trying to capture an agent called the prey in minimal time.

In the version of the pursuit domain used in this paper, the world is a rectangular, toroidal grid, where moving off one side of the grid brings the agent back on the opposite side. Four predators attempt to capture the randomly moving prey by surrounding it on all sides in as few time steps as possible. At each time step, each agent can select to move in any of the four cardinal directions or to remain in its current position. All agents pick their actions simultaneously, and collisions are handled using priorities that are randomized at each time step. In addition, each agent is able to observe the positions of all other agents. A view of the domain is shown in Figure 1.

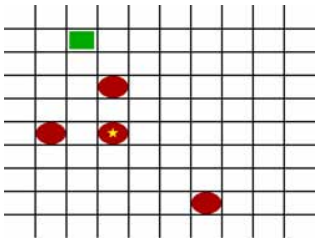


Figure 1: A view of the pursuit domain, where the rectangle is the prey, the ovals are predators, and the oval with the star is the ad hoc predator being evaluated.

2.2 Evaluation

Ad hoc team agents must be able to cooperate with a variety of previously unseen teammates to accomplish a task. To this end, we adopt the evaluation framework proposed by Stone et al. [17]. This evaluation specifies that the performance of an ad hoc agent explicitly depends on the possible teammates it will encounter as well as the potential tasks it may face. The evaluation proceeds by sampling a task and a team from the set of possible tasks and teammates. Then, it removes one of the agents from the team selected at random, replaces it with the ad hoc agent, and observes the performance of the newly created team. The overall performance of the ad hoc agent is averaged over a number of these samples.

3. METHODS

In order to cooperate effectively with its teammates, the ad hoc agent chooses its actions by planning about their long term effects. To do so, the ad hoc agent must have a model of the domain, the prey, and its teammates. This paper assumes that the ad hoc agent has a correct model of the domain and the prey, but must determine how to model its teammates.

Even if the ad hoc agent has a perfect model of its teammates, the planning problem is still difficult. With four predators and a single prey, the pursuit domain has a branching factor of $5^5 = 3125$ actions and, in a 20x20 world, there are $(20 * 20)^5 \approx 10^{13}$ different states. While it is possible to reduce the size of the state space using some symmetries of the world, planning efficiently is important in the pursuit domain.

3.1 UCT

To plan efficiently, our ad hoc agent uses UCT [13], a Monte Carlo Tree Search (MCTS) algorithm that employs upper confidence bounds for controlling the tradeoff between exploration and exploitation. Previous work has shown that UCT performs well on domains with high branching factors, such as Go [8] and large POMDPs [16]. In addition, previous work has shown that UCT can both compete with optimal planners on small pursuit problems and scale to larger problems [2].

At each time step, the ad hoc agent performs a number of rollouts, where a rollout is the simulation of an episode starting from the current world state and ending when the prey is captured. Then, the ad hoc agent uses the time it takes to capture the prey in each simulation to evaluate the actions it selected in the rollout. It selects its actions by choosing the one with the highest upper confidence bound, causing it to explore when it is unsure of the best action and exploit its knowledge when it is confident of the results.

3.2 Model Selection

Performing the simulations for the UCT rollouts requires that the ad hoc agent has a model for how its teammates behave. If there is a (presumably correct or approximately correct) single model for this behavior, the planning is straightforward. On the other hand, the problem is more difficult if the ad hoc agent is given several possible models. Assuming that the ad hoc agent starts with some prior belief distribution over which model correctly reflects its teammates' behaviors, the ad hoc agent can update these beliefs by observing its teammates. Specifically, it can update the models using Bayes theorem:

$$P(\text{model}|\text{actions}) = \frac{P(\text{actions}|\text{model}) * P(\text{model})}{P(\text{actions})}$$

If the correct model is in the given set of models, then the ad hoc agent's beliefs will converge to this model.

On the other hand, if the correct model is not in the set, using Bayes rule may drop their posterior probability to 0 for a single wrong prediction. This may punish generally well-performing models that make one mistake, while leaving poor models that predict nearly randomly. Therefore, it may be advantageous to update the probabilities more conservatively. Research in regret minimization has shown that updating model probabilities using the polynomial weights algorithm is near optimal if examples are chosen adversari-

ally [3]. Since it is expected that the ad hoc agent's models are not perfect, the agent updates its beliefs using polynomial weights:

$$\begin{aligned} \text{loss} &= 1 - P(\text{actions}|\text{model}) \\ P(\text{model}|\text{actions}) &\propto (1 - \eta * \text{loss}) * P(\text{model}) \end{aligned}$$

where $\eta = 0.5$. This scheme ensures that good models are not prematurely removed, but it does reduce the rate of convergence. We find that in practice it performs very well as the observed examples of the teammates may be arbitrarily unrepresentative of the agent's overall decision function.

Given the current belief distribution over the models, using this information, the ad hoc agent can sample teammate models for planning, choosing one model for each rollout similarly to the approach adopted by Silver and Veness [16]. Sampling the model once per rollout is desirable compared to sampling a model at each time step because this resampling can lead to states that no model predicts. Ideally, a different state-action evaluation would be stored for each model, but that would require many more rollouts to plan effectively. Instead, the state-action evaluations from all the models are combined to improve the generalization of the planning.

3.3 Learning Models

The previous sections described how the ad hoc agent can select the correct model and use it for planning, but they did not specify where these models come from in the first place. One option is that the ad hoc agent is given a model that was coded by hand, but another interesting option is when the ad hoc agent learns the model itself. If the ad hoc agent can observe representative potential teammates before interacting with them, it can build a model for their behaviors. To build these models, the ad hoc agent uses the features given in Table 1, where all positions are relative to the modeled predator. These features were chosen experimentally using cross validation on the training set. The ad hoc agent considers the previous actions of the teammate because a few of potential teammates appeared to keep an internal state that changed infrequently. In this case, a model is a mapping from these features to a next action, and a training instance represents only a single predator's actions: a trajectory of length k provides $4k$ training instances.

Description	# Features	Values
Predator Number	1	$\{0, 1, 2, 3\}$
Prey x position	1	$\{-10, \ldots, 10\}$
Prey y position	1	$\{-10, \ldots, 10\}$
Predator$_i$ x position	3	$\{-10, \ldots, 10\}$
Predator$_i$ y position	3	$\{-10, \ldots, 10\}$
Neighboring prey	1	{true,false}
Cell neighboring prey is occupied	4	{true,false}
Previous two actions	2	$\{\leftarrow, \rightarrow, \uparrow, \downarrow, \bullet\}$

Table 1: Features for predicting a teammate's actions.

In some cases, rather than having extensive observations of its teammates for learning models, the ad hoc agent will only have a small number of observations of its current teammates. It could still build a model from this data, but it may be able to improve the accuracy of the model by using information it has about similar teammates through transfer learning (TL). Following standard TL terminology, we consider the current teammates to be the *target* teammates: the goal is to improve performance when teamed with these agents. We call the previously observed teammates the *source* teammates, which can provide knowledge to transfer to modeling the target teammates. To perform this transfer, we introduce the TwoStageTransfer algorithm.

TwoStageTransfer was inspired by the TwoStageTrAdaBoost algorithm created by Pardoe and Stone [15]. TwoStageTrAdaBoost is an algorithm for transfer learning that uses the source data directly in the target task rather than transferring derived classifiers. It searches for the optimal weighting of the source data using n-fold cross validation with the target data. TwoStageTrAdaBoost focuses on transfer for regression rather than classification and treats all source data as coming from the same task. In contrast, TwoStageTransfer tackles the classification problem and uses the information that source data may come from different sources. In this case, the ad hoc agent has observed many other agents, some of which are more similar to the target teammate than others. Therefore, tracking the source of the data may be important as it allows the ad hoc agent to discount data coming from agents that are very different from it. Recent research into transfer learning has shown that such information may improve results [20, 11].

TwoStageTransfer's goal is to find the best possible weighting of each set of source data and create a classifier using these weights. The full algorithm is described in Algorithm 1. TwoStageTransfer takes in the target data set T, the set of source data sets $\mathbb{S} = \{S_1, \ldots, S_n\}$, a number of boosting iterations m, a number of folds k for cross validation, and a maximum number of source data sets to include b. We use the annotation S^w to mean the data set S taken with weight w spread over the instances. The base model learner used in this case is a decision tree learner based on C4.5 trees that handles weighted instances.

Ideally, TwoStageTransfer would try every combination of weightings, but this proves to be computationally expensive as transferring from n source data sets and considering m different weight levels leads to m^n possible combinations. In our case, we have $n = 28$ data sets and $m = 10$ weightings (as discussed in Section 4), which leads to 10^{28} combinations. These data sets are discussed in more depth in Section 4.3. Rather than try all of them, TwoStageTransfer first evaluates each data source independently, as if it were the only source data set, and calculates the ideal weight of that data source as specified in Algorithm 1. Then, it adds the data sources in decreasing order of the calculated weights. As it adds each data set, it finds the optimal weighting of that set with the data that has already been added. Finally, it adds the data with the optimal weight and repeats the procedure with the next data set. Note that this algorithm requires only $nm + nm = 2nm$ combinations to be evaluated, nm for the initial evaluations and then m when adding each n data sets. In this case, it requires only $2 * 10 * 28 = 560$ combinations to be evaluated. However, this greedy approach may not find the optimal weights. In addition, it is possible to limit the number of data sets used in the final transfer, which is desirable in our setting as training with all 50,000 instances from each of the 28 available source data sets (see Section 4) of the source data sets can exceed 4 GB of RAM usage.

Algorithm 1 Transfer learning with multiple sources

TwoStageTransfer $(T, \mathbb{S}, m, k, b)$
  for all S_i in $\mathbb{S}$: **do**
    $w_i \leftarrow$ CalculateOptimalWeight$(T, \emptyset, S_i, m, k)$
  end for
  Sort $\mathbb{S}$ in decreasing order of w_i's
  $F \leftarrow \emptyset$
  for i from 1 to b **do**
    $w \leftarrow$ CalculateOptimalWeight(T, F, S_i, m, k)
    $F \leftarrow F \cup S_i^w$
  end for
  Train classifier c on $T \cup F$
  return c
end
CalculateOptimalWeight(T, F, S, m, k):
  for i from 1 to m **do**
    $w_i = \frac{|T|}{|T|+|S|}(1 - \frac{i}{m-1})$
    Use k-fold cross validation on T
    Calculate err_i from k-fold cross validation on T using F and S^{w_i} as additional training data
  end for
  return w_j such that $j = \underset{i}{\text{argmax}}(\text{err}_i)$
end

4. RESULTS

This section evaluates a number of ad hoc agents that vary in the type and amount of information they have about their teammates. If it has access to the true model, the ad hoc agent can plan to cooperate with its teammates optimally, as approximated by UCT(True), or use the same behavior as the missing teammate (Match(True)). However, in the fully general ad hoc teamwork scenario, such a model is not generally available. Thus, this paper instead focuses on the case in which the ad hoc agent must learn a model of its teammates by observing them. In this section, the amount of available information available to the ad hoc agent varies, ranging from observing its current teammates to only observing teammates that may differ greatly from the current ones. Additional results focus on the case in which the ad hoc agent has a small number of observations of the current teammates and must transfer information learned from prior observations of similar teammates.

To properly evaluate an ad hoc team agent, it is important to test it with a variety of possible teammates. To prevent any bias in the development of these teammates, we collected 31 agents created by undergraduate and graduate computer science students. These agents were created for an assignment in a workshop on agent design with no discussion of ad hoc teams. The students were asked to create a team of predators that captured the prey as quickly as possible. The agents produced varied wildly in their approaches as well as their effectiveness. While almost all of the agents performed well, two agents were removed due to their low performance, leaving 29 agents. In Sections 4.1 and 4.3, the teammates come from this set of 29 students, but for Section 4.2, the teammates come from the set of 12 students used by Barrett et al. [2] to prevent any bias in the selection of the unknown teammates.

For these evaluations, a random team is selected, a single agent is replaced by the ad hoc agent, and the team is evaluated based on its time to capture the prey. Throughout this paper, we refer to a teammate type as its behavior function, meaning that agents coming from different students have different types. To simulate wear and tear on the teammates, agents of the same type may further vary based on their speed. This speed is controlled by giving each agent a random chance to stay still rather than taking their desired action, which can be thought of as the gears on a robot slipping. The chance of staying still is randomly sampled independently for each agent from $[0, 0.2]$, but in training all teammates are observed with a 0.1 probability of staying still. Results are averaged over 1,000 episodes where a random student's team is selected for each episode. The random teams, probabilities of agents staying still, and starting positions are fixed across the ad hoc agents to allow for paired statistical analysis, and all statistical tests are performed as paired Student-T tests with $p = 0.05$.

When the ad hoc agent observes its potential teammates, it watches a team of four predators for 50,000 steps, resulting in a total of 200,000 training instances. These predators have a 0.1 probability of staying still. With this information, the ad hoc agent learns a decision tree using a learning algorithm based on C4.5 trees that handles weighted instances. Several other classifiers were tried including SVMs, naive Bayes, and decision lists as well as boosted versions of these classifiers, but decision trees tended to outperform these methods in a combination of prediction accuracy and training time. All model learning is performed offline, but the ad hoc agent updates its belief over the models online.

The behaviors tested are listed below, with ad hoc agents planning with the learned models compared to agents given either the true model or a set of representative, hand-coded models:

- **Match(True)**: Match teammates' behavior. The ad hoc agent knows the true model of its teammates, and behaves as the agent it is replacing would.
- **UCT(True)**: Plan with the true model. The ad hoc agent knows the true model of its teammates and plans with this model.
- **UCT(HC)**: Plan with hand-coded models. The ad hoc agent is given a set of hand-coded models to plan with.
- **UCT($\mathbf{DT_{cor}}$)**: Plan with correct decision tree. The ad hoc agent knows the type of teammates it is cooperating with and has observed this type before.
- **UCT($\mathbf{DT_{all}}$)**: Plan with all decision trees. The ad hoc agent does not know the type of teammates it is cooperating with, but it has observed several types of possible teammates, including the current teammate type.
- **UCT($\mathbf{DT_{oth}}$)**: Plan with other decision trees. The ad hoc agent does not know the type of teammates it is cooperating with, and it has observed several types of possible teammates, but *not* the current teammate type.
- **UCT($\mathbf{DT_{tra}}$)**: Plan with a decision tree created using transfer learning. The ad hoc agent knows the type of teammates it is cooperating with and has briefly observed this type, but it has also observed several other types of teammates for longer periods of time.

For the DT_{cor}, DT_{all}, and DT_{oth} models, the ad hoc agent builds a separate decision tree for each of the 29 types of teammates it has observed. In the UCT(DT_{cor}) behavior,

the ad hoc agent knows the type of its current teammate and only uses its model of that type for planning, while in the UCT(DT_{all}) and UCT(DT_{oth}) behaviors it uses all available models. However, in the UCT(DT_{oth}) behavior, the true teammates are not drawn from the set of known teammate types. Finally, to create the DT_{tra} models, the ad hoc agent observes the full 50,000 steps of 28 teammate types, but only 1,000 steps of the type it is currently cooperating with. Therefore, it uses transfer learning in the form of TwoStageTransfer to reuse knowledge learned from the other 28 teammate types to create the DT_{tra} models, using the same decision tree learner as the base learner.

This section compares the performance of ad hoc agents to the unrealistic behaviors of Match(True) and UCT(True) and the previous best result of UCT(HC), leading to three main results. As described in Section 4.1, the first result is that learning models from known teammate types outperforms previous techniques. Then, Section 4.2 presents the second result, showing that these learned models still perform well for teammates of previously unseen types. Finally, Section 4.3 shows that transfer learning can improve the performance of the ad hoc agent if only a small amount of data on the current teammates is available.

4.1 Known Teammate Types

This section compares the performance of ad hoc agents that learn models of their teammates to agents that have access to true models of their teammates. Given the true behavior model of its teammates, the ad hoc agent can either behave as the missing teammate would (Match(True)), or it can plan using this true model (UCT(True)). If the teammates are optimal, Match(True) is also optimal, but regardless of the teammates' behaviors, UCT(True) should be close to optimal, with any loss caused by the approximations of the planning algorithm. Therefore, these two behaviors serve as baselines and UCT(HC) represents the current state of the art behavior [2]. The hand-coded models given to the ad hoc agent are chosen to be representative of the space of behaviors in the pursuit domain, but they may be only coarse approximations of the true teammate behavior. In addition, they may require a significant amount of time and effort on the behalf of the designer to create.

Compared to these behaviors, we introduce two new behaviors in which the ad hoc agent learns a model of its teammates. Both create models for each of the 29 types of agents it has observed, but the UCT(DT_{cor}) agent also knows the type of its current teammates. More reflective of the fully general ad hoc teamwork scenario is the UCT(DT_{all}) behavior, whereby the ad hoc agent does not know the type of its current teammates and must track its beliefs over the learned models by observing its teammates.

Figure 2 shows how an ad hoc agent planning with the learned models compares to planning with the true model or a set of hand-coded models. Unsurprisingly, the ad hoc agent performs best when planning using the True model, as it should be optimal given an optimal planning algorithm. Though conventional wisdom suggests that matching the teammates behavior would be fairly effective, this performs poorly as shown by the Match(True) bar because the teammates may be arbitrarily far from optimal. The difference between UCT(DT_{cor}) and UCT(HC) is statistically significant as is the difference between UCT(HC) and Match(True). The differences between UCT(True) and

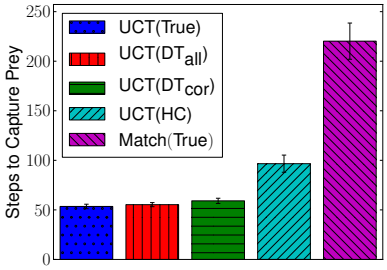


Figure 2: Interacting with observed student teammates

UCT(DT_{all}) as well as the differences between UCT(DT_{all}) and UCT(DT_{cor}) are not statistically significant.

If the ad hoc agent plans with the DT_{cor} or DT_{all} models, it outperforms an ad hoc agent that plans using the HC models, showing that learning a model of the teammates is effective. In addition, if the ad hoc agent plans with the DT_{cor} models, its performance is very close to the gold standard of planning with True models. However, using the DT_{all} models allows the ad hoc agent to outperform an agent planning with the DT_{cor} models, which is surprising given selecting from the set of DT_{all} models should increase the difficulty of the problem. We hypothesize that this difference occurs from imperfections in learning the models. Some student agents are fairly stochastic and it is possible and likely that the learned decision trees overfit this data. By using a set of models of all the agents, the ad hoc agent plans about a broader range of possible teammates and better accounts for this randomness.

4.2 Unknown Teammate Types

While the previous section focused on the case where the ad hoc agent encounters teammates of a type it has previously observed, the ad hoc agent may not always be that lucky. Instead, it may have to cooperate with teammates that are fairly different from any it has seen before. The ad hoc agent therefore plans with its 29 learned models, but encounters a 30^{th} type of teammate drawn from a set of 12 student agents described in the work by Barrett et al. [2].

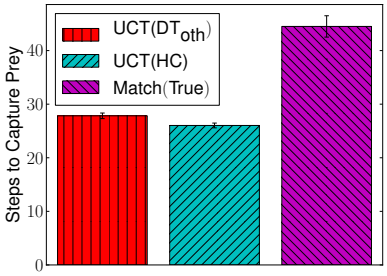


Figure 3: Interacting with the 12 unobserved student teammates from Barrett et al.

Figure 3 shows that having the ad hoc agent plan using the HC models works very well for these teammates. Planning using the DT_{oth} models performs slightly worse than the HC models due to differences between the learned models and the encountered teammates. The performance difference is statistically significant. However, the difference is small, and may be reduced by using a larger, more varied

set of training agents. On the other hand, planning using the DT_{oth} models still far outperforms the Match(True) behavior, showing that the DT_{oth} models are still effective for planning.

4.3 Teammates with Limited Observations

The previous section discussed the case in which the ad hoc agent had no previous observations of its current teammates. However, this is the worst case scenario; in many cases, the ad hoc agent may build up a small number of observations of the current teammate model. Then, it can use transfer learning to reuse its observations of other agents to improve the performance of this model. In this case, the ad hoc agent has 50,000 training steps of each of the other 28 teammate types, but only 1,000 training steps of the current type of teammate. The teammates used for these tests are the same as those in Section 4.1, specifically, the set 29 teammates generated by the students.

The model learned using transfer learning is compared to the upper limit of UCT(DT_{all}) where the ad hoc agent has 50,000 training instances of the current type of teammate, though observed with a different amount of noise. The lower limit is given by only using the 1,000 training steps of the current teammate type and not performing any transfer (DT_{target}). Figure 4 shows that using TwoStageTransfer to perform transfer learning is helpful in this case. However, analysis of the UCT(DT_{tra}) results shows that there are a small number of very long episodes caused by inaccuracies in the learned models, but this happens less frequently when planning with the DT_{target} models. Therefore, it is desirable for the ad hoc agent to use both models, which outperforms using either one individually as shown by the UCT(DT_{tra} + DT_{target})bar. For all uses of TwoStageTransfer, a value of $m = 10$ was experically determined to be most effective. DT_{tra} is built using TwoStageTransfer where $b = 5$, meaning that the ad hoc agent is transferring information from the five most helpful source agents. Five was chosen as it provides a good tradeoff between performance and training time. On the other hand, the ad hoc agent could merely take the one step greedy approach by performing TwoStageTransfer where $b = 1$. If it also plans using the model learned from the target data, the ad hoc agent's performance is shown in DT_{tra*} + DT_{target}. The differences from the UCT(DT_{target}) behavior to each other behavior is significant, but the differences among the behaviors planning with the other learned models are not.

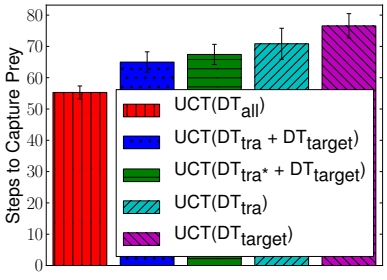


Figure 4: Interacting with partially observed student teammates

In the current setup, the ad hoc agent is given observations of 1,000 steps of the target teammate performing the task. However, the amount of target data may vary, and it is important to understand how the performance of the transfer learning algorithms react to this variable. Figure 5 shows how the performance of the agent drops off given decreasing amounts of data about the current teammate type. In the tests where only 10 and 100 steps of the target agent were observed, episodes did not complete with the capture of the prey within 10,000 steps in 13 and 12 episodes respectively out of the 1,000 episodes. These episodes are excluded from the graph as it is unclear what value they should take, but it is clear that they indicate poor performance in these settings.

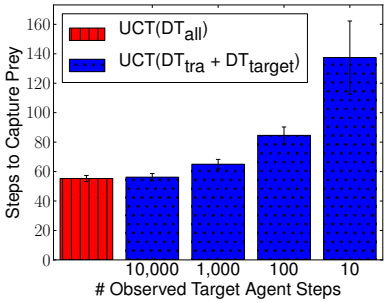


Figure 5: Varying the number of observations of the current teammate type.

5. RELATED WORK

This formulation of the ad hoc teamwork problem and the evaluation framework were proposed by Stone et al. [17]. Barrett and Stone [1] analyze current research on ad hoc teams and present several dimensions for describing ad hoc team problems. One line of early research on ad hoc teams involves an agent attempting to teach a novice agent while performing a repeated joint task [5]. Other research includes Jones et al.'s [12] research on pickup teams cooperating to accomplish a treasure hunt. In the area of robot soccer, Liemhetcharat and Veloso [14] explore ad hoc teamwork for role-based teams, and Bowling and McCraken [4] consider the case where the ad hoc agent is given a different playbook from its teammates. Further work into ad hoc teams using stage games and biased adaptive play was performed by Wu et al. [19], while Han et al. [10] explore how a single agent can affect the collective behavior of a large, multi-agent system.

The problem of opponent modeling is closely related to the problem of ad hoc teams. Where opponent modeling focuses on modeling opponents and considers the worst case scenarios for their actions, ad hoc teamwork instead focuses on cooperating with teammates and can make stronger assumptions about their actions. Important work on opponent modeling is regularly presented at the Workshop on Plan, Activity, and Intent Recognition (PAIR) as well as the Workshop on Applied Adversarial Reasoning and Risk Modeling (AARM). One interesting approach is the AWESOME algorithm [6] which achieves convergence and rationality in repeated games. Another approach is to explicitly model and reason about other agents' beliefs such as the work on I-POMDPs [9] and I-DIDs [7]. However, modeling other agents' beliefs greatly expands the space for planning, and these approaches do not currently scale to larger problems.

6. CONCLUSION

Most existing research on ad hoc teamwork focuses on the case in which the ad hoc agent has access to a correct model of its teammates. This paper is the first to have the ad hoc agent learn models of its teammates autonomously. Planning with these learned models allows the ad hoc agent to cooperate with its teammates effectively even when its teammates were not observed during the model learning. This paper also presents a transfer learning algorithm, TwoStage-Transfer, that can significantly improve results when the ad hoc agent has a limited number of observations of its teammates.

While this paper answers questions about the variety of teammates that ad hoc team agents can effectively cooperate with, it also raises new questions that should be explored in future research. One possible research avenue is into communication between ad hoc teams. Wireless connectivity is becoming more and more common, and approaches such as the Robot Operating System (ROS) are standardizing communication protocols. Therefore, it is likely that ad hoc team agents may be able to communicate with their teammates, although this communication may be limited by what their teammates were developed to understand. All results in this paper are reported the pursuit domain, but future research should test whether similar algorithms perform well on other domains. In addition, this work focuses on agents that follow mostly fixed behaviors, with little adaptation to the ad hoc agent's behaviors; handling adaptive teammates is a complicated, but exciting area for future research.

7. REFERENCES

[1] S. Barrett and P. Stone. An analysis framework for ad hoc teamwork tasks. In *AAMAS '12*, June 2012.

[2] S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *AAMAS '11*, May 2011.

[3] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. 2007.

[4] M. Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *AAAI*, pages 53–58, 2005.

[5] R. I. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. *JAIR*, 4:477–507, 1996.

[6] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67, May 2007.

[7] P. Doshi and Y. Zeng. Improved approximation of interactive dynamic influence diagrams using discriminative model updates. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 907–914, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[8] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS '06*, December 2006.

[9] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *JAIR*, 24(1):49–79, July 2005.

[10] J. Han, M. Li, and L. Guo. Soft control on collective behavior of a group of autonomous agents by a shill agent. *Journal of Systems Science and Complexity*, 19:54–62, 2006.

[11] P. Huang, G. Wang, and S. Qin. Boosting for transfer learning from multiple data sources. *Pattern Recognition Letters*, 33(5):568 – 579, 2012.

[12] E. Jones, B. Browning, M. B. Dias, B. Argall, M. M. Veloso, and A. T. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *ICRA*, pages 570 – 575, May 2006.

[13] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *ECML '06*, 2006.

[14] S. Liemhetcharat and M. Veloso. Modeling mutual capabilities in heterogeneous teams for role assignment. In *IROS '11*, pages 3638 –3644, 2011.

[15] D. Pardoe and P. Stone. Boosting for regression transfer. In *ICML '10*, June 2010.

[16] D. Silver and J. Veness. Monte-carlo planning in large pomdps. In *NIPS '10*. 2010.

[17] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI '10*, July 2010.

[18] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[19] F. Wu, S. Zilberstein, and X. Chen. Online planning for ad hoc autonomous agent teams. In *IJCAI*, 2011.

[20] Y. Yao and G. Doretto. Boosting for transfer learning with multiple sources. In *CVPR '10*, June 2010.

Overcoming Incorrect Knowledge in Plan-Based Reward Shaping

Kyriakos Efthymiadis
Department of Computer Science,
University of York, UK
kirk@cs.york.ac.uk

Sam Devlin
Department of Computer Science,
University of York, UK
devlin@cs.york.ac.uk

Daniel Kudenko
Department of Computer Science,
University of York, UK
kudenko@cs.york.ac.uk

ABSTRACT

Reward shaping has been shown to significantly improve an agent's performance in reinforcement learning. Plan-based reward shaping is a successful approach in which a STRIPS plan is used in order to guide the agent to the optimal behaviour. However, if the provided knowledge is wrong, it has been shown the agent will take longer to learn the optimal policy. Previously, in some cases, it was better to ignore all prior knowledge despite it only being partially incorrect.

This paper introduces a novel use of knowledge revision to overcome incorrect domain knowledge when provided to an agent receiving plan-based reward shaping. Empirical results show that an agent using this method can outperform the previous agent receiving plan-based reward shaping without knowledge revision.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Experimentation

Keywords

Reinforcement Learning, Reward Shaping, Knowledge Revision

1. INTRODUCTION

Reinforcement learning (RL) has proven to be a successful technique when an agent needs to act and improve in a given environment. The agent receives feedback about its behaviour in terms of rewards through constant interaction with the environment. Traditional reinforcement learning assumes the agent has no prior knowledge about the environment it is acting on. Nevertheless, in many cases (potentially abstract and heuristic) domain knowledge of the RL tasks is available, and can be used to improve the learning performance.

In earlier work on *knowledge-based reinforcement learning* (KBRL) [8, 3] it was demonstrated that the incorporation of domain knowledge in RL via reward shaping can significantly improve the speed of converging to an optimal policy. Reward shaping is the process of providing prior knowledge to an agent through additional rewards. These rewards help direct an agent's exploration, minimising the number of suboptimal steps it takes and so directing it towards the optimal policy quicker. Plan-based reward shaping [8] is a particular instance of knowledge-based RL where the agent is provided with a high level STRIPS plan which is used in order to guide the agent to the desired behaviour.

However, problems arise when the provided knowledge is partially incorrect or incomplete, which can happen frequently given that expert domain knowledge is often of a heuristic nature. For example, it has been shown in [8] that if the provided plan is flawed then the agent's learning performance drops and in some cases is worse than not using domain knowledge at all.

This paper presents, for the first time, an approach in which agents use their experience to revise incorrect knowledge whilst learning and continue to use the then corrected knowledge to guide the RL process. Figure 1 illustrates the interaction between the knowledge base and the RL level, where the contribution of this work is the knowledge revision.

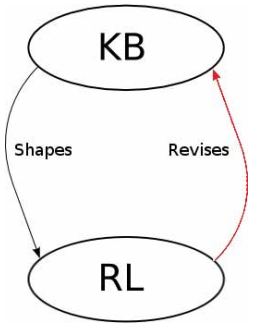


Figure 1: Knowledge-Based Reinforcement Learning.

We demonstrate, in this paper, that adding knowledge revision to plan-based reward shaping can improve an agent's performance (compared to a plan-based agent without knowledge revision) when both agents are provided with incorrect knowledge.

2. BACKGROUND

2.1 Reinforcement Learning

Reinforcement learning is a method where an agent learns by receiving rewards or punishments through continuous interaction with the environment [13]. The agent receives a numeric feedback relative to its actions and in time learns how to optimise its action choices. Typically reinforcement

learning uses a Markov Decision Process (MDP) as a mathematical model [11].

An MDP is a tuple $\langle S, A, T, R \rangle$, where S is the state space, A is the action space, $T(s, a, s') = Pr(s'|s, a)$ is the probability that action a in state s will lead to state s', and $R(s, a, s')$ is the immediate reward r received when action a taken in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and reward function) are available, this task can be solved using dynamic programming [2].

When the environment dynamics are not available, as with most real problem domains, dynamic programming cannot be used. However, the concept of an iterative approach remains the backbone of the majority of reinforcement learning algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states, $V(s)$, or state-action pairs, $Q(s, a)$. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. The SARSA algorithm is such a method [13]. After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \quad (1)$$

where α is the rate of learning and γ is the discount factor. It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'.

It is important whilst learning in an environment to balance exploration of new state-action pairs with exploitation of those which are already known to receive high rewards. A common method of doing so is $\epsilon-$greedy exploration. When using this method the agent explores, with probability ϵ, by choosing a random action or exploits its current knowledge, with probability $1 - \epsilon$, by choosing the highest value action for the current state [13].

Temporal-difference algorithms, such as SARSA, only update the single latest state-action pair. In environments where rewards are sparse, many episodes may be required for the true value of a policy to propagate sufficiently. To speed up this process, a method known as eligibility traces keeps a record of previous state-action pairs that have occurred and are therefore eligible for update when a reward is received. The eligibility of the latest state-action pair is set to 1 and all other state-action pairs' eligibility is multiplied by λ (where $\lambda \leq 1$). When an action is completed all state-action pairs are updated by the temporal difference multiplied by their eligibility and so Q-values propagate quicker [13].

Typically, reinforcement learning agents are deployed with no prior knowledge. The assumption is that the developer has no knowledge of how the agent(s) should behave. However, more often than not, this is not the case. As a group we are interested in *knowledge-based reinforcement learning*, an area where this assumption is removed and informed agents can benefit from prior knowledge.

2.2 Reward Shaping

One common method of imparting knowledge to a reinforcement learning agent is reward shaping. In this approach, an additional reward representative of prior knowledge is given to the agent to reduce the number of suboptimal actions made and so reduce the time needed to learn [10, 12]. This concept can be represented by the following formula for the SARSA algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, s') + \gamma Q(s', a') - Q(s, a)] \quad (2)$$

where $F(s, s')$ is the general form of any state-based shaping reward.

Even though reward shaping has been powerful in many experiments it quickly became apparent that, when used improperly, it can change the optimal policy [12]. To deal with such problems, potential-based reward shaping was proposed [10] as the difference of some potential function Φ defined over a source s and a destination state s':

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \quad (3)$$

where γ must be the same discount factor as used in the agent's update rule (see Equation 1).

Ng et al. [10] proved that potential-based reward shaping, defined according to Equation 3, does not alter the optimal policy of a single agent in both infinite- and finite- state MDPs.

More recent work on potential-based reward shaping, has removed the assumptions of a single agent acting alone and of a static potential function from the original proof [10]. In multi-agent systems, it has been proven that potential-based reward shaping can change the joint policy learnt but does not change the Nash equilibria of the underlying game [4]. With a dynamic potential function, it has been proven that the existing single and multi agent guarantees are maintained provided the potential of a state is evaluated at the time the state is entered and used in both the potential calculation on entering and exiting the state [5].

2.3 Plan-Based Reward Shaping

Reward shaping is typically implemented bespoke for each new environment using domain-specific heuristic knowledge [3, 12] but some attempts have been made to automate [7, 9] and semi-automate [8] the encoding of knowledge into a reward signal. Automating the process requires no previous knowledge and can be applied generally to any problem domain. The results are typically better than without shaping but less than agents shaped by prior knowledge. Semi-automated methods require prior knowledge to be put in but then automate the transformation of this knowledge into a potential function.

Plan-based reward shaping, an established semi-automated method, generates a potential function from prior knowledge represented as a high level STRIPS plan.

The STRIPS plan is translated[1] into a state-based representation so that, whilst acting, an agent's current state can be mapped to a step in the plan[2] (as illustrated in Figure 2).

The potential of the agent's current state then becomes:

$$\Phi(s) = CurrentStepInPlan * \omega \quad (4)$$

[1]This translation is automated by propagating and extracting the pre- and post- conditions of the high level actions through the plan.

[2]Please note that, whilst we map an agent's state to only one step in the plan, one step in the plan will map to many low level states. Therefore, even when provided with the correct knowledge, the agent must learn how to execute this plan at the low level.

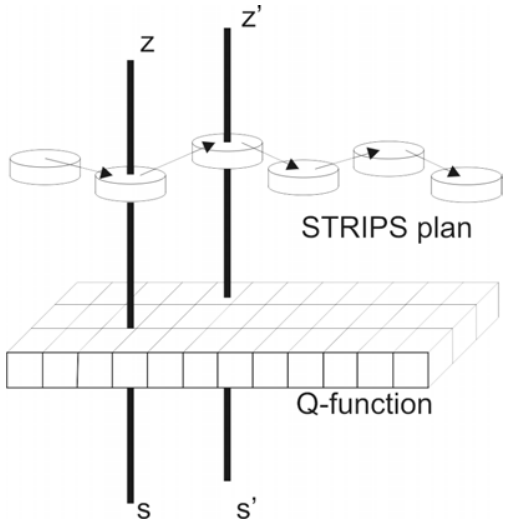


Figure 2: Plan-Based Reward Shaping.

where $CurrentStepInPlan$ is the corresponding state in the state-based representation of the agent's plan and ω is a scaling factor.

To not discourage exploration off the plan, if the current state is not in the state-based representation of the agent's plan then the potential used is that of the last state experienced that was in the plan. This feature of the potential function makes plan-based reward shaping an instance of dynamic potential-based reward shaping [5].

To preserve the theoretical guarantees of potential-based reward shaping, the potential of all goal states is set to zero so that it equals the initial state of all agents in the next episode.

These potentials are then used as in Equation 3 to calculate the additional reward given to the agent and so encourage it to follow the plan without altering the agent's original goal. The process of learning the low-level actions necessary to execute a high-level plan is significantly easier than learning the low-level actions to maximise reward in an unknown environment and so with this knowledge agents tend to learn the optimal policy quicker. Furthermore, as many developers are already familiar with STRIPS planners, the process of implementing potential-based reward shaping is now more accessible and less domain specific [8].

However, this method struggles when given partially incorrect knowledge and, in some cases, fails to learn the optimal policy within a practical time limit. Therefore, in this paper, we propose a generic method to revise incorrect knowledge online allowing the agent to still benefit from the correct knowledge given.

3. EVALUATION DOMAIN

In order to evaluate the performance of adding knowledge revision to plan-based reward shaping the same domain was used as that which is presented in the original work [8]; the flag collection domain.

The flag-collection domain is an extended version of the navigation maze problem which is a popular evaluation domain in RL. An agent is modelled at a starting position from where it must move to the goal position. In between, the agent needs to collect flags which are spread throughout the maze. During an episode, at each time step, the agent is given its current location and the flags it has already collected. From this it must decide to move up, down, left or right and will deterministically complete their move provided they do not collide with a wall. Regardless of the number of flags it has collected, the scenario ends when the agent reaches the goal position. At this time the agent receives a reward equal to one hundred times the number of flags which were collected.

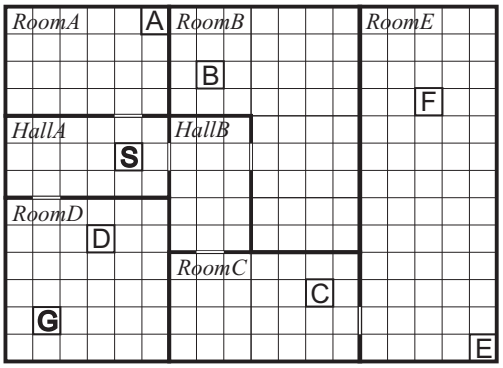


Figure 3: Flag-Collection Domain.

Figure 3 shows the layout of the domain in which rooms are labelled *RoomA-E* and *HallA-B*, flags are labelled *A-F*, S is the starting position of the agent and G is the goal position.

```
MOVE(hallA,roomD)
TAKE(flagD,roomD)
```

Listing 1: Example Partial STRIPS Plan

Given this domain, a partial example of the expected STRIPS plan is given in Listing 1 and the corresponding translated state-based plan used for shaping is given in Listing 2 with the $CurrentStepInPlan$ used by Equation 4 noted in the left hand column.

```
0    robot_in(hallA)
1    robot_in(roomD)
2    robot_in(roomD)  taken(flagD)
```

Listing 2: Example Partial State-Based Plan

3.1 Assumptions

To implement plan-based reward shaping with knowledge revision we must assume an abstract high level knowledge represented in STRIPS and a direct translation of the low level states in the grid to the abstract high level STRIPS states (as illustrated in Figure 2). For example, in this domain the high level knowledge includes rooms, connections between rooms within the maze and the rooms which flags should be present in. Whilst the translation of low level to high level states allows an agent to lookup which room or hall it is in from the exact location given in its state representation.

The domain is considered to be static i.e. there are no external events not controlled by the agent which can at any point change the environment.

It is also assumed that the agent is running in simulation, as the chosen method of knowledge verification currently requires the ability to move back to a previous state.

However, we do not assume full observability or knowledge of the transition and reward functions. When the agent chooses to perform an action, the outcome of that action is not known in advance. In addition, we do not assume deterministic transitions, and therefore the agent does not know if performing an action it has previously experienced will result in transitioning to the same state as the previous time that action was selected. This assumption has a direct impact on the way knowledge verification is incorporated into the agent and is discussed later in the paper. Moreover the reward each action yields at any given state is not given and it is left to the agent to build an estimate of the reward function through continuous interaction with the environment.

Domains limited by only these assumptions represent many domains typically used throughout RL literature. The domain we have chosen allows the agent's behaviour to be efficiently extracted and analysed, thus providing useful insight especially when dealing with novel approaches such as these. Plan-based reward shaping is not, however, limited to this environment and could be applied to any problem domain that matches these assumptions.

Future work is aimed towards extending the above assumptions by including different types of domain knowledge and evaluating the methods on physical environments, real life applications and dynamic environments.

4. IDENTIFYING, VERIFYING AND REVISING FLAWED KNOWLEDGE

In the original paper on plan-based reward shaping for RL [8] there was no mechanism in place to deal with faulty knowledge. If an incorrect plan was used the agent was misguided throughout the course of the experiments and this led to undesired behaviour; long convergence time and poor quality in terms of total reward. Moreover, whenever a plan was produced it had to be manually transformed from an action-based plan as in Listing 1 to a state-based plan as in Listing 2.

In this work we have 1) incorporated the process of identifying, verifying and revising flaws in the knowledge base which is provided to the agent and 2) automated the process of plan transformation. The details are presented in the following subsections.

4.1 Identifying incorrect knowledge

At each time step t the agent performs a low level action a (e.g. move_left) and traverses to a different state s' which is a different square in the grid. When the agent traverses into a new square it automatically picks up a flag if a flag is present in that state. Since the agent is performing low level actions it can gather information about the environment and in this specific case, information about the flags it was able to pick up. This information allows the agent to discover potential errors in the provided knowledge. Algorithm 1 shows the generic method of identifying incorrect knowledge. We illustrate this algorithm with an instantiation of the plan states to the flags the agent should be collecting i.e. predicate `taken(flagX)` shown in Listing 2. The preconditions are then instantiated to the preconditions which achieve the respective plan state which in this study refers to the presence of flags. The same problem instantiation is also used in the empirical evaluation.

Algorithm 1 Knowledge identification.

```
get plan states preconditions
initialise precondition confidence values
for episode = 0 to max_number_of_episodes do
  for current_step = 0 to max_number_of_steps do
    if precondition marked for verification then
      switch to verification mode
    else
      plan-based reward shaping RL
    end if
  end for/* next step */
  /* update the confidence values */
  for all preconditions do
    if precondition satisfied then
      increase confidence
    end if
  end for
  /* check preconditions which need to be marked for
  verification */
  for all preconditions do
    if confidence value < threshold then
      mark the current precondition for verification
    end if
  end for
end for/* next episode */
```

More specifically, at the start of each experiment, the agent uses the provided plan in order to extract a list of all the flags it should be able to pick up. These flags are then assigned a *confidence value* much like the notion of *epistemic entrenchment* in belief revision [6]. The confidence value of each flag is set to the ratio $successes/failures$ and is computed at the end of each episode with $successes$ being the number of times the agent managed to find the flag up to the current episode, and $failures$ the times it failed to do so. If the confidence value of a flag drops below a certain threshold, that flag is then marked for verification.

This dynamic approach is used in order to account for the early stages of exploration where the agent has not yet built an estimate of desired states and actions. If a static approach were to be used which would only depend on the total number of episodes in a given experiment, failures to pick up flags would be ignored until a much later point in the experiment and the agent would not benefit from the revised knowledge at the early stages of exploration. Additionally, varying the total number of episodes would have a direct impact on when knowledge verification will take place.

4.2 Knowledge verification

When a flag is marked for verification the agent is informed of which flag has failed at being picked up and the abstract position it should appear in e.g. *RoomA*. The agent is then left to freely interact with the environment as in every other case but its mode of operation is changed once it enters the abstract position of the failing flag. At that point the agent will perform actions in order to try and verify the existence of the flag which is failing. Algorithm 2 shows the generic method of verifying incorrect knowledge by the use of depth first search (DFS). The algorithm is illustrated by using the same instantiations as those in Algorithm 1.

To verify the existence of the flag the agent performs a DFS of the low-level state space within the bounds of the high-level abstract state of the plan. A node in the graph is a low-level state s and the edges that leave that node are the available actions a the agent can perform at that state. An instance of the search tree is shown in Figure 4 in which the grey nodes ($N1-N3$) have had all their edges expanded and green nodes ($N4-N7$) have unexpanded edges ($E9-E14$).

However, instead of performing DFS in terms of nodes the search is performed on edges. At each time step instead of selecting to expand a state (node), the agent expands one of the actions (edges). The search must be modified in this way because of our assumptions on the environment the agent is acting in. When an agent performs an action a at state s it ends up at a different state s'. The transition probabilities of those actions and states however are not known in advance. As a result the agent cannot choose to transition to a predefined state s', but can only choose an action a given the current state s. Performing DFS by taking edges into account enables the agent to search efficiently while preserving the theoretical framework of RL.

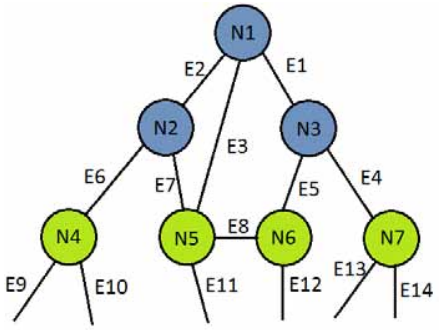


Figure 4: Instance of the Search Tree.

After expanding an edge/making an action the agent's coordinates in the grid are stored along with the possible actions it can perform. The graph is expanded with new nodes and edges each time the agent performs an action which results in a transition to coordinates which have not been experienced before. If the agent transitions to coordinates which correspond to an existing node in the graph, it simply selects to expand one of the unexpanded edges i.e. perform an action which has not been tried previously.

If a node has had all of its edges expanded (i.e. all of the available actions at that state have been tried once) the node is marked as *fully expanded*. However, instead of backtracking as it happens in traditional DFS, the agent jumps to the last node in the graph which has unexpanded edges. This approach ensures that the assumptions on the domain regarding transition probabilities are not violated as a reverse action does not necessarily exist. A similar jump is performed in the case where expanding a node leads the agent into breaking the search condition i.e. the agent steps out of the room which contains the flag which is failing. It is worth noting that while the agent performs DFS, in order to be fair when comparing with other approaches, expanding an edge or jumping to a different node takes a time step to complete. In the context of this work performing each available action only once is sufficient since we have a deterministic domain. A stochastic domain would require each action to be performed multiple times before marking a node as *fully expanded*. If the agent was to be acting in a physical environment, the knowledge verification would not be by DFS but by heuristic search relying on the agent's sensors of the environment e.g. search at places not directly visible by the camera.

Algorithm 2 Knowledge verification.

```
get state
get precondition marked for verification
if all nodes in the graph are marked as fully expanded then
    mark precondition for revision
    stop search
    break
end if
if search condition is violated then
    jump to a node in the graph with unexpanded edges
    break
end if
if state is not present in the graph then
    add state and available actions as node and edges in the graph
end if
if all edges of current node have been expanded then
    mark node as fully expanded
    jump to a node in the graph with unexpanded edges
    break
end if
expand random unexpanded edge
mark edge as expanded
if precondition has been verified then
    reset precondition confidence value
    stop search
end if
```

The search finishes once the agent has either found the failing flag or all of the nodes that were added to the graph have been marked as *fully expanded*. If found, the confidence value associated with the flag is reset and the agent returns to normal operation. If not, the agent returns to normal operation but the flag is marked for revision.

It is worth noting that the search does not have a cut-off value considering the small size of the grid graph the agent needs to search in. Furthermore, whilst verifying knowledge, no RL updates are made. The reason is for the agent not to get penalised or rewarded by following random paths while searching which would otherwise have a direct impact on the learnt policy.

4.3 Revising the knowledge

As discussed previously, when an agent fails to verify the existence of a flag, that flag is marked for revision. Belief revision is concerned with revising a knowledge base when new information becomes apparent by maintaining consistency among beliefs [6]. In the simplest case where the belief base is represented by a set of rules there are three different actions to deal with new information and current beliefs in a knowledge base: expansion, revision and contraction. In this specific case, where the errant knowledge the agent has to deal with is based on extra flags which appear in the knowledge base but not in the simulation, revising the knowledge

base requires a contraction[3]. Furthermore, since the beliefs in the knowledge base are independent of each other, as the existence or absence of a flag does not depend on the existence or absence of other flags, contraction equals deletion. The revised knowledge base is then used to compute a more accurate plan.

To illustrate the use of this method consider a domain similar to that shown in Figure 3 which contains one flag, *flagA* in *roomA*. The agent is provided with the plan shown in Listing 3. This plan contains an extra flag which is not present in the simulator, *flagC* in *roomC*. According to the plan the agent starts at *hallA* and has to collect *flagA* and *flagC* and reach the goal state in *roomD*.

```
0 robot_in(hallA)
1 robot_in(hallB)
2 robot_in(roomC)
3 robot_in(roomC) taken(flagC)
4 robot_in(hallB) taken(flagC)
5 robot_in(hallA) taken(flagC)
6 robot_in(roomA) taken(flagC)
7 robot_in(roomA) taken(flagC) taken(flagA)
8 robot_in(hallA) taken(flagC) taken(flagA)
9 robot_in(roomD) taken(flagC) taken(flagA)
```

Listing 3: Example Incorrect Plan

Let's assume that the verification threshold for each flag is set at 0.3. At the end of the first episode the confidence value of each flag is computed. If *flagA* was picked up, its threshold will be equal to 1 which is greater than the verification threshold. As a result this flag will not be marked for verification. However, since *flagC* does not appear in the simulator its confidence value will be equal to 0, which is less than the verification threshold, and the flag will be marked for verification.

During the next episode when the agent will step into the room where *flagC* should appear in i.e. *roomC*, it will switch into verification mode. At this point the agent will perform a DFS within the bounds of *roomC* to try and satisfy *flagC*. The DFS will reveal that *flagC* cannot be satisfied and as a result will be marked for revision. When the episode ends the knowledge base will be updated to reflect the revision of *flagC* and a new plan will then be computed. The new plan is shown in Listing 4.

```
0 robot_in(hallA)
1 robot_in(roomA)
2 robot_in(roomA) taken(flagA)
3 robot_in(hallA) taken(flagA)
4 robot_in(roomD) taken(flagA)
```

Listing 4: Example Correct Plan

5. EVALUATION

In order to assess the performance of this novel approach a series of experiments were conducted in which the agent was provided with flawed knowledge in terms of missing flags. Specifically the agent was given different instances of wrong knowledge: 1) one non-existing flag in the plan, 2) two non-existing flags in the plan and 3) three non-existing flags in the plan. This setting was chosen in order to assess how the agent deals with the increasing number of flaws in the knowledge and what the impact is on the convergence time in terms of the number of steps, and the performance in terms of the total accumulated reward.

All agents implemented SARSA with $\epsilon-$greedy action selection and eligibility traces. For all experiments, the agents' parameters were set such that $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon = 0.1$ and $\lambda = 0.4$. All initial Q-values were set to zero and the threshold at which a flag should be marked for verification was set to 0.3.

These methods, however, do not require the use of SARSA, $\epsilon-$greedy action selection or eligibility traces. Potential-based reward shaping has previously been proven with Q-learning and RMax [1]. Furthermore, it has been shown before without eligibility traces [10, 3] and proven for any action selection method that chooses actions based on relative difference and not absolute magnitude [1].

In all our experiments, we have set the scaling factor of Equation 4 to:

$$\omega = MaxReward/NumStepsInPlan \tag{5}$$

As the scaling factor affects how likely the agent is to follow the heuristic knowledge, maintaining a constant maximum across all heuristics compared ensures a fair comparison. For environments with an unknown maximum reward the scaling factor ω can be set experimentally or based on the designer's confidence in the heuristic.

Each experiment lasted for 50 000 episodes and was repeated 10 times for each instance of the faulty knowledge. The agent is compared to the original plan-based RL agent [8] without knowledge revision when provided with incorrect knowledge and when the same agent is provided with correct knowledge. The averaged results are presented in Figures 5, 6, 7 and 8. For clarity these figures only display results up to 5000 episodes, after this time no significant change in behaviour occurred.

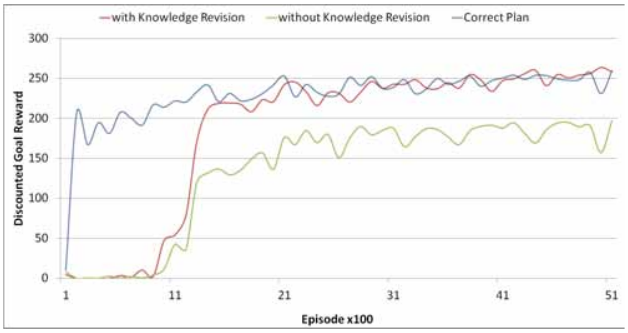


Figure 5: Non-existing flags: 1 flag.

It is apparent that the plan-based RL agent without knowledge revision is not able to overcome the faulty knowledge and performs sub-optimally throughout the duration of the experiment. However, the agent with knowledge revision manages to identify the flaws in the plan and quickly rectify its knowledge. As a result after only a few hundred episodes of performing sub-optimally it manages to reach the same performance as the agent which is provided with

[3]A rule ϕ, along with its consequences is retracted from a set of beliefs K. To retain logical closure, other rules might need to be retracted. The contracted belief base is denoted as $K\dot{-}\phi$. [6]

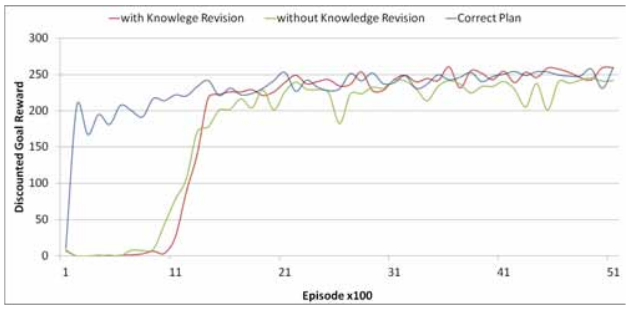


Figure 6: Non-existing flags: 2 flags.

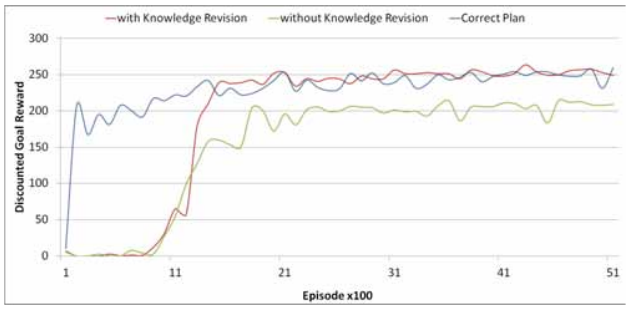


Figure 7: Non-existing flags: 3 flags.

correct knowledge[4].

The agents were provided with more instances of incorrect knowledge reaching up to eight missing flags and similar results occurred on all different instances of the experiments with the agent using knowledge revision outperforming the original plan-based agent.

In terms of convergence time Figure 8 shows the number of steps each agent performed on average per experiment. It is clear that the plan-based agent with knowledge revision manages to improve its learning rate by almost 40%. The agent with correct knowledge is outperforming both agents but there is a clear improvement in the plan-based RL agent with knowledge revision which manages to outperform the agent without knowledge revision by a large margin.

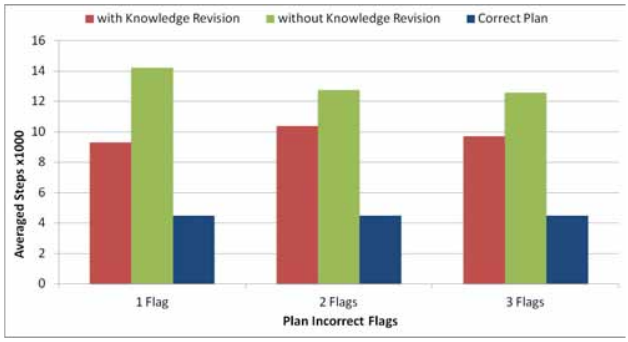


Figure 8: Average number of steps taken per experiment.

[4]Please note the agents' illustrated performance does not reach 600 as the value presented is discounted by the time it takes the agents to complete the episode.

These empirical results demonstrate that when an agent is provided with incorrect knowledge, knowledge revision allows the agent to incorporate its experiences to the provided knowledge base and thus benefit from more accurate plans.

6. CLOSING REMARKS

When an agent receiving plan-based reward shaping is guided by flawed knowledge it can be led to undesired behaviour in terms of convergence time and overall performance in terms of total accumulated reward.

Our contribution is a novel generic method for identifying, verifying and revising incorrect knowledge if provided to a plan-based RL agent.

Our experiments show that using knowledge revision in order to incorporate an agent's experiences to the provided high level knowledge can improve its performance and help the agent reach its optimal policy. The agent manages to revise the provided knowledge early on in the experiments and thus benefit from more accurate plans.

Although we have demostrated the algorithm in a grid world domain, it can be successfully applied to any simulated, static domain where some prior heuristic knowledge and a mapping from low level states to abstract plan states is provided.

In future work we intend to investigate the approach of automatically revising knowledge when different types of flawed knowledge (incomplete e.g. the provided plan is missing states the agent should achieve, stochastic e.g. certain states the agent should achieve in the plan cannot always be achieved, and combinations of these) are provided to an agent. Additionally we aim to evaluate the algorithm on physical environments, real life applications and dynamic environments.

7. ACKNOWLEDGEMENTS

This study was partially sponsored by QinetiQ under the EPSRC ICASE project *"Planning and belief revision in reinforcement learning"*.

8. REFERENCES

[1] J. Asmuth, M. Littman, and R. Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 604–609, 2008.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control (2 Vol Set)*. Athena Scientific, 3rd edition, 2007.

[3] S. Devlin, M. Grześ, and D. Kudenko. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 2011.

[4] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of The Tenth Annual International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[5] S. Devlin and D. Kudenko. Dynamic potential-based reward shaping. In *Proceedings of The Eleventh Annual International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[6] P. Gärdenfors. Belief revision: An introduction. *Belief revision*, 29:1–28, 1992.

[7] M. Grześ and D. Kudenko. Multigrid Reinforcement Learning with Reward Shaping. *Artificial Neural Networks-ICANN 2008*, pages 357–366, 2008.

[8] M. Grześ and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *Proceedings of the 4th IEEE International Conference on Intelligent Systems (IS'08)*, pages 22–29. IEEE, 2008.

[9] B. Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th International Conference on Machine learning*, page 608. ACM, 2007.

[10] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, 1999.

[11] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., New York, NY, USA, 1994.

[12] J. Randløv and P. Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pages 463–471, 1998.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Context sensitive reward shaping in a loosely coupled multi-agent system

Yann-Michaël De Hauwere
Computational Modeling Lab
Vrije Universiteit Brussel
Belgium
ydehauwe@vub.ac.be

Sam Devlin
Department of Computer Science
University of York
United Kingdom
devlin@cs.york.ac.uk

Daniel Kudenko
Department of Computer Science
University of York
United Kingdom
kudenko@cs.york.ac.uk

Ann Nowé
Computational Modeling Lab
Vrije Universiteit Brussel
Belgium
anowe@vub.ac.be

ABSTRACT

Reward shaping is a commonly used approach in single agent reinforcement learning to speed up the learning process. Potential based reward shaping has recently found its way to improve the performance of multi-agent reinforcement learning. Both in single and multi-agent settings these speedups are achieved without losing any theoretical convergence guarantees. This paper describes the use of context aware potential functions in a loosely coupled multi-agent system. In some multi-agent settings, the interactions between the agents only occur sporadically, in certain regions of the state space. It is clear that if speedups through reward shaping are to be achieved, that a different shaping signal should be used in these different regions. We demonstrate how this can be achieved within FCQ-learning, which is an algorithm capable of automatically detecting when agents should take each other into consideration. Coordination problems can even be anticipated before the actual problems occur.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms

Keywords

Reinforcement Learning, Sparse Interactions, Reward Shaping

1. INTRODUCTION

The most common issue in multi-agent reinforcement learning (MARL) is the exponential increase of the state space with each additional agent. Agents could of course only observe local state information, but due to this lack of observability of the other agents, the world appears non-deterministic to each agent. This exponential increases results in impractically long learning times, whereas this lack of observability results in suboptimal policies.

In single agent reinforcement learning (RL), one commonly used approach to speed up the learning is to incorporate domain knowledge about the problem at hand, by means of reward shaping. This technique is the process of providing an agent with additional rewards in order to guide the learning process to achieve a faster convergence time. Recent work has aimed at applying this method in multi-agent environments by using a joint plan of the solution in order to shape the learning agents [5].

In certain multi-agent environments, the agents only rarely interact with each other. Hence that state information about other agents should only be present in the state space in those situations where agents are influencing each other. In situations where agents are not influencing each other, single agent RL is applied. Research in this domain is focused around learning when agents should coordinate their actions [11] and learn when agents should augment their state space to include information from other agents [12, 3].

The concept of delayed rewards, which is common in RL problems, was also recently incorporated in a multi-agent learning algorithm with sparse interactions, called FCQ-learning [4]. In this paper we build upon this algorithm and extend it by means of reward shaping. Both in states where agents are learning individually and in states where agents use augmented state information this reward shaping is applied, albeit a different one. The main idea we present here is that based on the context of agents, a different reward shaping should be applied which helps them to achieve the particular subgoal of the context they are currently in.

The remainder of this paper is structured as follows. First we will present the basic background about single and multi-agent RL, reward shaping and sparse interactions. Next, we explain in detail how different reward shaping functions are incorporated in FCQ-learning. We evaluate our algorithm empirically in Section 5 and end with some conclusions.

2. SINGLE AND MULTI-AGENT REINFORCEMENT LEARNING

2.1 Single agent RL

Reinforcement Learning (RL) is an approach to solve a Markov Decision Process (MDP), where an MDP can be described as follows. Let $S = \{s_1, \ldots, s_N\}$ be the state space of a finite Markov chain $\{x_l\}_{l \geq 0}$ and let $A = \{a^1, \ldots, a^r\}$ be the action set available to the agent. Each combination of starting state s_i, action choice $a^i \in A$ and next state s_j has an associated transition probability $T(s_i, a^i, s_j)$ and immediate reward $R(s_i, a^i)$. The goal is to learn a policy π, which maps an action to each state so that the expected discounted reward J^π is maximised:

$$J^\pi \equiv E\left[\sum_{t=0}^{\infty} \gamma^t R(s(t), \pi(s(t)))\right] \quad (1)$$

where $\gamma \in [0, 1)$ is the discount factor and expectations are taken over stochastic rewards and transitions. This goal can also be expressed using Q-values which explicitly store the expected discounted reward for every state-action pair:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a') \quad (2)$$

So in order to find the optimal policy, one can learn this Q-function and subsequently use greedy action selection over these values in every state. In RL the agent typically does not have any knowledge about the underlying model, i.e. the transition and reward function. Watkins described an algorithm to iteratively approximate the optimal values Q^*. In the Q-learning algorithm [21], a table consisting of state-action pairs is stored. Each entry contains the value for $\hat{Q}(s, a)$ which is the learner's current hypothesis about the actual value of $Q(s, a)$. The $\hat{Q}$-values are updated according to following update rule:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha_t [R(s, a) + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)] \quad (3)$$

where α_t is the learning rate at time step t.

Provided that all state-action pairs are visited infinitely often and a appropriate learning rate is chosen, the estimates $\hat{Q}$ will converge to the optimal values Q^* [18].

Reward shaping in single agent RL

In many RL problems, the agent is only awarded a reward upon achieving a certain goal. As can be seen from Equation 3, this reward is propagated through the state space. This process is however time consuming. One approach to speed up this process are eligibility traces. This technique maintains a record of visited state-action pairs and therefore knows, which state-action paris are eligible to be updated. The last visited state-action has an eligibility set to 1. All previous pairs are multiplied by λ, where $\lambda < 1$. By updating all state-action pairs by the temporal difference multiplied with their eligibility, Q-values propagate quicker [17].

Another approach to overcome this problem, known as reward shaping, was introduced [14, 15]. Reward shaping is the principle of giving additional numerical feedback to intermediate states that guide the learning agent. Since manually determining which additional reward should be given can be a difficult or even impossible task, Ng et al. introduced potential-based reward shaping (PBRS) [14]. In PBRS, the additional reward given to the agent is defined as the difference of a potential function Φ, defined over a source state s and a destination state s':

$$F(s, s') = \gamma \Phi(s') - \Phi(s) \quad (4)$$

where γ has the same value as in the Q-update rule which now takes following form:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha_t [R(s, a) + F(s, s') + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)] \quad (5)$$

As such, Ng et al proved a single agent's optimal policy would be unchanged by additional potential-based rewards (or more generally that the order of policies ranked by value would remain constant). Grzes and Kudenko introduced a way of automatically generating these potential functions from a high level STRIPS operator [9] which relaxes the requirement for having knowledge about the underlying MDP which is being solved [16].

2.2 Multi agent RL

Problems in which multiple agents are acting simultaneously, are modelled as a Markov Game (MG). In a MG, the state information $s_i \in S$ is the set of system states, which contains the information of all the agents. Actions are the joint result of multiple agents choosing an action individually. Let $A_k = \{a_k^1, \ldots, a_k^r\}$ be the action set available to agent k, and $k : 1 \ldots n$, n being the total number of agents present in the system. Transition probabilities $T(s_i, a^i, s_j)$ in a MG depend on a starting state s_i, ending state s_j and a joint action from state s_i, i.e. $a^i = (a_1^i, \ldots, a_n^i)$ with $a_k^i \in A_k$. The reward function $R_k(s_i, a^i)$ is now individual to each agent k, meaning that agents can receive different rewards for the same state transition.

In a special case of the general Markov game framework, the so-called team games or multi-agent MDPs (MMDPs) optimal policies still exist [1, 2]. In this case, all agents share the same reward function and the Markov game is purely cooperative. This specialisation allows us to define the optimal policy as the joint agent policy, which maximises the payoff of all agents. In the non-cooperative case one typically tries to learn an equilibrium between agent policies [10, 8, 20]. These systems require each agent to calculate equilibria between possible joint actions in every joint state and as such assume that each agent retains estimates over all joint actions in all states. Moreover, these algorithms also face a selection problem if multiple equilibria exist.

Reward shaping in multi agent RL

From the description above, it is clear that learning in MAS adds several additional problems, compared to learning in single agent systems. One of the most important ones, being the exponential increase in the state-action space in which the agents are learning. This significantly slows down the convergence of the agent and as a consequence their ability to adapt quickly to the changing environment [19].

Recently Devlin et al. introduced a method for transforming coordinated plans made together into a potential function to shape the rewards of the agents and guide their exploration in MAS [5]. The authors illustrated that re-

ward shaping in MAS also offers an elegant way to incorporate domain knowledge in the learning process and as such speed up convergence. Moreover, it is also proven that potential-based reward shaping in multi-agent systems does not change the Nash equilibria of the underlying stochastic game the agents are playing [6].

In this paper we use the approach of [5] to generate a potential function from a given individual greedy plan. The potential of the agent's current state is given by:

$$\Phi(s) = CurrentStepInPlan * w \quad (6)$$

where *CurrentStepInPlan* is the position of the current state of the agent in its plan. w is a scaling factor defined as follows:

$$w = MaxReward/NumStepsInPlan \quad (7)$$

As the scaling factor affects how likely the agent is to follow the heuristic knowledge, maintaining a constant maximum across all heuristics compared ensures a fair comparison. For environments with an unknown maximum reward the scaling factor w can be set experimentally or based on the designer's confidence in the heuristic.

3. SPARSE INTERACTIONS

3.1 Concept

In many multi-agent environments, agents only need to interact with each other in specific regions of the state space because in these regions agents can interfere with each other's performance, i.e., they are not independent of each other. These interdependencies that occur in certain states are called sparse interactions [13]. Learning these sparse interactions have gained a lot of attention in recent years, since they significantly reduce the state-action space in which the agents are learning [12, 3].

Melo & Veloso learn a set of interaction states, i.e. states in which agents influence each other, by using an active perception mechanism that can provide exactly this information. By attaching a cost to the use of this mechanism, the learning agent learns the set of states in which using this mechanism is better than to just ignore the other agents. The latter would result in a higher penalty [12].

De Hauwere et al. use statistical tests in CQ-learning on the immediate reward signal to detect influence of other agents and will augment the state space to include state information about other agents to avoid coordination problems. In later work the authors introduced FCQ-learning which will be described in depth in the next section as this is the basis for the work presented in this paper [3].

Kok & Vlassis introduced an approach where agents learn when it is beneficial to coordinate with other agents in fully cooperative games [11]. They attempt to reduce the action space which agents need to consider, by letting the agents learn individually, and detecting differences in the reward signal based on different joint actions. Only in states where a dependency between the agents exists the agents will coordinate by selecting an action together in the joint action space.

3.2 FCQ-learning

In this paper we focus on enhancing the learning process through reward shaping of one technique that learns from sparse interactions. This technique, called FCQ-learning, is capable of detecting the influence from other agents several time steps ahead.

The idea is that agents learn in which of their local states they will augment their state information to incorporate the information of other agents and use a more global system state. This idea is represented in Figure 1. The local states for one agent are represented at the bottom. In its local states labeled 4 and 6 it augmented its information to include global state information illustrated at the top. Note that local state information only contains state information about the agent itself such as its own location, battery level, other sensory input, ...

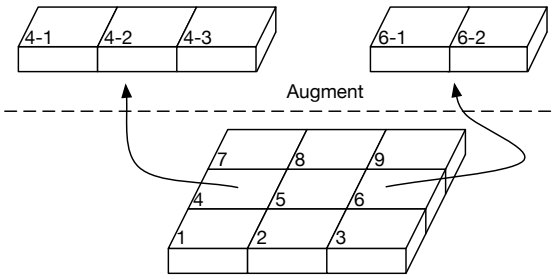


Figure 1: The state information of states 4 and 6 is augmented to incorporate additional information in order to solve coordination problems.

Given this information the most important challenge is detecting in which states, the state information must be augmented. FCQ-learning samples the returns it collects, and groups them according to its local state and the local state of other agents. Using these samples, the agent can perform a Friedmann statistical test which can identify the significance of the difference between the different local states of the other agents for its own local state. This principle is represented in Figure 2. Agent 1 starts sampling the rewards until termination of the episode in local state x^i based on the local state information y^i,y^j and y^k of Agent 2. If enough samples have been collected and if a significant difference is detected among these states, the state information for the local state of agent 1 is augmented to include the information of the other agent for which the difference was detected (as illustrated in Figure 1).

At every time step, when the agent selects an action, it will check if its current local state is a state which has been augmented and contains the state information of other agents. In this case, it will observe the global state to determine if it contains the information from its augmented state. If so, it will condition its action based on this augmented state information, otherwise it can act independently using only its own local state information. If its local state information has never been augmented it can also act without taking the other agents into consideration.

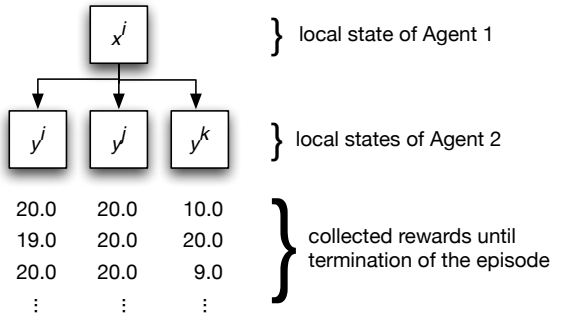


Figure 2: Agent 1 in local state x^i is collecting rewards until termination of the episode based on the local state information of agent 2.

We distinguish two situations for updating the Q-values:

1. An agent is in an augmented state. Following update rule is used:

$$Q_k^{aug}(js, a_k) \leftarrow (1-\alpha_t)Q_k^{aug}(js, a_k) + \alpha_t[r(js, a_k) + \gamma \max_{a'_k} Q_k(s', a'_k)] \quad (8)$$

   where Q_k stands for the Q-table containing the local states, and Q_k^{aug} contains the augmented states (js). Note that this augmented Q-table is initially empty as we use a bottom up approach to learning sparse interactions. The Q-values of the local states of an agent are used to bootstrap the Q-values of the states that were augmented with global state information. This is to capture the idea of a sparse interaction, where agents only influence each other in certain states, after which they can act independently again.

2. An agent is in a local state. The Q-learning rule of Equation 3 is used with only local state information.

We do not consider the case where we use the Q-table with joint states to bootstrap in our update scheme since at timestep t an agent can not know at that time that it will be in a state where coordination will be necessary at timestep $t+1$ as this will also depend on he actions of other agents.

For every augmented state a confidence value is maintained which indicates how certain the algorithm is that this is indeed a state in which coordination might be beneficial. If the augmented state of a local state is visited, the confidence value is increased. Otherwise the confidence values of all the augmented states, that have been created from the current local state are decreased. This ensures that states, where an agent would request state information about another agent, but where this state information does not correspond to the augmented state, are reduced again to states where agents only consider local state information. By reducing this value less than we increase it, we built some fault tolerance against too quickly reducing states again.

The algorithm is more formally described in Algorithm 1.

Algorithm 1 FCQ-Learning algorithm for agent k

```
Initialise Q_k and Q_k^aug to zero;
while true do
  if ∀ Agents k, state s_k of Agent k is a safe state then
    Select a_k for Agent k from Q_k
  else
    Select a_k for Agent k from Q_k^aug
  end if
  Store the state information of other agents, and collect
  the rewards until termination of the episode
  if enough samples have been collected then
    perform Friedmann test on the samples for the state
    information of the other agents. If the test indicates
    a significant difference, augment s_k to include state
    information of the other agents
  end if
  if s_k is an augmented state for Agent k then
    Update Q_k^aug(js) ← (1 − α_t)Q_k^aug(js) +
    α_t[r(js, a_k) + γ max_a Q_k(s'_k), a]
    increment confidence value for s_k
  else
    Update Q_k(s) ← (1 − α_t)Q_k(s) + α_t[r(js, a_k) +
    γ max_{a_k'} Q_k(s'_k, a'_k)].
    decrease confidence value for s_k if extra state information was requested
  end if
end while
```

4. CONTEXT SENSITIVE REWARD SHAPING

An agent, learning in an environment, can have different subgoals it is trying to achieve, while accomplishing the global goal of the task at hand. Incorporating all these subgoals into one shaping function is not always possible. The idea we present here is to have different appropriate shaping functions, that depend on the context the agent is currently in. This context is defined by the subgoal it is currently trying to accomplish. The shaping function for a context will guide the agent in achieving the particular goal of that context. These shaping functions can be defined for one agent alone, or can be generated from a joint plan, including other agents' state. As acknowledged by [5], individual plans might contain conflicting knowledge, which results in agents interfering with each other. On the other hand, shaping functions based on joint plans require these plans to be available and interferences to be identified beforehand. As this may not always be possible, it is a better idea to use a different shaping function in these different contexts: *interfering* or *not interfering*. If these interferences can be detected automatically, agents can autonomously switch to a different shaping function, which is engineered for the particular context the agent is currently in. This allows the designer of the system to have multiple simple shaping functions, rather than try to build one shaping function that covers all the subtleties that occur when multiple agents act in the same environment.

We implement our approach to context sensitive reward shaping by using FCQ-learning to detect the different contexts. FCQ-learning samples the state space and will automatically augment certain states to include state informa-

tion about other agents if the agent is influenced by them. This means that an agent can be in one of following contexts:

1. **Individual:** The agent is not influenced by any other agent and only uses local state information

2. **Coordinating:** The agent is influenced by another agent and uses augmented state information.

As explained above, a different shaping function is to be used in these different context since they have different subgoals. As an example to clarify this, we explain the setup of the experiments we used. These are navigation tasks, where both agents have to reach the same location but the order in which they reach this location will determine the reward they receive. Agent 1 needs to enter the goal location before agent 2. The global goal of the system is for both agents to reach the goal location in the correct order. Agents are initially unaware of the presence of the other agent or the order in which they need to reach the goal.

In the *Individual* context the shaping function is used is calculated as described in Section 2.2. The potential is calculated from the shortest path from the initial position to the goal. So the potential is actually a gradient that steers the agent in the direction of the goal, as if it was a beacon. In this context, agents ignore eachother and are only concerned with reaching the goal.

In the *Coordinating* context the global goal of the system is used to shape the rewards. Agent 1, which has to reach goal first, is further guided towards the goal. Agent 2 receives an incentive to not reduce the distance between itself and agent 1. As such we are certain that agent 1 can reach the goal first and accomplish the global goal of the system. This principle is similar to a separation-based heuristic as described in [7].

The idea is presented visually in Figure 3

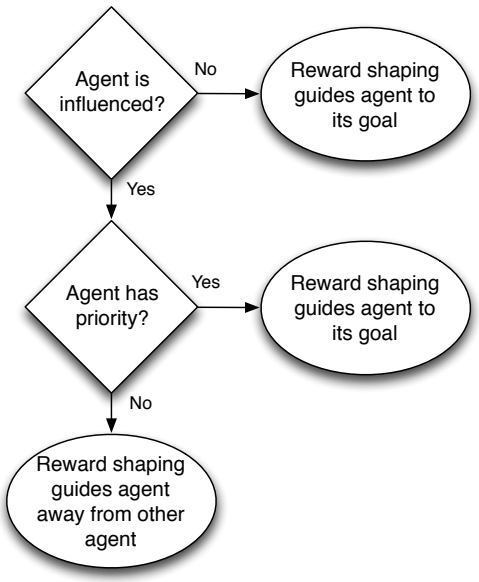


Figure 3: Context sensitive use of reward shaping in MAS

5. EXPERIMENTS

To validate this approach to reward shaping within the FCQ-learning framework, we use two of the same environments of FCQ-learning (*Grid game 2* and *TunnelToGoal*) and compare FCQ-learning with and without shaping. The environments are navigation tasks in which the agents have to reach a predefined goal in the environment from an initial position. The environments used are shown in Figure 4.

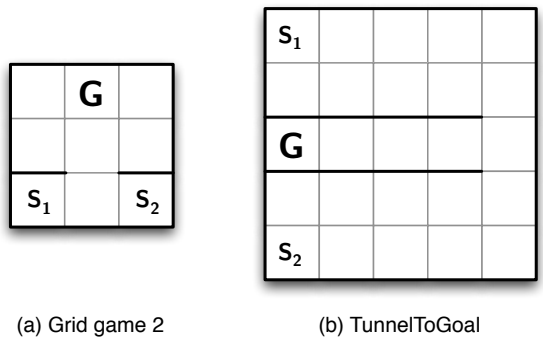


Figure 4: Navigation games

The initial positions are marked with S_i for each agent i. The goal position is indicated with the letter G. Each cell can only be occupied by one agent at any given time. These environments induce some form of coordination problem where following the shortest path to the respective goals of the agents would result in collisions between them. Moreover, agents need to coordinate during the episode in order to enter the goal in the correct order. I.e. to achieve the highest reward agent 1 needs to reach the goal before agent 2. All experiments were run for $10,000$ learning episodes (an episode is completed if all agents reach their respective goal state) and averaged over 20 independent runs. The experiments were using a learning rate of 0.2. Exploration was regulated using a fixed ϵ-greedy policy with $\epsilon = 0.09$ and a discount factor $\gamma = 0.95$ was used. Reaching the goal results in a reward of $+20$ in *Grid game 2* and of $+600$ in *TunnelToGoal* if the agents reach the goal in the correct order. Otherwise, the maximum reward they can achieve is $+10$ in *Grid game 2* and $+300$ in *TunnelToGoal*.

If the agents collide they remain in the location they were in before the collision and receive a penalty of -10 for colliding. Transitions succeed in 90% of the cases. In the other 10% agents are moved to one of the adjacent cells at random.

We will begin by describing the qualitative results of the learned policies for FCQ-learning with and without a shaping function. We compare the number of steps the agents need to complete an episode and how often they collide during an episode, the average reward they received per episode, the size of the statespace in which they are learning and how often they use the augmented state space. These results were obtained by taking the average over the last 100 episodes.

The results in the *Grid game 2* environment are actually very similar. Since this is a relative small environment, this is to be expected.

In the *TunnelToGoal* environment we see bigger changes. The number of steps is almost divided by two to complete an

	FCQ	FCQ with shaping
# steps	21.1	15.48
reward	12.96	14.25
statespace	10.97 & 15.24	10.36 & 13.59
collisions	0.16	0.16
# joint plays	2.29	2.29

Table 1: Final results for the Grid game 2 environment

	FCQ	FCQ with shaping
# steps	30.01	15.8
reward	217.8	276.72
statespace	48.59 & 57.55	37.41 & 39.58
collisions	0.42	0.20
# joint plays	6.0	2.23

Table 2: Final results for the TunnelToGoal environment

episode. This number indicates the number of steps required for both agents to complete an episode. Hence, this is the number of steps required for the slowest agent to reach the goal. In this environment, surprisingly this turned out to be agent 1 if no shaping was used, even though this agent had to reach the goal first in order to reach the highest reward. The result of this, can be seen by looking at the row in which the average reward per episode is shown. Even though FCQ-learning without reward shaping has a larger state space and uses augmented state information more often per episode than with reward shaping, it does not manage to reach the same performance level. This is due to the fact that the incorporation of background knowledge in the shaping function when using augmented state information significantly helps the agent reach the optimal solution (in which agent 1 has to reach the goal first).

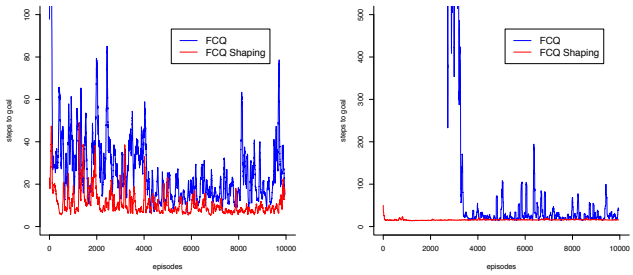


Figure 5: Average number of steps to complete an episode for Grid game 2 (left) and TunnelToGoal (right)

In Figure 5 we show the number of steps needed to complete an episode during the learning process. On the left we see the results for the *Grid game 2* environment on the right the results for the *TunnelToGoal* environment. In the larger TunnelToGoal environment we clearly see the effect of reward shaping. Agents are learning a lot faster to reach their goal.

The consequence of this increase in learning speed is also reflected in Figure 6. This figure shows the average number

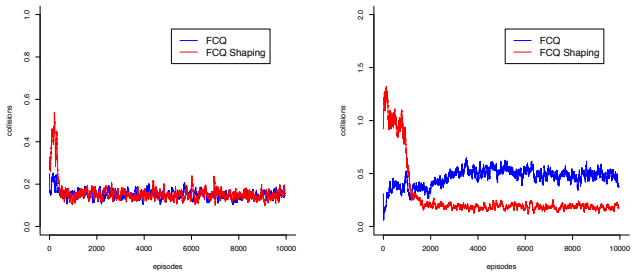


Figure 6: Average number of collisions per episode for Grid game 2 (left) and TunnelToGoal (right)

of collisions per episode. In the beginning of the learning process, while the agents are still identifying joint states, we see a higher number of collisions in the FCQ variant with reward shaping. Since the shaping helps them reach their goal faster, the number of collisions is also increased sooner. Note that these collisions are necessary for FCQ-learning to identify the states in which they should use augmented state information. If we would let agents learn independently with independent reward shaping, the shaping function would lead them to states in which agents are influencing each other and in which they would receive low rewards. FCQ-learning on the other hand switches to a different shaping function in these states in order to solve the coordination problem that occurs in these states. This happens around learning episode 1000, after which we see a sudden drop in the number of collisions per episode. In *Grid game 2* we see the same effect, albeit a lot more subtle since this environment is a lot smaller than *TunnelToGoal*.

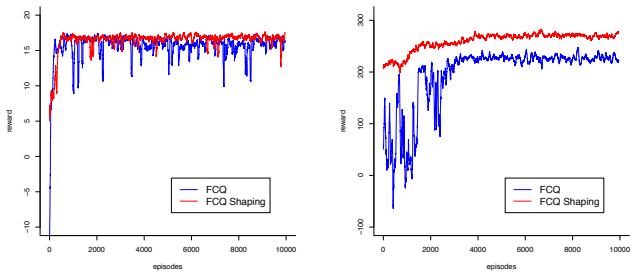


Figure 7: Average reward per episode for Grid game 2 (left) and TunnelToGoal (right)

The rewards obtained by the agents per episode, shown in Figure 7, also follow this evolution and around episode 1000 we see an increase in the obtained rewards, due to this decrease in the number of collisions. If no shaping function is used, it takes FCQ learning a lot longer to reach a stable policy, which is not as good as when a shaping function is used. This is largely due to the fact that this shaping function reflects how the global goal of the system should be achieved in these conflict states.

Finally, we show the number of times agents used aug-

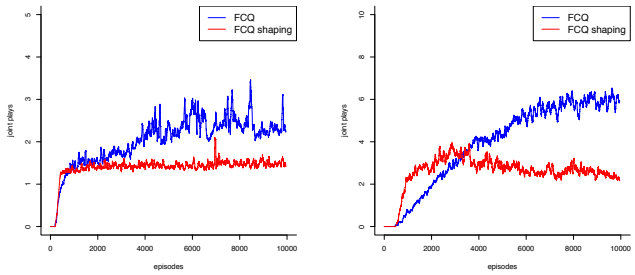


Figure 8: Average number of joint plays per episode for Grid game 2 (left) and TunnelToGoal (right)

mented state information per episode in Figure 8. FCQ-learning without a shaping function has a larger state space (which can also be seen in Tables 1 and 2). The reason why FCQ-learning without reward shaping has a larger state space is due to the fact that agents do not get feedback on their interaction immediately. The effect of having coordinated correctly is only reflected in the goal state several time steps ahead. Since it takes several learning episodes for this feedback to be backpropagated, other states are being augmented in the mean time. This also explains why FCQ-learning without shaping does not find the correct coordination policy as often as with reward shaping. The agent could have learned contradicting policies in the different augmented states. Learning to coordinate between these augmented states will take many learning episodes, especially since interactions are sparse.

6. CONCLUSION

This paper presents an extension to FCQ-learning through the incorporation of context-dependent reward shaping.

FCQ-learning is an algorithm that aims at learning a policy in an environment where multiple agents are loosely coupled. This means that agents influence each other only in certain regions of the state space. Moreover, we assume that agents are able to complete a goal independently, but coordinating is beneficial in the sense that it results in a higher payoff in the future. So FCQ-learning is capable of detecting delayed coordination problems. In states in which such problems are detected, the state space is augmented to include the state information of other agents and select actions independently using this augmented state information.

Reward shaping has already proved in both single and multi agent systems to be able to significantly speed up the learning process by giving intermediate rewards, based on some domain knowledge. We illustrated how in multi agent systems, different shaping functions can be used, dependending on the context of the agent, i.e. is it in a local state, or is it in an augmented state. This distinction allows to use the speedup achieved by learning with sparse interactions combined with the speedup obtained through reward shaping.

We evaluated our approach using a navigation task with a delayed coordination problem and observed a speedup in the convergence to a stable policy, as well as an improvement in the final policy learned by the agents. These results motivate further research in how such context sensitive reward shaping can be exploited even further in single and multi-agent systems.

7. REFERENCES

[1] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195–210, Renesse, Holland, 1996.

[2] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[3] Y.-M. De Hauwere, P. Vrancx, and A. Nowé. Learning multi-agent state space representations. In *the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 715–722, Toronto, Canada, 2010.

[4] Y.-M. De Hauwere, P. Vrancx, and A. Nowé. Solving delayed coordination problems in mas (extended abstract). In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1115–1116, Taipei, Taiwan, 2011.

[5] S. Devlin and D. Kudenko. Plan-based reward shaping for multi-agent reinforcement learning. In *The 9th European Workshop on Multi-Agent Systems*, Maastricht, The Netherlands, 2011.

[6] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 225–232, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[7] S. Devlin, D. Kudenko, and M. Grzes. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(2):251–278, 2011.

[8] A. Greenwald and K. Hall. Correlated-Q learning. In *AAAI Spring Symposium*, pages 242–249. AAAI Press, 2003.

[9] M. Grzes and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, volume 2, pages 10–22 –10–29, sept. 2008.

[10] J. Hu and M. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[11] J. Kok, P. 't Hoen, B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG05)*, pages 29–36, 2005.

[12] F. Melo and M. Veloso. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 773–780, 2009.

[13] F. Melo and M. Veloso. Local multiagent coordination in decentralised mdps with sparse interactions. Technical Report CMU-CS-10-133, School of Computer Science, Carnegie Mellon University, 2010.

[14] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[15] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 463–471, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[16] M. R. K. Ryan. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *In Proceedings of The 19th International Conference on Machine Learning*, pages 522–529. Morgan Kaufmann, 2002.

[17] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[18] J. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Journal of Machine Learning*, 16(3):185–202, 1994.

[19] K. Tumer and N. Khani. Learning from actions not taken in multiagent systems. *Advances in Complex Systems*, 12(4-5):455–473, 2009.

[20] P. Vrancx, K. Verbeeck, and A. Nowé. Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics (Part B: Cybernetics)*, 38(4):976–981, 2008.

[21] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Emergence of Honest Signaling through Reinforcement Learning

David Catteeuw
Artificial Intelligence Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
dcatteeu@vub.ac.be

Bernard Manderick
Artificial Intelligence Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
bmanderi@vub.ac.be

ABSTRACT

Until now, biologist have mostly studied under what circumstances honest signaling is stable. *Stability*, however, is not sufficient to explain the *emergence* of honest signaling. We observe that honest signaling can emerge through learning. The settings where honest signaling evolves, however, do not exactly match those where honest signaling is evolutionary stable. They do, however, match the set where honest signaling is a Pareto-optimal Nash equilibrium. As such, we provide an alternative explanation for the emergence and existence of honest signaling.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Economics, Experimentation

Keywords

signaling games, handicap principle, reinforcement learning, honest signaling

1. INTRODUCTION

We study the emergence of honest signaling in the Philip Sidney game. It was introduced by Maynard Smith [11] and has become the standard game theoretic model of the handicap principle. The *handicap principle* [18] states that under conflict of interest honest signaling can only be an equilibrium if signals are costly. The handicap principle may explain, among other phenomena, the male characteristics used for sexual selection, such as a peacock's tail. The peacock's tail is a handicap for its survival, but, as such, it is an honest signals of the male's fitness. Females can reliably infer from the size of those tails which males would make better mates, since only the fittest can afford the largest tails.

To demonstrate or contest the significance of honest signaling [12, 13], biologists are almost exclusively relying on an finding the evolutionary stable strategies of different game theoretic models such as the Philip Sidney game [4, 11]. Such explanations of honest signaling, however, do not tell us whether honest signaling will emerge, it merely states that if it would emerge (for example through mutation) it can persist. The same critique has been formulated by Huttegger and Zollman [9]. We discuss this related work in Section 5.

Here, we apply learning dynamics (Section 3) to different classes of the Philip Sidney game (Section 2) and we observe that:

- a simple adaptive process, such as reinforcement learning can lead to honest signaling,
- the settings where honest signaling is evolutionary stable do not fully match the settings were honest signaling emerges through learning,
- the learning process always converged to a Pareto-optimal Nash equilibrium.

A Nash equilibrium is Pareto-optimal if there is no other Nash equilibrium where no players are worse off and at least one player is strictly better off. We discuss the experimental results in Section 4.

2. THE PHILIP SIDNEY GAME

The Philip Sidney game, see Figure 1, is a *signaling game*, introduced by Maynard Smith [11] as an example of the handicap principle. The Philip Sidney game is a two-player extensive form game of incomplete information. The first player (Sender) can be in one of two states: healthy or needy, with probability p and $1-p$ respectively. In both cases he can either send a signal at some cost c or he can remain silent. Player 2 (Receiver) does not know the true state of Sender, but he can observe whether or not Sender signals. Furthermore, Receiver has a resource and must decide whether or not to donate his resource to Sender.

The players' chances of survival depend on the state $t \in \{\texttt{healthy}, \texttt{needy}\}$, the signal $s \in \{\texttt{signal}, \texttt{silent}\}$, and the action $a \in \{\texttt{donate}, \texttt{keep}\}$ as follows. Sender is sure to survive if he receives the resource. If Sender does not receive the resource and he is needy, then his chance of survival is 0. If he does not receive the resource and he is healthy, then his chance of survival is $V < 1$. Thus, he benefits more from receiving the resource when he is needy, than when he is healthy. The survival probabilities for Sender $v_S(t, s, a)$ are summarized by Equations 1 and 2.

$$v_S(t, s, a) = \begin{cases} (1-c)w_{t,a} & \text{if } s = \texttt{signal} \\ w_{t,a} & \text{if } s = \texttt{silent} \end{cases} \quad (1)$$

$$w = \begin{array}{c} \\ \texttt{healthy} \\ \texttt{needy} \end{array} \begin{array}{c} \begin{array}{cc} \texttt{donate} & \texttt{keep} \end{array} \\ \begin{pmatrix} 1 & V \\ 1 & 0 \end{pmatrix} \end{array} \tag{2}$$

On the other hand, if Receiver keeps the resource to himself, he is sure to survive. If he donates the resource to Sender his chances of survival $v_R(t, s, a)$ are reduced to $S < 1$, see Equation 3.

$$v_R(t, s, a) = \begin{cases} S & \text{if } a = \texttt{donate} \\ 1 & \text{if } a = \texttt{keep} \end{cases} \tag{3}$$

Clearly there is no reason for Receiver to donate his resource, except that the players may be related by some factor r, such that the utility u of each player is his survival probability *plus* r times the other player's survival probability. See Equations 4 and 5 for Sender's and Receiver's utility u_S and u_R.

$$u_S(t, s, a) = v_S(t, s, a) + r v_R(t, s, a) \tag{4}$$
$$u_R(t, s, a) = v_R(t, s, a) + r v_S(t, s, a) \tag{5}$$

Increasing the relatedness factor r reduces the conflict between the players. For $r = 1$, both players have the same utilities. This is known as Hamilton's principle of inclusive fitness [5]. The game tree and the utilities for each possible outcome of the Philip Sidney game are shown in Figure 1 and the parameters are summarized in Table 1.

	meaning
c	signal cost
p	probability of Sender being healthy
r	relatedness factor
S	Receiver's chance of survival when donating resource
V	Sender's chance of survival when healthy but not getting resource

Table 1: The parameters of the Philip Sidney game.

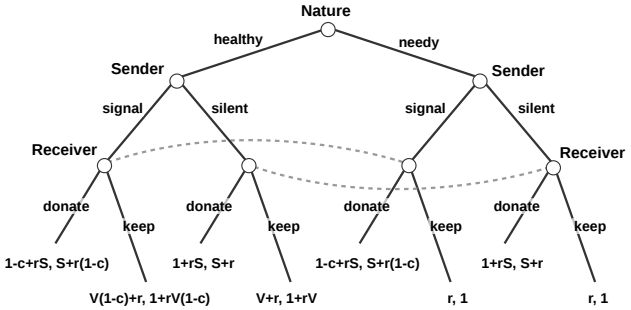


Figure 1: The Philip Sidney game. Decision nodes between which Receiver cannot distinguish are connected by a dotted line. The utilities u_S, u_R for both players are given at the terminal nodes.

The standard way to represent the behavior of players of an extensive form game is by a behavioral strategy profile. A behavioral strategy profile is a list of strategies, one for each information state – this is a set of player nodes between which the player cannot distinguish. Sender's information states are `healthy` and `needy`, Receiver's information states are `signal` and `silent`. The behavior corresponding to honest signaling: "Sender signals only when needy and Receiver donates only when signal" is denoted by $(\texttt{l}, \texttt{s}, \texttt{d}, \texttt{k})$, where the letters `l`, `s`, `d`, `k` respectively stand for the pure strategies: `silent`, `signal`, `donate`, and `keep`.

We selected a number of Philip Sidney games that allow us to illustrate the most important effects we observed during simulations of learning dynamics. The exact parameters of those games are given in Table 2. Table 3 shows an overview of the characteristics of these games and the types of equilibria that apply to honest signaling. We come back to these games in Section 4 where we discuss the experimental results.

	c	r	S	V
1	0.19	0.5	0.6	0.65
2	0.2	0.5	0.7	0.8
3	0.4	0.5	0.8	0.8
4	0.1	0.5	0.8	0.8
5	0.2	0.5	0.7	0.5
6	0.0	0.5	0.8	0.95
7	0.0	0.5	0.8	0.8
8	0.2	0.25	0.7	0.5

Table 2: Some Philip Sidney games. For each game the probability of Sender being healthy $p = 0.5$.

	costly?	conflict?	ESS?	NE?	PO?	learned?
1	yes	partial	yes	yes	yes	always
2	yes	partial	yes	yes	yes*	sometimes
3	yes	partial	yes	yes	no	never
4	yes	partial	no	yes	yes*	sometimes
5	yes	partial	no	no	no	never
6	no	none	yes	yes	yes	always
7	no	partial	no	no	no	never
8	yes	full	no	no	no	never

Table 3: For which Philip Sidney games, signaling is costly, there is a conflict of interest, honest signaling is an evolutionary stable strategy (ESS), a Nash equilibrium (NE), a Pareto-optimal Nash equilibrium (PO), and honest signaling emerged during experiments with learning dynamics. In the column of Pareto-optimal (PO) we write "yes*" when honest signaling is not the only Pareto-optimal Nash equilibrium.

3. SELFISH ADAPTIVE BEHAVIOR

Reinforcement learning algorithms are ideal as models of simple adaptive behavior of selfish individuals. A reinforcement learner repeatedly observes the state of the environment, selects an action and receives a payoff indicating the quality of his action. An individual can maintain statistics of the payoffs for all combinations of observations and actions. This way, he can learn which are the most rewarding actions in each state of the environment and hence change his behavior by selecting better actions more often.

We use Q-learning [17] with ϵ-greedy action selection, because of its ease of applicability. The parameters are easy to choose and the algorithm poses no constraints on the payoffs. Sender has a Q-value $Q_{t,s}$ for each possible state $t \in \{\texttt{healthy}, \texttt{needy}\}$ and each possible signal he can send, $s \in \{\texttt{signal}, \texttt{silent}\}$. Receiver has a Q-value $Q_{s,a}$ for each possible signal he can observe, $s \in \{\texttt{signal}, \texttt{silent}\}$, and each possible action he can take, $a \in \{\texttt{donate}, \texttt{keep}\}$.

We initialize Q-values optimistically by setting them to a value which is greater than or equal to the highest payoff of the game. This makes sure that there is sufficient exploration in the beginning of the game [15, p.40]. In the experiments we set initial Q-values to 2.

Now, for each game with outcome (t, s, a), both players update the Q-value that corresponds to the observation they made and the action they took during that game, according to the update rule 6. The other Q-values remain unchanged.

$$\begin{aligned} Q_{t,s} &\leftarrow Q_{t,s} + \alpha\,(u_S(t,s,a) - Q_{t,s}) \\ Q_{s,a} &\leftarrow Q_{s,a} + \alpha\,(u_R(t,s,a) - Q_{s,a}) \end{aligned} \tag{6}$$

In this update rule, $\alpha \in [0, 1]$ represents the learning rate. In the extreme case of $\alpha = 0$, nothing is ever learned. In the other extreme case where $\alpha = 1$, Q-values only reflect the last utility for their corresponding outcome. In fact, the Q-values represent the exponentially weighted moving average of the utilities [15, p.38]. Both players used the same learning rate $\alpha = 0.1$ in all experiments.

To balance exploration and exploitation, we used ϵ-greedy action selection: with probability $1-\epsilon$ individuals select the action with the highest Q-value (braking ties randomly) and with probability ϵ they select an action at random according to a uniform distribution. For all experiments we used a constant exploration rate $\epsilon = 0.01$.

4. EXPERIMENTS AND RESULTS

In all experiments reported here, initial Q-values were 2, the learning rate $\alpha = 0.1$ and the exploration rate $\epsilon = 0.01$ for both players. For each game setting, we did 100 runs. Per run, the game was repeated 2000 times. This is long enough, since most of the time, the behavior was already stable after a few hundred iterations.

In Table 4 we present the learned behavior for each of the game settings given in Table 2. Per game we give a list of behaviors, each of them preceded by the percentage of runs that converged to that behavior. The basic notation for behavior is described at the end of Section 2. The question mark represents a wild card and $(\frac{1}{2}\texttt{s} + \frac{1}{2}\texttt{1})$ represents a mixed strategy where Sender signals with probability 1/2 and is silent with probability 1/2.

4.1 Partial Conflict and Costly Signals

Games 1 to 5 all have a partial conflict of interest between players but costly signals, thus the handicap principle says that honest signaling can be evolutionary stable. This is the case for games 1 to 3, but not for games 4 and 5.

For the first setting, *game 1*, as expected, honest signaling always emerged through learning. A typical run is shown in Figure 2.

In the second setting, *game 2*, some runs (59%) converged to honest signaling: $(\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$. Some runs converged to another equilibrium : "Sender remains always silent and Receiver donates when silent", $(\texttt{1}, \texttt{1}, ?, \texttt{d})$. The question mark

	learned behavior
1	100% $(\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$
2	59% $(\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$, 41% $(\texttt{1}, \texttt{1}, ?, \texttt{d})$
3	100% $(\texttt{1}, \texttt{1}, \texttt{d}, \texttt{d})$
4	88% $(\texttt{1}, \texttt{1}, ?, \texttt{d})$, 12% $(\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$, $(\frac{1}{2}\texttt{s} + \frac{1}{2}\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$
5	100% $(\texttt{1}, \texttt{1}, ?, \texttt{d})$
6	56% $(\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$, 44% $(\texttt{s}, \texttt{1}, \texttt{k}, \texttt{d})$
7	51% $(\texttt{s}, \texttt{s}, \texttt{d}, ?)$, 49% $(\texttt{1}, \texttt{1}, ?, \texttt{d})$
8	100% $(\texttt{1}, ?, \texttt{k}, \texttt{k})$

Table 4: The behavioral strategy profiles to which learning converged and for which percentage of the runs for each of the Philip Sidney games in Table 2. The notation is explained in the text.

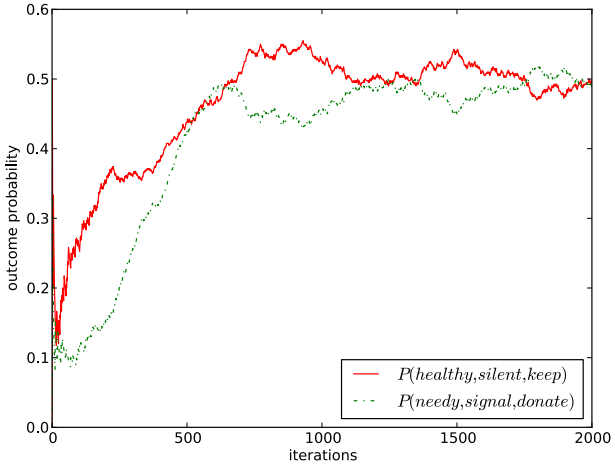


Figure 2: A typical run for game 1 showing the emergence of honest signaling. The solid red line represents the probability of seeing outcome $(\texttt{healthy}, \texttt{silent}, \texttt{keep})$ and the dotted green line the probability of seeing outcome $(\texttt{needy}, \texttt{signal}, \texttt{donate})$. Other than equilibrium outcomes quickly converged towards zero probability and are left out for clarity.

in $(\texttt{1}, \texttt{1}, ?, \texttt{d})$ denotes that the receiver can either donate or keep in case Sender signals. Note, that Sender will still signal at a rate of $\epsilon/2 = 0.005$ due to the constant exploration rate (see Section 3). The difference with game setting 1 may be explained by the fact that game 2 has two Pareto-optimal Nash equilibria: $(\texttt{1}, \texttt{s}, \texttt{d}, \texttt{k})$ and $(\texttt{1}, \texttt{1}, ?, \texttt{d})$. These correspond to the learned behavior. In contrast, in game setting 1 honest signaling is the unique Pareto-optimal Nash equilibrium and the only behavior learned.

In *game 3*, honest signaling is evolutionary stable (and by definition also a Nash equilibrium). The learning dynamics, however, never converged to the honest signaling equilibrium, instead it converged to "Sender remains always silent and Receiver always donates", $(\texttt{1}, \texttt{1}, \texttt{d}, \texttt{d})$, which is a Pareto-optimal Nash equilibrium for this game. So, as before, the learning process converged to a Pareto-optimal Nash equilibrium.

Game 4 shows the opposite effect of game 3. Here, honest signaling is not evolutionary stable although it is a (Pareto-

optimal) Nash equilibrium. There is another Pareto-optimal Nash equilibrium: "Sender is always silent and Receiver donates when Sender is silent", $(\mathtt{l}, \mathtt{l}, ?, \mathtt{d})$. The learning dynamics evolved to this equilibrium in 88% of the runs. In a smaller number of runs, 12 out of 100, we observed periods of honest signaling $(\mathtt{l}, \mathtt{s}, \mathtt{d}, \mathtt{k})$ interleaved by periods where Sender, when healthy, is bluffing half of the time: $(\frac{1}{2}\mathtt{s} + \frac{1}{2}\mathtt{l}, \mathtt{s}, \mathtt{d}, \mathtt{k})$. This last behavior is not Pareto-optimal in itself but the combination of $(\mathtt{l}, \mathtt{s}, \mathtt{d}, \mathtt{k})$ and $(\frac{1}{2}\mathtt{s} + \frac{1}{2}\mathtt{l}, \mathtt{s}, \mathtt{d}, \mathtt{k})$ is, as long as Sender signals at most 1/3 of the time when he is healthy.

In *game 5* honest signaling is not even a Nash equilibrium. Again, the learning process converged to a Pareto-optimal Nash equilibrium: "Sender is always silent and Receiver donates when Sender is silent", $(\mathtt{l}, \mathtt{l}, ?, \mathtt{d})$.

4.2 Cost-free Signals

Game 6 has cost-free signals and no conflict of interest. Honest signaling is evolutionary stable and thus also a Nash equilibrium. Since there is no cost for signaling, both $(\mathtt{l}, \mathtt{s}, \mathtt{d}, \mathtt{k})$ and $(\mathtt{s}, \mathtt{l}, \mathtt{k}, \mathtt{d})$ come down to the same thing. These behaviors simply have the meaning of `silent` and `signal` switched. The learning dynamics reached both equilibria in a more or less equal amount of runs. In fact, this example shows how arbitrary signals can acquire meaning through learning processes. This was already studied extensively [14], see the related work in Section 5.

Game 7 also has cost-free signals but with partial conflicting interest. According to the handicap principle honest signaling cannot be evolutionary stable. Honest signaling is not a Nash equilibrium either. There are, however, many equilibria were partial information is transferred. Some of them were identified by Huttegger and Zollman [9], see Section 5. In our case the learning dynamics always converged to any of the pure strategy equilibria: $(\mathtt{s}, \mathtt{s}, \mathtt{d}, ?)$ and $(\mathtt{l}, \mathtt{l}, ?, \mathtt{d})$, both in about half of the runs. All Nash equilibria have the same payoffs $(1.4, 1.3)$, so both strategies profiles are also Pareto-optimal Nash equilibria.

4.3 Full Conflict

When there is a full conflict, Receiver prefers to keep his resource whether Sender is healthy or not, and Sender would prefer Receiver to donate at all times. In such settings, there are no Nash equilibria (and hence no evolutionary stable strategies) where Receiver donates. As a consequence, it does not really matter whether Sender is signaling honestly or not. Moreover, when signals are costly, Sender is better off by remaining silent when he is healthy. In *game 8*, for example, we observed Sender learned to signal honestly. Receiver, however, never bothered about the signal, $(\mathtt{l}, \mathtt{s}, \mathtt{k}, \mathtt{k})$.

In *summary*, the learning process always converged to a Pareto-optimal Nash equilibrium. So, for this learning process, honest signaling can emerge, but only when it is a Pareto-optimal Nash equilibrium. We were able to identify settings in which honest signaling is evolutionary stable but not Pareto-optimal, for example game 3. We also identified settings were honest signaling is not evolutionary stable but could emerge through learning, for example game 4.

5. RELATED WORK

Huttegger and Zollman [9] study evolutionary dynamics in the Philip Sidney game, and contrast their results to those obtained by calculating the evolutionary stable strategies. They find that in many cases honest signaling has far smaller basins of attraction than other equilibria. This may mean that honest signaling is far less significant as is suggested by the existence of evolutionary stable strategies. Indeed, there is an ongoing debate about how widespread honest signaling really is [12, 13]. Our work is exploring the same question and we argue that analyzing dynamics of adaptive processes leading to honest signaling can provide better insights.

Closely related to the emergence of honest signaling is the emergence of signaling itself. The emergence of signaling in a game theoretic setting was first studied by Lewis [10]. His signaling games represent a communication problem where there is neither cost for signaling, nor conflict. The problem is fully cooperative and the goal for the players is to come up with a convection for the meaning of the signals without any pre-existent means of communication. Recently, both evolutionary dynamics [7, 8] and learning dynamics [1, 2, 3, 14] received more attention and it was shown that perfect communication can evolve from initially random behavior through simple adaptive processes. While we concentrate on honest signaling in this text, we could indeed observe that initially arbitrary signals can acquire meaning through repeated interaction between learning processes. Games 6 and 7 are signaling games where signals are cost-free. We observed that the meaning acquired by the signals was swapped in about half of the runs, see Table 4 and Section 4.2.

6. CONCLUSION

In biology honest signaling has mostly been studied by doing a static analysis of signaling games, that is verifying whether honest signaling is an evolutionary stable strategy. In the process, they have been ignoring other equilibria which may be equally or even more important. Both the work of Huttegger and Zollman [9], using evolutionary dynamics, and our work, using learning dynamics, show that honest signaling can emerge from initially random behavior through adaptive processes and more importantly it shows the existence of settings where honest signaling is an equilibrium, but where it is not (necessarily) the outcome of the dynamic process. We also observed the opposite, honest signaling can emerge through learning in settings where it is not evolutionary stable.

Note that there is an important difference between evolutionary stability and evolutionary dynamics on the one hand and learning dynamics on the other hand. Evolutionary stability and dynamics considers how and whether characteristics due to genetic mutations can spread in a *population*. Learning dynamics considers whether *individuals* can acquire certain characteristics. Learned behavior can spread through the population through imitation processes (individuals imitate the behavior of fitter individuals), but it can also influence genetic evolution since natural selection may favor individuals that can acquire the characteristic through their lifetime [6]. As such the genetic evolution is guided by the learning. In the limit the characteristic can even be "canalized" and does not need to be learned at all [16].

This allows us to question whether analysis of evolutionary stability is a good or the only good methodology to study the emergence of honest signaling. Other equilibrium concepts exist, but it is not clear which one should apply. Learning dynamics provides a simple and intuitive alternative.

One may wonder, in how far the choice of learning model (Section 3) influenced our results. We argue that the choice is a reasonable one. Similar models of reinforcement learning have been applied to signaling games [2, 3]. All of them are very limited in terms of computational and informational requirements as mentioned before. They keep track of statistics summarizing the quality of their actions and keep a balance between exploiting the best actions and exploring the others. Despite their simplicity, these models have characteristics also seen in human and animal behavior.

7. REFERENCES

[1] R. Argiento, R. Pemantle, B. Skyrms, and S. Volkov. Learning to signal: Analysis of a micro-level reinforcement model. *Stochastic Processes and their Applications*, 119(2):373–390, Feb. 2009.

[2] J. A. Barrett and K. J. S. Zollman. The role of forgetting in the evolution and learning of language. *Journal of Experimental & Theoretical Artificial Intelligence*, 21(4):293–309, Dec. 2009.

[3] D. Catteeuw, J. De Beule, and B. Manderick. Roth-Erev Learning in Signaling and Language Games. In P. De Causmaecker, J. Maervoet, T. Messelis, K. Verbeeck, and T. Vermeulen, editors, *Proceedings of the Benelux Conference on Artificial Intelligence*, pages 65–74, Ghent, Belgium, 2011.

[4] A. Grafen. Biological signals as handicaps. *Journal of theoretical biology*, 144(4):517–46, June 1990.

[5] W. D. Hamilton. The genetical evolution of social behaviour. *Journal of Theoretical Biology*, 7(1):1–52, 1964.

[6] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex systems*, 1(1):495–502, 1987.

[7] J. Hofbauer and S. M. Huttegger. Feasibility of communication in binary signaling games. *Journal of theoretical biology*, 254(4):843–849, Oct. 2008.

[8] S. M. Huttegger, B. Skyrms, R. Smead, and K. J. S. Zollman. Evolutionary dynamics of Lewis signaling games: signaling systems vs. partial pooling. *Synthese*, 172(1):177–191, Feb. 2009.

[9] S. M. Huttegger and K. J. S. Zollman. Dynamic stability and basins of attraction in the Sir Philip Sidney game. *Proceedings of the Royal Society B: Biological Sciences*, 277(1689):1915–22, June 2010.

[10] D. K. Lewis. *Convention: A Philosophical Study*. Harvard University Press, Cambridge, 1969.

[11] J. Maynard Smith. Honest signalling: The Philip Sidney game. *Animal Behaviour*, 42:1034–1035, 1991.

[12] J. Maynard Smith and D. Harper. *Animal signals*. Oxford University Press, Oxford, UK, 2003.

[13] W. A. Searcy and S. Nowicki. *The evolution of animal communication: reliability and deception in signaling systems*. Princeton University Press, Princeton, NJ, 2005.

[14] B. Skyrms. *Signaling: Evolution, Learning and Information*. Oxford University Press, New York, 2010.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[16] C. H. Waddington. Canalization of Development and the Inheritance of Acquired Characters. *Nature*, 150:563–565, 1942.

[17] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[18] A. Zahavi. Mate selection - a selection for a handicap. *Journal of Theoretical Biology*, 53:205–214, Sept. 1975.

nMetaQ: An n-agent Reinforcement Learning Algorithm Based on Meta Equilibrium

Yujing Hu, Zhaonan Sun, Xingguo Chen, Yang Gao*
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
{huyujing.yujing.hu, sunzhaonan, chenxgspring}@gmail.com, gaoy@nju.edu.cn

Ruili Wang
School of Engineering of Advanced Technology
Massey University
Palmerston North, New Zealand
R.Wang@massey.ac.nz

ABSTRACT

Multi-agent reinforcement learning (MARL) has been widely studied over the last years. In MARL, one approach is to combine game theory with reinforcement learning (RL) to help with selecting actions and updating policies. Markov games are adopted in this approach as the framework and policies are learnt based on equilibrium theories. Several algorithms have been proposed based on this idea, such as minimax-Q, NashQ, FFQ, Correlated-Q and MetaQ. However, some of these algorithms are proposed only for 2-agent problems while the others have difficulty in dealing with problems with more than 2 agents. Since many tasks involve more than 2 agents in the real world, an algorithm which can deal with n-agent ($n > 2$) problems is needed. In this paper, we propose nMetaQ based on MetaQ. nMetaQ can be applied to a multi-agent environment that has more than 2 agents. Experimental results demonstrate the empirical convergence of nMetaQ and show its satisfactory adaptive performance. The most important advantage of nMetaQ is that it can work efficiently and effectively in an n-agent ($n > 2$) environment while previous methods may not.

Categories and Subject Descriptors

I.2.m [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms

Keywords

Multi-agent Reinforcement Learning, Game Theory, Metagame, nMetaQ

1. INTRODUCTION

There has been an increasing interest in reinforcement learning since the late 1980s in the fields of Machine Learning (ML) and Artificial Intelligence (AI). RL is a learning technique in which agents learn from interactions with an environment without specifying how a task is to be completed [9]. Among the large numbers of RL algorithms proposed so far, Q-learning [25] has been regarded as one of the most important breakthroughs in RL [23] and its learning framework is adopted by some well-known RL algorithms [13, 16].

Multi-agent reinforcement learning can be seen as the extension of RL to the multi-agent domain. Recently, it has gained growing attention because most tasks in the real world need to be completed by more than one agent. Single-agent RL methods can be used to solve multi-agent learning problems by treating the other agents in a system as part of the environment. However, this may not be an effective way since the environment is no longer stationary with the existence of the other agents. A more rational way is to assume that all agents are taking part in one game, in which some agents may have consistent goals while the others may not. From the technical point of view, this has taken the community from the realm of Markov Decision Problems to the realm of game theory, and in particular Markov games (or stochastic games) [21].

Littman's minimax-Q algorithm [11] is widely considered as the first algorithm which introduces game theory into RL. But, the NashQ algorithm [7, 8] presented by Hu and Wellman is mentioned more frequently since it extends Littman's work to a broader framework. Similar algorithms like FFQ [12] and CE-Q [4] provide more alternatives for solving multi-agent tasks. However, these algorithms have some limitations. For instances: minimax-Q and FFQ suit a few special cases; NashQ has expensive cost of computation and its effectiveness can not be guaranteed well due to the fact that a Nash equilibrium may be Pareto dominated by a non-equilibrium solution.

To address these issues mentioned above, a novel algorithm MetaQ was proposed in a recent paper [2] based on meta equilibrium. MetaQ performs bettern than other game theory-based MARL algorithms (including NashQ, FFQ, CE-Q) due to three properties of meta equilibrium: (i) always existing; (ii) being not Pareto dominated by non-equilibrium solutions; (iii) being computed very efficiently. Furthermore,

*Corresponding Author

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

these properties not only exist in 2-agent situations, but also can be retained in n-agent ($n > 2$)[1] ones. In our opinion, this is the most prominent advantage of a meta equilibrium compared with other types of equilibria. However, MetaQ can only deal with 2-agent problems. In order to be used widely, MetaQ needs to be extended. In this paper, we propose nMetaQ, which can be seen as an extension of MetaQ from 2-agent problems to n-agent ones. Our work has two major contributions: (i) an algorithm for computing meta equilibria in general n-agent ($n \geq 2$) games; (ii) the effectiveness of nMetaQ algorithm is verified to be able to solve n-agent ($n \geq 2$) Markov games by our experiments.

The rest of this paper is organized as follows. In the next section, we explain some basic concepts in RL and game theory and introduce some related work in MARL. In Section 3, metagame theory including some important theorems and the algorithm for computing meta equilibria is introduced, followed by the description of our algorithm, nMetaQ, in Section 4. We give our experimental results in Section 5 and provide some discussion in Section 6. Finally, we summarize our work in Section 7.

2. PRELIMINARIES

In multi-agent settings, learning is complicated due to the fact that multiple agents may be learning simultaneously. This can thus make the environment nonstationary for a learner [19]. Obviously, game theory can provide us a powerful tool for modeling the multi-agent systems. We start this section by reviewing some basic concepts in RL, game theory and some related work. Then, a general framework which generalizes all game theory-based RL algorithms is given.

2.1 Markov Decision Process and Reinforcement Learning

A Markov Decision Process (MDP) has the capability of giving an accurate description of single-agent decision problems. In fact, almost all RL algorithms adopt MDPs or one of its variants as their fundamental frameworks. An MDP can be defined as follows:

DEFINITION 1. *A Markov Decision Process is a tuple $< S, A, r, p >$, where S is the discrete state space; A is the action space of the agent; $r : S \times A \to \mathbf{R}$ is the reward function, and $p : S \times A \times S \to [0, 1]$ is the transition function.*

The objective of an agent in an MDP is to find an optimal policy π^*, which maximizes the *return* R in the long run:

$$\pi^* = \arg\max_{\pi} R^{\pi} = \arg\max_{\pi} E_{\pi}\Big\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\Big\}, \qquad (1)$$

where R^{π} stands for the long-time return under policy π and is defined by the expected value of the sum of the discounted rewards; $E_{\pi}\{\}$ denotes the expected value with respect to π; r_{t+k+1} is the reward received at the k-th step after the current time t; t can be any time step, and γ is the *discount rate* that determines how much contribution the future rewards can make to the present value.

Generally, the problem of finding the optimal policy π^* is identical to that of computing the *optimal state-value function* V^* or the *optimal action-value function* Q^*. According to the Bellman optimality equation [1], V^* can be computed as:

$$\begin{aligned} V^*(s) &= E_{\pi^*}\Big\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\Big\} \\ &= \max_a E_{\pi^*}\Big\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\Big\} \\ &= \max_a \sum_{s'} p(s, a, s')[r(s, a, s') + \gamma V^*(s')], \end{aligned} \qquad (2)$$

and a similar equation for computing Q^* is:

$$Q^*(s, a) = \sum_{s'} p(s, a, s')[r(s, a, s') + \gamma \max_{a'} Q^*(s', a')]. \qquad (3)$$

Owing to the expensive cost for solving the Bellman optimality equation, a more practical solution is to update the value function and the corresponding policy repeatedly in an iterative search process, through which both the value function and the policy will finally converge to their optimal values. Precisely, this is the basic idea that reinforcement learning relies upon.

Q-learning is one of the most important RL algorithms because it has a deep impact on the development of RL. We briefly introduce it here since it is highly related to our work. More information about RL can be found in [9] and [23]. Q-learning is an off-policy TD control algorithm [22]. It focuses on the update of the action-value function Q, of which the updating rule is quite a simple one:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)], \qquad (4)$$

where the maximum Q-value of the next s' is backed up to the Q-value of the current state-action pair (s, a). By this rule, the action-value function Q directly approximates Q^* independent of the policy being followed. As a consequence, this significantly simplified the analysis of the algorithm and enabled early convergence proofs [26]. The pseudo code of Q-learning is shown in Table 1.

Table 1: the pseudo code of Q-learning

1. Initialize $Q(s, a)$ arbitrarily;
2. Repeat (for each episode):
3. **Initialize** state s;
4. **Repeat** (for each step):
5. Choose a from s using policy derived from Q (e.g., ϵ-greedy);
6. Take action a, observe r and s';
7. Update the Q value as: $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$;
8. $s \leftarrow s'$;
9. **until** s is the terminal state;
10. until no more episodes;

2.2 One-stage Game and Markov Games

One of the fundamental concepts in game theory is *one-stage game*:

DEFINITION 2. *An n-agent ($n \geq 2$) one-stage game Γ is a tuple $< N, A^1, \ldots, A^n, U^1, \ldots, U^n >$, where N is the set of agents; A^i is the action space of agent i(i=1,...,n). Denote $A = \prod_{i=1}^{n} A^i$ as the joint action space of the agents, and $U^i : A \to \mathbf{R}$ is the utility function for agent i.*

[1] In the remaining part of this paper, the n in "n-agent" is larger than 2 by default.

In this definition, we use *agent* and *action* to replace the standard game theoretic terminology *player* and *strategy*, respectively, for the purpose of contextual consistency. For solving a game, an important concept is Nash equilibrium [15]:

DEFINITION 3. *In an n-agent ($n \geq 2$) one-stage game Γ, a joint action $\vec{a} = (a_*^1, ..., a_*^n)$ is a pure strategy Nash equilibrium if and only if for any $i \in N$, $U^i(\vec{a}) \geq U^i(a_*^1, ..., a_*^{i-1}, a', a_*^{i+1}, ..., a_*^n)$ holds for any $a' \in A^i$.*

That is to say, in a Nash equilibrium, each agent's action is the best response to the other agents' joint action.

Markov games can be seen as an extension of both the standard one-stage game and MDP. An n-agent Markov game is defined as follows:

DEFINITION 4. *An n-agent ($n \geq 2$) Markov game is a tuple $< N, S, A^1, \ldots, A^n, r^1, \ldots, r^n, p >$, where N is the set of agents; S stands for the state space of the environment; A^i is the action space of agent i(i=1,...,n); $r^i : S \times A^1 \times \cdots \times A^n \rightarrow \mathbf{R}$ is the payoff function for agent i; $p : S \times A^1 \times \cdots \times A^n \times S \rightarrow [0, 1]$ is the transition function of the environment.*

Markov games are a general form of MDP and conversely MDP is a special case of Markov games with just one agent. In each state s, each agent chooses an action according to its action-value function Q. We denote the Q function of agent i by Q^i, and the tuple $< N, A^1, \ldots, A^n, Q^1(s), \ldots, Q^n(s) >$ can be regarded as a one-stage game in state s. Therefore, Markov games link the one-stage game and MDP together.

2.3 Game Theory-based MARL Algorithms

2.3.1 Minmax-Q

Littman [11] proposed the first game theory-based MARL algorithm, minimax-Q. Minimax-Q focuses on 2-agent zero-sum games. The Q function is extended from $Q(s, a)$ to $Q(s, a, o)$ by taking the opponent's action o into consideration. In this algorithm, the agent keeps a conservative expectation of its actions. The state-value function is defined by:

$$V(s) = \max_{\pi} \min_{o} \sum_{a} Q(s, a, o)\pi(s, a), \tag{5}$$

and the updating rule for Q function is changed to:

$$Q_{t+1}(s, a, o) \leftarrow (1 - \alpha)Q_t(s, a, o) + \alpha[r_t + \gamma V_t(s')]. \tag{6}$$

The limitation of minimax-Q is that it can only deal with 2-agent zero-sum games.

2.3.2 NashQ

Based on the concept of Nash equilibrium, Hu and Wellman [7, 8] proposed NashQ learning algorithm to solve more general problems. Like minimax-Q, NashQ still adopts the framework of Q-learning, the difference is that NashQ needs to maintain the Q-table $Q^i (i = 1, ..., n)$ for each agent. The updating rule for each Q-table is:

$$Q_{t+1}^i(s, \vec{a}) \leftarrow (1 - \alpha)Q_t^i(s, \vec{a}) + \alpha[r_t^i + \gamma NashQ_t^i(s')], \tag{7}$$

where $\vec{a}$ represents a joint action taken by all the agents, and $NashQ_t^i(s')$ is the corresponding Q-value of agent i for the selected Nash equilibrium in the game $(Q_t^1(s'), \ldots, Q_t^n(s'))$.

2.3.3 FFQ

The convergence proof of NashQ makes it a milestone in MARL research [8], but its assumptions are restrictive since these assumptions put limits on the types of games as well as on the intermediate results of learning. For this reason, Littman [12] proposed friend-or-foe Q-learning (FFQ) for two special Nash equilibria: adversarial equilibrium and coordination equilibrium. Each of them corresponds to the two opponent types "friend" and "foe", respectively. For simplicity, we use the 2-agent version of FFQ for discussion. From the perspective of agent 1, the updating rule is Equation (7) with:

$$NashQ^1(s) = \max_{a_1 \in A^1, a_2 \in A^2} Q^1(s, a_1, a_2) \tag{8}$$

if agent 2 is considered a friend or

$$NashQ^1(s) = \max_{\pi^1} \min_{a_2 \in A^2} \sum_{a_1 \in A^1} Q^1(s, a_1, a_2)\pi^1(s, a_1) \tag{9}$$

if the opponent is considered a foe. Compared to NashQ, FFQ provides stronger convergence guarantees.

2.3.4 Correlated-Q

A Nash equilibrium is a profile of independent actions (or strategies), whereas a correlated equilibrium is more general than a Nash equilibrium in that it allows for dependencies among agents' actions. Greenwald and Hall [4] adopted the concept of correlated equilibrium and presented the Correlated-Q learning (CE-Q) algorithm.

DEFINITION 5. *Given a one-stage game Γ, a correlated equilibrium is a probabilistic distribution p over the joint action space A if for all $i \in N$ and for all $a_i, a_i' \in A^i$:*

$$\begin{aligned} &\sum_{a_{-i} \in A^{-i}} p(a_i, a_{-i})U^i(a_i, a_{-i}) \geq \\ &\sum_{a_{-i} \in A^{-i}} p(a_i, a_{-i})U^i(a_i', a_{-i}). \end{aligned} \tag{10}$$

In this definition, a_{-i} represents the joint action taken by the other agents except agent i and A^{-i} is the corresponding joint action space. The set of CE is a convex polytope and can be computed via linear programming. To resolve the problem of selecting from multiple equilibria, Greenwald and Hall [4] defined utilitarian, egalitarian, plutocratic, and dictatorial CE-Q learning based on four equilibrium selection mechanisms.

2.4 A General Framework

Those algorithms mentioned above share the same framework as shown in Table 2, where Θ and Φ are two operators related to a certain kind of equilibrium. Θ represents computing equilibrium actions and Φ_i is agent i's Q-value corresponding to the equilibrium actions. It is worth noting that in minimax-Q and FFQ, agents only need to maintain their own Q-tables while those in NashQ or CE-Q also need to keep the other agents' Q-tables for computing equilibria in each state.

There are no doubts that all these algorithms make great contributions to the development of MARL. However, in our opinion, they all have some deficiencies when being applied to the environments where there are more than two agents. Obviously, minimax-Q only fits the 2-agent zero-sum Markov games and thus it has a limited practical val-

Table 2: the general framework of game theory-based MARL algorithms

1. Initialize $Q^i(s, a_i, a_{-i})$ arbitrarily;
2. Repeat (for each episode):
3. **Initialize** state s;
4. **Repeat** (for each step):
5. Select action $\vec{a} = \Theta(Q^i(s), Q^{-i}(s))$;
6. Observe and receive the experience $(s, \vec{a}, r, s')$;
7. Update the Q value as:
$Q^i(s, \vec{a}) \leftarrow (1-\alpha)Q^i(s, \vec{a}) + \alpha[r + \gamma\Phi_i(s')]$;
8. $s \leftarrow s'$;
9. **until** s is the terminal state;
10. until no more episodes;

ue. As for NashQ, there are mainly three problems. Firstly, algorithms for finding a Nash equilibrium (e.g., Lemke-Howson [10], PNS [17], MIP [20]) in 2-agent games have the exponential worst-case running time. With more than two agents, the cost will be very expensive and still there is no efficient way to find Nash equilibrium in n-agent games. Secondly, a Nash equilibrium could probably be Pareto dominated (e.g., the prisoners' dilemma shown in Figure 1(a)). Thirdly, mixed strategy Nash equilibria may not be suitable for MARL because in the learning process, a game with certain Q-values in each state can hardly be repeated. FFQ has lower computational cost and can work in n-agent environments. But, it only suits a few special cases, where agents treat each other as either a friend or a foe.

With the help of metagame theory, most of the problems mentioned above can be addressed [2]. In the next section, we will state some basic concepts in metagame theory.

3. METAGAME THEORY

3.1 Metagame

The concept of *metagame* was first proposed by Howard [6]. A metagame is a hypothetical game derived from the original game by assuming that one agent can know the other agents' actions first and its action is a reaction function of other agents' joint action. Here we give the definition of metagame in a 2-agent situation. For an n-agent situation, it can be inferred easily.

DEFINITION 6. *For a two-agent one-stage game* $\Gamma = < A^1, A^2, U^1, U^2 >$, *metagame* 1Γ *can be defined as a tuple* $< A^1_{(1)}, A^2_{(1)}, U^1_{(1)}, U^2_{(1)} >$ *where* $A^1_{(1)} = (A^1)^{A^2}$, $A^2_{(1)} = A^2$. *For* $(f_1, a_2) \in A_{(1)}$, $U^k_{(1)}(f_1, a_2) = U^k(f_1(a_2), a_2)$ $(k = 1, 2)$. *And metagame* 2Γ *is also a similar tuple* $< A^1_{(2)}, A^2_{(2)}, U^1_{(2)}, U^2_{(2)} >$ *where* $A^1_{(2)} = A^1$, $A^2_{(2)} = (A^2)^{A^1}$, *and* $U^k_{(2)}(a_1, f_2) = U^k(a_1, f_2(a_1))$ $(k = 1, 2)$. *Here we omit the agent set* N *in this definition.*

For example, we can extend the prisoners' dilemma to its 1Γ form, which is shown in Figure 1(b). In 1Γ, B keeps its action space as in the original game. For A, there are four reactive strategies for the action chosen by B:

f1 : Always confess no matter what action B takes.

f2 : Always deny no matter what action B takes.

(A,B)	Confess	Deny
Confess	(-9,-9)	(0,-10)
Deny	(-10,0)	(-1,-1)

(a) original prisoners' dilemma

(A,B)	Confess	Deny
f1	(-9,-9)	(0,-10)
f2	(-10,0)	(-1,-1)
f3	(-9,-9)	(-1,-1)
f4	(-10,0)	(0,-10)

(b) 1Γ of prisoners' dilemma

Figure 1: Prisoners' Dilemma

f3 : Confess when B confesses and deny when it denies.

f4 : Deny when B confesses and confess when it denies.

1Γ and 2Γ are called one-order metagame. Recursively, by regarding an n-order $(n \geq 1)$ metagame as the original game, $(n+1)$-order metagames can be derived. Thus, we can define two-order metagames (e.g., 12Γ, 21Γ) and metagames with higher orders (e.g., 121Γ, 2112Γ). However, Howard [6] has proved that 12Γ and 21Γ are enough for 2-agent games, and further extensions based on them are meaningless. Therefore, 12Γ and 21Γ are called *complete game*. For general n-agent $(n \geq 2)$ games, complete game can be defined as follows:

DEFINITION 7. *For an* n*-agent* $(n \geq 2)$ *game* Γ, *metagame* $\eta\Gamma$ *is a complete game if the prefix* η *is a permutation of 1,2,...,n.*

Thus, when there are three agents in the game, the set of complete games includes: 123Γ, 132Γ, 213Γ, 231Γ, 312Γ and 321Γ. As a matter of fact, we can easily infer that there are $n!$ complete games for an n-agent $(n \geq 2)$ game.

3.2 Meta Equilibrium

Another core concept in metagame theory is *meta equilibrium*. We state its definition and some important properties in this subsection.

DEFINITION 8. *For a game* Γ *and one of the agents' joint actions* $\vec{a}$, *if there exist a metagame* $\eta\Gamma$ *and its pure strategy Nash equilibrium* $\vec{x}$ *that satisfy:* $\phi(\vec{x}) = \vec{a}$, *then we call* $\vec{a}$ *the meta equilibrium of* Γ.

DEFINITION 9. *A meta equilibrium* $\vec{a}$ *is said to be symmetric if it can be derived from all complete games.*

In the definitions above, ϕ is the operator for computing the real actions. The set of all meta equilibria of Γ is denoted by $E(\Gamma)$. Especially, the set of all the meta equilibria deduced from $\eta\Gamma$ is denoted by $\hat{E}(\eta\Gamma)$. The set of symmetric meta equilibria is denoted by $SymE(\Gamma)$

In contrast to Nash equilibrium, the following theorems[2] can guarantee the effectiveness and rationality of meta equilibrium:

THEOREM 1. *Every game has at least one meta equilibrium.*

[2]All proofs of these theorems are omitted here. They will be given in an extensive version of this paper.

THEOREM 2. *In any game, if the joint action $\vec{a}$ is a pure strategy Nash equilibrium, then it is also a symmetric meta equilibrium.*

THEOREM 3. *In any game, if a joint action $\vec{a}_1$ is a Nash equilibrium which is Pareto dominated by another joint action $\vec{a}_2$, then $\vec{a}_2$ is a symmetric meta equilibrium.*

THEOREM 4. *In any game, if a joint action $\vec{a}_1$ is a meta equilibrium which is Pareto dominated by another joint action $\vec{a}_2$, then $\vec{a}_2$ is also a meta equilibrium.*

From Theorems 2, 3 and 4, we can see that a meta equilibrium and its Pareto dominated actions (if exist) are in the same set. We have enough reasons to believe that the optimal actions are in $SymE(\Gamma)$ or $E(\Gamma)$. We should consider $SymE(\Gamma)$ first because symmetric meta equilibria satisfy all agents' rational expectations. However, $SymE(\Gamma)$ is empty sometimes. But, this does not mean that there are no other solutions. The existence of meta equilibrium is guaranteed by Theorem 1. At least, a meta equilibrium could meet some agents' expectations with the other agents making concessions. Obviously, this is what often happens in our real life. Although it may not be a symmetric one, we still believe that it has effectiveness.

3.3 Computing Meta Equilibria

To find a meta equilibrium in a game is much easier than to find a Nash equilibrium. In fact, there is no need to extend the game to one of its metagames and then find a Nash equilibrium in the metagame. Thomas [24] proved a sufficient and necessary condition of meta equilibrium, which can be considered as an identical definition of meta equilibrium. It is this definition that provides us an effective way to compute meta equilibria. The definition can be described as follows:

DEFINITION 10. *For an n-agent ($n \geq 2$) game Γ, a joint action $\vec{a}$ is called a meta equilibrium from a metagame $k_1k_2 \cdots k_r\Gamma$ if for any i there holds:*

$$U^i(\vec{a}) \geq \min_{a_{P_i}} \max_{a_i} \min_{a_{F_i}} U^i(a_{P_i}, a_i, a_{F_i}, a_{N_i}), \qquad (11)$$

where P_i is the set of agents list before sign i in the prefix $k_1k_2 \cdots k_r$, F_i is the set of agents listed after sign i and N_i is the set of agents which do not appear in the prefix. If i is not listed in the prefix, let P_i be $\emptyset$ and F_i be the set of all agents listed in the prefix.

Take a 3-agent game Γ as an example, to compute the set $\hat{E}(321\Gamma)$, we need to find the joint action $\vec{a} = (a_1, a_2, a_3)$ which satisfies:

$$\begin{cases} U^1(a_1,a_2,a_3) \geq \min_{a'_3} \min_{a'_2} \max_{a'_1} U^1(a'_1,a'_2,a'_3), \\ U^2(a_1,a_2,a_3) \geq \min_{a'_3} \max_{a'_2} \min_{a'_1} U^2(a'_1,a'_2,a'_3), \\ U^3(a_1,a_2,a_3) \geq \max_{a'_3} \min_{a'_2} \min_{a'_1} U^3(a'_1,a'_2,a'_3). \end{cases}$$

Then for $\hat{E}(123\Gamma)$, $\vec{a}$ should satisfy:

$$\begin{cases} U^1(a_1,a_2,a_3) \geq \max_{a'_1} \min_{a'_2} \min_{a'_3} U^1(a'_1,a'_2,a'_3), \\ U^2(a_1,a_2,a_3) \geq \min_{a'_1} \max_{a'_2} \min_{a'_3} U^2(a'_1,a'_2,a'_3), \\ U^3(a_1,a_2,a_3) \geq \min_{a'_1} \min_{a'_2} \max_{a'_3} U^3(a'_1,a'_2,a'_3). \end{cases}$$

In the same way, we are able to compute $\hat{E}(132\Gamma)$, $\hat{E}(213\Gamma)$, $\hat{E}(231\Gamma)$ and $\hat{E}(312\Gamma)$. Furthermore, by computing the intersection of the six sets, it could be found that $\vec{a}$ is a symmetric meta equilibrium if it satisfies:

$$\begin{cases} U^1(a_1,a_2,a_3) \geq \min_{a'_2} \min_{a'_3} \max_{a'_1} U^1(a'_1,a'_2,a'_3), \\ U^2(a_1,a_2,a_3) \geq \min_{a'_1} \min_{a'_3} \max_{a'_2} U^2(a'_1,a'_2,a'_3), \\ U^3(a_1,a_2,a_3) \geq \min_{a'_2} \min_{a'_1} \max_{a'_3} U^3(a'_1,a'_2,a'_3). \end{cases} \qquad (12)$$

However, it is not the best way to compute $SymE(\Gamma)$ like this. Note that there are $n!$ complete games for an n-agent game. If n is too big, the computational cost will be too expensive. Obviously, Equation (12) is quite simple and beautiful. Also, there is a general form of this equation. The following theorem can provide us a more efficient way to compute the set of symmetric meta equilibria:

THEOREM 5. *For any n-agent ($n \geq 2$) game, a joint action $\vec{a} = (a_1, a_2, \ldots, a_n)$ is a symmetric meta equilibrium if and only if for any i ($1 \leq i \leq n$), there holds:*

$$U^i(a_1, a_2, \ldots, a_n) \geq \min_{a_{-i}} \max_{a_i} U^i(a_i, a_{-i}). \qquad (13)$$

We give an algorithmic form for this theorem in Table 3. In this algorithm, no complex computational operations are involved. All we need is some basic operations like comparing or seeking an intersection of two sets. Non-symmetric meta equilibria can be computed in a similar way by using Equation (11) to replace Equation (13) in the third step. Therefore, the meta equilibria of a game can be computed very efficiently.

Table 3: an algorithm for computing the set of symmetric meta equilibria

Precondition: a game $\Gamma =< N, A^1, \ldots, A^n, U^1, \ldots, U^n >$
Return: the set of symmetric meta equilibria $SymE(\Gamma)$

1. Initialize $SymE(\Gamma) = \emptyset$;
2. For i from 1 to n:
3. Find the set $SymE_i$ for agent i:
$SymE_i = \{\vec{a} | U^i(\vec{a}) \geq \min_{a_{-i}} \max_{a_i} U^i(a_i, a_{-i})\}$;
4. **If** $SymE_i$ is $\emptyset$
$SymE(\Gamma) = \emptyset$ and **break**;
5. **Else if** $SymE(\Gamma)$ is $\emptyset$
$SymE(\Gamma) = SymE_i$;
6. **Else**
$SymE(\Gamma) = SymE(\Gamma) \cap SymE_i$;
7. **If** $SymE(\Gamma)$ is $\emptyset$
break;
8. Return $SymE(\Gamma)$;

4. nMETAQ ALGORITHM

We use three **E**s to conclude the properties of meta equilibrium, namely: *existing*, *effective* and *efficient*. Firstly, there always exists at least one meta equilibrium for any game. Secondly, a meta equilibrium is a pure strategy equilibrium and it is not Pareto dominated by any non-equilibrium action. Thirdly, a meta equilibrium can be computed very efficiently, allowing for a broader range of application. Due to these properties of meta equilibrium, MetaQ works better than previous game theory-based MARL algorithms [4, 8, 11, 12]. In our opinion, the greatest advantage of meta

equilibrium compared with other types of equilibrium is that the three **E**s are still retained when there are more than two agents in the game. However, MetaQ only suits 2-agent Markov games. In order to be used widely, it should be extended to a more general form. Based on the equilibrium-computing algorithm provided in the last section, we give the general form of MetaQ, nMetaQ, where n means there are $n(n \geq 2)$ agents. We describe nMetaQ in Table 4 from the perspective of the ith agent.

In nMetaQ, all other agents' Q-tables are maintained by agent i for the purpose of computing meta equilibria in Steps 4 and 8. But, it is not necessary if all agents are homogenous and information exchange among them is allowed. In this situation, the equilibrium-computing process can be executed like this: (i) each agent finds the quasi-meta-equilibria set (like $SymE_i$ in Table 3) according to its own table; (ii) each agent broadcasts its set and achieves an appropriate meta equilibrium through a negotiation process. In this way, each agent can maintain its own Q-table only so that lots of storage space can be saved.

There is also an equilibrium selection problem for nMetaQ. Additionally, this problem is more serious than that of CE-Q because the condition of a meta equilibrium can be satisfied more easily. CE-Q has already provided us four selection mechanisms to solve this kind of problems. We employ the utilitarian one, which means to choose a meta equilibrium that maximizes the sum of all agents' utility. However, these mechanisms are too subjective and may not be truly effective. We will study this problem in our future work.

5. EXPERIMENTS AND RESULTS

We conduct the experiments in two grid-world games to test the performance of our algorithm. Although a grid-world game is pretty simple, it contains all key features of a dynamic multi-agent system and has become a standard benchmark for verifying an MARL algorithm.

Especially, we employ two 3-agent grid-world games: $GW1$ and $GW2$. Each of them is a 4×4 grid world. Figure 2 depicts their initial states. There are three agents in both games, which are denoted by circles and labeled as A, B and C, respectively. In each game, each agent's task is to move from its initial position to its goal, with the fewer steps, the better. In $GW1$, the three agents have their own goals G_A, G_B and G_C, respectively, while in $GW2$ all agents have the same goal G'.

In each step of the game, each agent can move only one grid in one of the four possible directions: up, down, left or right. A negative reward of -10 is received when an agent steps out of the border or there is a collision between two agents, and agents will be placed back to their previous positions in both the situations. Collision happens when two agents move to the same cell except that the cell is one of their goals. If an agent reaches its goal, a positive reward of 100 will be given. In other cases, the agent will get a small negative reward for its actions with the value of -1. The game ends only when all these three agents reach their goals.

It is easy to check that in $GW1$, the optimal strategies will take 5, 4 and 6 steps for A, B and C, respectively. In $GW2$, the numbers become 4, 4 and 5. We list some of the optimal solutions in Figure 3.

5.1 Convergence

Table 4: nMetaQ algorithm

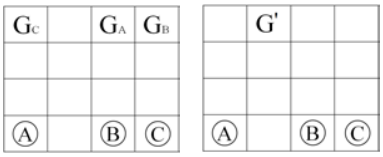
1. **Initialize** $Q^i(s, a_i, a_{-i})$ arbitrarily;
2. **Repeat** (for each episode):
3. **Initialize** state s;
4. Compute the meta equilibria of the game $(Q^i(s), Q^{-i}(s))$ and choose an appropriate one $\vec{a}_m(s)$;
5. **Repeat** (for each step):
6. Take the action $a_{m,i}$ of $\vec{a}_m(s)$ with probability $(1-\epsilon)$ or a random action with probability ϵ;
7. Observe and receive the experience $(s, \vec{a}, r, s')$;
8. Compute the meta equilibria in s' and choose an appropriate one $\vec{a}_m(s')$;
9. Update Q^j for $j = 1, ..., n$: $Q^j(s, \vec{a}) \leftarrow (1-\alpha)Q^j(s, \vec{a}) + \alpha[r + \gamma Q^j(s', \vec{a}_m(s'))]$;
10. $s \leftarrow s'$, $\vec{a}_m(s) \leftarrow \vec{a}_m(s')$;
11. **until** s is the terminal state;
12. **until** no more episodes;

The first experiment is carried out for the purpose of verifying the convergence of nMetaQ empirically. All agents in this experiment are nMetaQ agents. 50000 episodes are run for each game and the number of steps used to finish the game in each episode is recorded. We repeat this training process for 20 times and take the average values. The results are shown in Figure 4, where the vertical axis stands for the steps and the horizontal axis stands for the number of episodes. There are three learning curves in each figure, one for each agent. We do not plot all the 50000 points for each agent because this may lead to an illegible curve. Instead, we average the steps of every 1000 episodes and connect the 50 average points to make a curve.

The figure reveals that nMetaQ converges after about 3000 episodes in $GW1$ and $GW2$. In Figure 4(b), the learning curves of A and B almost overlap. Due to the ϵ-greedy policy used in the training process, the convergent values of the three agents are a bit higher than the theoretic optimal ones. Thus, we run a test after each training process to check whether the agents have learned the optimal policies. Only greedy actions are taken in each test. Interestingly, the agents choose several different optimal solutions in these 20 tests. This indicates that nMetaQ converges to different optima and demonstrates the diversity of meta equilibrium from another perspective.

5.2 Online Performance

The second experiment is carried out to investigate the online performance of nMetaQ agents when against other types of agents. In this experiment, A and B are still nMetaQ



(a) GW1 (b) GW2

Figure 2: Grid World Games: Initial States

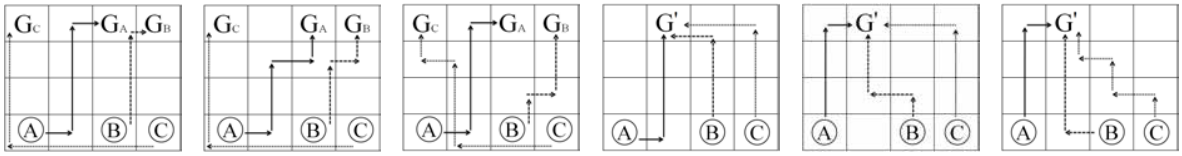
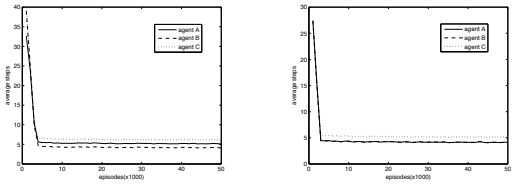
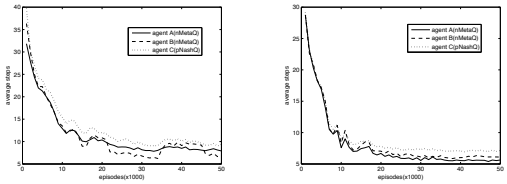


Figure 3: Optimal Solutions in GW1 and GW2



(a) learning curves in $GW1$ (b) learning curves in $GW2$

Figure 4: Convergence of nMetaQ



(a) learning curves in $GW1$ (b) learning curves in $GW2$

Figure 5: Online Performance of nMetaQ

agents while C is a NashQ agent. However, C only cares about finding pure strategy Nash equilibria. As noted in Section 2, a mixed strategy Nash equilibrium may not be suitable for MARL. Thus, we employ the concept of pure strategy Nash equilibrium for C. When there are no pure Nash equilibria, it will choose the joint action that maximizes the sum of each agent's utility. We call this algorithm *pNashQ*. The learning curves of each agent are shown in Figure 5.

From the two figures, we can see that all three curves converge, but the speed of convergence is almost four times lower than that in the first experiment. It is easy to explain this phenomenon. The agents have different types of rationality. Thus, it is difficult to predict what actions the other agents will take. Therefore, it takes more time for each agent to adapt to each other's behavior. The learning curves are rather circuitous. This also indicates the adapting process of the agents.

Recall that a pure strategy Nash equilibrium is also a symmetric meta equilibrium. Besides, C also use a utilitarian mechanism to choose actions when there is no pure strategy Nash equilibria. Therefore, A and B may often choose the same joint action chosen by C. That is to say, the two meta-rational agents can understand the intents of C well and then adapt to its behavior to achieve their common interests.

In fact, some other experiments are conducted in which nMetaQ agents are against Q-learning agents, or both Q-learning agents and pNashQ agents. The Q-learning agents in these experiments take actions according to their own tables. As a consequence, nMetaQ agents adapt other agents well and the learning curves are even smoother than those of the second experiment. However, due to the limitation of pages, the results of these experiments are not shown in this paper.

6. DISCUSSION

The first problem of nMetaQ is that the space complexity is exponential to the number of agents. For an n-agent ($n \geq 2$) Markov game which has $|S|$ different states, assuming that all agents have $m (m \geq 1)$ available actions, we need $|S| \cdot m^n$ entries for each agent's Q-table. If the agent in nMetaQ has to maintain the other agents' tables, the total space requirement becomes $n|S| \cdot m^n$.

This problem is also known as *curse dimensionality*, which is a common problem in RL. To address this problem, an intuitive solution is to use a function approximation method (e.g., neural networks [18], linear function [23], kernel-based method [14]) instead of this tabular nMetaQ. However, even if the Q-tables are replaced by an approximate function, we still need a table to represent the game in each state and then compute the meta equilibria. Another solution is to use an automated abstraction method [3, 5], which is a hot topic in the community of *large extensive-form game*. By automated abstraction, the state or action space of the original game Γ is reduced, making it a smaller game Γ'. Then, an equilibrium σ' is computed in Γ', followed by a reverse mapping applied to σ' to compute the approximate equilibrium σ of Γ. This is just an idea currently. Whether automated abstraction is suitable and how to use it in our nMetaQ still need to be studied in our future work.

Another problem worthy of discussion is the assumption of agents' meta-rationality: if some agents in the system are not meta-rational, is our algorithm still feasible? In NashQ, choosing a Nash equilibrium may cause an unwanted return if the opponent agents do not act in a Nash-rational way. Because the condition of Nash equilibrium is too restrictive, even if we do not know what kind of rationality the opponent agents have, we do not have enough reasons to assume they have Nash-rationality. But, meta-rationality is different. From Equations (11) and (13), we can see that a meta-rational agent will get rather satisfied if the utility value is higher than some kind of its conservation expectation. In other words, what the agent gets is often acceptable even if it may not be the best. This kind of rationality is rather common in the real world. Thus, for unknown agents, we can also assume they have meta-rationality. Our second

experiment has proved that nMetaQ is still effective in this situation.

7. CONCLUSIONS

In this paper, we propose nMetaQ algorithm, which is an extension of MetaQ and can be applied to multi-agent systems that have more than 2 agents. We have shown how to compute meta equilibria in a game and then presented nMetaQ algorithm. Also, we use two 3-agent grid-world games to test its performance. The result shows that nMetaQ has a good convergence property and can adapt to non-meta-rational agents well, which suggests that nMetaQ has the ability to solve multi-agent problems with more than two agents. However, there still exist some issues such as the problem of curse dimensionality. In addition, we have not proved the convergence of our algorithm, which is rather important.

Our future work includes: (i) convergence proof; (ii) using automated abstraction to address the problem of cure dimensionality, and (iii) trying to solve the equilibrium selection problem.

8. ACKNOWLEDGMENTS

We would like to acknowledge the support for this work from the National Science Foundation of China (Grant Nos. 61035003, 61175042, 61021062), the National 973 Program of China (Grant No. 2009CB320702), the 973 Program of Jiangsu, China (Grant No. BK2011005) and Program for New Century Excellent Talents in University (Grant No. NCET-10-0476).

9. REFERENCES

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] Y. Gao, J. Huang, H. Rong, and Z. Zhou. Meta-game equilibrium for multi-agent reinforcement learning. *AI 2004: Advances in Artificial Intelligence*, pages 1–5, 2005.

[3] A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 192–200. ACM, 2007.

[4] A. Greenwald, K. Hall, and R. Serrano. Correlated Q-learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 242–249. AAAI Press, 2003.

[5] J. Hawkin, R. Holte, and D. Szafron. Automated action abstraction of imperfect information extensive-form games. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence(AAAI-11)*, pages 319–325, 2011.

[6] N. Howard. *Paradoxes of Rationality: Games, Metagames, and Political Behavior*. MIT Press, 1971.

[7] J. Hu and M. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, 1998.

[8] J. Hu and M. Wellman. Nash Q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, 4:1039–1069, 2003.

[9] L. Kaelbling, M. Littman, and A. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] C. Lemke and J. Howson Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[11] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th international conference on machine learning*, pages 157–163, 1994.

[12] M. Littman. Friend-or-foe Q-learning in general-sum games. In *Proceedings of the 18th International Conference on Machine Learning*, pages 322–328, 2001.

[13] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[14] D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average-cost problems. *IEEE Transactions on Automatic Control*, 47(10):1624–1636, 2002.

[15] M. Osborne and A. Rubinstein. *A course in game theory*. MIT Press, 1994.

[16] J. Peng and R. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22(1):283–290, 1996.

[17] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

[18] M. Riedmiller. Neural fitted Q iteration: first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, pages 317–328, 2005.

[19] T. Sandholm. Perspectives on multiagent learning. *Artificial intelligence*, 171(7):382–391, 2007.

[20] T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, pages 495–501, 2005.

[21] Y. Shoham, R. Powers, and T. Grenager. Multi-agent reinforcement learning: a critical survey. *Unpublished survey. http://robotics. stanford. edu/shoham*, 2003.

[22] R. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[23] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[24] L. Thomas. Games, theory, and applications, 1984.

[25] C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[26] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Coordination Graphs in Continuous Domains for Multiagent Reinforcement Learning

Scott Proper
Oregon State University
Corvallis, OR 97331, USA
proper@eecs.oregonstate.edu

Kagan Tumer
Oregon State University
Corvallis, OR 97331, USA
kagan.tumer@oregonstate.edu

ABSTRACT

Many different coordination methods have been tried with multiagent reinforcement learning in the past, but there has been very little research testing the particular domains and situations in which different coordination methods might best apply. We test two coordination methods – difference rewards and coordination graphs – in a continuous, multiagent rover domain using reinforcement learning, and discuss the situations in which each of these methods perform better alone or together, and why. We also contribute a novel method of applying coordination graphs in a continuous domain by taking advantage of the wire-fitting approach used to handle our continuous state and action space.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed Systems

General Terms

Algorithms, Performance, Experimentation

Keywords

Multiagent Coordination, Reward Shaping, Coordination Graphs, Scaling, Wire Fitting, Neural Networks

1. INTRODUCTION

Reinforcement learning in multiagent systems is a wide area of research with applications ranging from robocup soccer [21], to rover coordination [1], to trading agents [20, 26], to air traffic management [23]. The problem of scaling reinforcement learning to such large domains is particularly challenging because the agents in the system provide a constantly changing environment in which each agent needs to learn its task. Agents must somehow learn to coordinate among themselves and develop a joint set of policies to solve the problem. Agents are usually learning simultaneously, further complicating the learning process as the behavior of the other agents is changing in unpredictable ways.

Approaches to such cooperative multiagent RL problems have been numerous and varied. Typically, a method to coordinate the agents is introduced that allows the agents to take advantage of some structure of the problem, and often some form of communication between agents is permitted. In particular, two forms of coordination that have been popular in the past include coordination graphs, which allow small groups of agents to coordinate actions based domain-specific dependencies, represented by a graph of edges between agents [8]; and difference rewards, which are a specific type of shaped reward that encourages good agent behavior by rewarding actions that are closely aligned with the desired overall system behavior, while still allowing agents to learn from the reinforcement signal [1]. Both of these methods have been show to perform well in various domains, for example coordination graphs have done very well in discrete predator-prey domains [13, 17], system control of connected machines [7], and traffic light control [15]. Difference rewards have been shown to do well in highly congested domains in which agents might compete for resources, including rover coordination [1], sensor network control [19], and air traffic control [18, 23].

There is unfortunately little previous work comparing and contrasting entirely different coordination techniques, such as coordination graphs or difference rewards (among others). For example, given a new domain, how can a researcher know which of many methods to apply? What characteristics of a domain suggest a particular method of coordination, and is it a good idea to try more than one? While this paper does not attempt to answer these question in full (an impossible task), we do examine the two coordination methods described above and compare them in a simulated domain of continuous rover navigation and observation with multiple agents. Our conclusions should be helpful for future researchers who wish to understand the differences between these methods for their own applications. Our results illustrate some of the possible differences between domains that can cause one or another coordination method to be an appropriate choice.

We make three main contributions in this paper: first, we apply wire-fitting, a method of performing reinforcement learning in continuous domains, to a multiagent problem. To our knowledge, this is the first time this has been done. Second, we demonstrate an implementation of coordination graphs in a continuous multiagent domain, which again we believe has never been done before. Finally, we compare and contrast two popular coordination techniques, coordination graphs and difference rewards, and obtain results applying them together and separately to the continuous rover domain.

In the next section, we discuss our general framework for multiagent reinforcement learning. In Section 3, we describe our solution to the problem of learning in a continuous state and action space. In Section 4, we describe our first coordination strategy, coordination graphs. In Section 5, we describe our second coordination strategy, reward shaping via difference rewards. In Section 6, we introduce our domain. In Section 7, we present experimental results and finally in Section 8, we discuss our conclusions.

2. MULTIAGENT REINFORCEMENT LEARNING

A Markov Decision Process (MDP) is a tuple $\langle S, A, P, R, I \rangle$ where S is a finite set of discrete states and A is a finite set of ac-

tions available to the agent. The actions are stochastic and Markovian in that an action $a \in A$ in a given state $s \in S$ results in a state s' with fixed probability given by $P(s'|s,a)$, which is called a transition function. The reward function $R : S \times A \to \mathbb{R}$ returns the reward $R(s,a)$ after taking action a in state s. I is the initial state distribution. Each action is assumed to take one time step. An agent's policy is a mapping $\pi : S \to A$. A stationary policy is one which does not change with time. A deterministic policy always maps the same state to the same action. For the remainder of this paper, "policy" refers to a stationary deterministic policy.

A reinforcement learning agent learns by a process of trial and error, repeatedly attempting actions within the environment and receiving a reward signal and a description of the state. The goal of an RL agent is to learn a policy that maximizes some measure of the long-term reward received, such as expected total reward, discounted reward, or average reward. We can indirectly learn such a policy by learning a *value function*, which maps states or state-action pairs to values. We can then compute the best action in each state by calculating the highest-valued action using one-step lookahead search and the value function.

In this paper we use a modified version of the *Q-learning* algorithm [22] to calculate an action-value function. The optimal action-value function or Q-function gives the expected discounted future reward for any state s when executing action a and then following the optimal policy. The Q-function satisfies the following Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \qquad (1)$$

where $\gamma \in [0,1)$ is the discount factor. The optimal policy for a state s is the action $a^* = \arg\max_a Q^*(s,a)$. At each time step, Q-learning updates the Q-values as follows after executing action a in state s and receiving the reward $r(s,a)$:

$$Q(s,a) \overset{\alpha}{\leftarrow} r(s,a) + \gamma Q(s', a^*) \qquad (2)$$

where $Q(s',a^*) = \max_{a'} Q(s',a')$, $\alpha \in [0,1]$ is the learning rate, and $A \overset{\alpha}{\leftarrow} B$ is equivalent to $A \leftarrow (1-\alpha)A + \alpha B$.

One version of multiagent Q-learning [22] updates each agent's Q-value independently using the update function:

$$Q_i(\mathbf{s}_i, a_i) \overset{\alpha}{\leftarrow} r_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, a^*_i) \qquad (3)$$

The notation Q_i indicates only that the Q-value is agent-based. The term r_i indicates that the reward is factored, i.e. a separate reward signal is given to each agent. There are a variety of means to calculate this reward signal based on global or local information; these will be discussed in later sections.

Once the Q-function has converged to its true value, the optimal action is suggested by maximizing the right hand side of the above update equation. However, to avoid convergence to a local optimum, all RL methods need to ensure that all actions in the relevant states are explored sufficiently thoroughly. Several exploration methods exist to provide such experience. In this paper, we use an ϵ-greedy strategy for our experiments, which takes a random action with probability ϵ and a greedy action with probability $1-\epsilon$.

3. LEARNING IN CONTINUOUS STATE AND ACTION SPACES

The problem of reinforcement learning in continuous state and action spaces has been well-studied in the past [5, 24], but remains a difficult problem for learning. Typical RL approaches assume discrete state and actions, however there exist solutions that extend RL into continuous problems. We discuss the solutions we use below.

3.1 Wire Fitting

To deal with the problem of acting in a space in which stands and actions may be continuous, Baird and Klopf propose the wire fitting algorithm, which efficiently stores an approximation of a continuous Q-function [3]. They propose using a function approximator (in this paper, we use a neural network) to concurrently output a fixed number of actions and corresponding values ("wires"), given a certain state. If the value of an interlying action is required, interpolation is performed as follows:

$$\begin{aligned} Q(s,a) &= \lim_{\epsilon \to 0} \frac{\sum_{i=0}^{n} \frac{q_i(s)}{\|a - a_i(s)\|^2 + c(q_{max}(s) - q_i(s)) + \epsilon}}{\sum_{i=0}^{n} \frac{1}{\|a - a_i(s)\|^2 + c(q_{max}(s) - q_i(s)) + \epsilon}} \\ &= \lim_{\epsilon \to 0} \frac{\sum_{i=0}^{n} q_i(s)/d_i(s,a)}{\sum_{i=0}^{n} 1/d_i(s,a)} \\ &= \lim_{\epsilon \to 0} wsum(s,a)/norm(s,a), \end{aligned} \qquad (4)$$

where s and a are the state and action for which a value is desired, n is the number of action-value pairs (wires) output and $a_i(s)$ and $q_i(s)$ are the outputs corresponding to the i^{th} action and action-value for this state, respectively. $q_{max}(s) \equiv \max_j q_j(s)$ is the maximum of all q_i. c_i is a smoothing factor and ϵ is a small constant preventing a division by zero. The interpolation gives a weighted average of different values, depending on the distance of the given action to the output actions on this particular state.

An agent using wire fitting having an experience $\langle s, a, r, s' \rangle$ may update the system as follows: Let $\theta_i^{a_j}$ denote the parameters of the function approximator that outputs action a_j. The following update will change the parameters $\theta_i^{a_j}$ of the function approximator outputting action a_j according to gradient descent on the error of the value:

$$\theta_i^{a_j} \leftarrow \theta_i^{a_j} + \alpha(r(s,a) + \gamma Q(s', a^*) - Q(s,a)) \frac{\partial Q(s,a)}{\partial a_j} \frac{\partial a_j}{\partial \theta_i^A} \qquad (5)$$

where:

$$\frac{\partial Q(s,a)}{\partial a_j} = \lim_{\epsilon \to 0} \frac{2(wsum(s,a) - norm(s,a) q_j)(a_j - a)}{(norm(s,a) d_i(s,a))^2} \qquad (6)$$

A similar update can be done for the parameters of the function approximators that output the values:

$$\frac{\partial Q(s,a)}{\partial q_j} = \lim_{\epsilon \to 0} \frac{norm(s,a)(d_i(s,a) + q_j c) - wsum(s,a) c}{(norm(s,a) d_i(s,a))^2} \qquad (7)$$

Because of the nature of the interpolator, the action with the highest value is always one of those output by the function approximator. Thus we do not have to use the interpolation function to find the action, we just select the action with the highest corresponding value. This results in fast action selection. In a fully trained system with a low error between experienced action-value pairs and the interpolation output, we can assume that this is the optimal action, though convergence guarantees can not be given.

The approach is called wire fitting because the interpolation function describes a surface in $S \times A \times \mathbb{R}$ space, which is draped over "wires" defined by the outputs of the function approximator. Because the whole value function is approximated, with enough wires, the system can reach any real valued optimal policy, generalize well, while allowing fast action selection.

3.2 Practical Issues

Neural networks may suffer from a problem known as unlearning or interference, which can cause the network to unlearn the correct output for other inputs because recent experience dominates

the training data [4]. We deal with this problem by storing examples of state, action and next state transitions and replaying them as if they are being re-experienced. This process is known as persistent excitation, and creates a constantly changing input to the neural network. Target outputs for the network are re-calculated from scratch using the wire-fitter each time as old targets would become incorrect as Q-values are updated over time. This method makes efficient use of data gathered from the world without relying on extrapolation, which allows reinforcement learning to learn with fewer episodes. A disadvantage is that if conditions change the stored data could become misleading.

An additional problem with applying Q-learning to continuous problems is that a single suboptimal action will not prevent a high value action from being carried out at the next time step. Thus the value of actions in a particular state can be very similar, as the value of the action in the next time step will be carried back. As the Q-value is only approximated for continuous states and actions it is likely that most of the approximation power will be used representing the values of the states rather than actions in states. The relative values of actions will be poorly represented, resulting in an unsatisfactory policy. The problem is compounded as the time intervals between control actions get smaller.

Advantage Learning [10] addresses this problem by emphasizing the differences in value between the actions. In Advantage Learning the value of the optimal action is the same as for Q-learning, but the lesser value of non-optimal actions is emphasized by a scaling factor ($k \propto \Delta t$). This makes a more efficient use of the approximation resources available. The Advantage Learning update is:

$$\mathcal{A}(s,a) \overset{\alpha}{\leftarrow} \frac{1}{k}\left[r(s,a) + \gamma \mathcal{A}(s',a^*)\right] + (1-\frac{1}{k})\mathcal{A}(s,a^*) \quad (8)$$

where $\mathcal{A}$ is analogous to Q in [22]. Following [6], we set $k = 1-\gamma$ for all experiments in this paper.

4. COORDINATION GRAPHS

A coordination graph can be described over a system of agents to represent the coordination requirements of that system. Such a graph contains a node for each agent and an edge between pairs of agents if they must directly coordinate their actions to optimize some particular Q_{ij}. See Figure 1 for an example coordination graph showing some possible coordination requirements between four agents. The existence of an edge (and therefore a coordination dependency) is dependent on the properties of the domain, state, and coordination requirements: for example, the rover domain introduced in Section 6 adds edges between the n closest neighbors within a certain communication radius. Coordination graphs are an *explicit* form of coordination, in that they require communication between agents.

In most cases, if each agent independently pursues a policy to optimize its own reward, this will not optimize the total utility, since each agent's actions affect the state and the utility of others. Hence, collaborative agents need to coordinate. A coordination graph allows the agents to specify and model coordination requirements [7]. An edge in a coordination graph indicates that two agents should coordinate their action selection, for example, so as to avoid collisions. A coordination graph may be specified as part of the domain, or if the graph is *context-specific* [9], as a combination of rules provided with the domain by the experimenter. Such rules determine whether an edge between any two vertices of the graph should exist, given the state.

As in Kok and Vlassis [14] we use an edge-based decomposition of a context-specific coordination graph. The global Q-function for such a decomposition is approximated by a sum over all local Q-functions, each defined over an edge (i,j) of the graph:

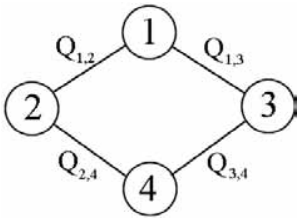


Figure 1: A possible coordination graph for a 4-agent domain. Q-values indicate an edge-based decomposition of the graph.

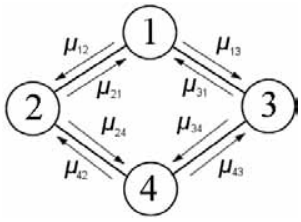


Figure 2: Messages passed using Max-plus. Each step, every node passes a message to each neighbor.

$$Q(\mathbf{s},\mathbf{a}) = \sum_{(i,j)\in E} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j). \quad (9)$$

where $\mathbf{s}_{ij} \subseteq \mathbf{s}_i \cup \mathbf{s}_j$ is the subset of state variables relevant to agents i and j, and $(i,j) \in E$ describes a pair of neighboring nodes (i.e., agents). The optimal action for a coordination graph is given by $\arg\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$. As with the agent-based Q-function, the notation Q_{ij} indicates only that the Q-value is edge-based. Parameters may or may not be shared between edges.

In general, an edge should be placed between two agents when the actions of those agents might interfere, such as when a collision is possible or the two agents might need to share a common resource. Such coordination must be context-specific, since agents are constantly changing states. It is possible to learn coordination requirements automatically [12], however in this paper coordination requirements (and thus the structure of the coordination graph) are defined by the experimenter. In the next section we adapt some methods described by Kok and Vlassis [14].

4.1 The Max-plus Algorithm

If we define a coordination graph over several agents, we must use an action selection algorithm that can take advantage of this structure. We wish to maximize the global payoff $\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$, (where $Q(\mathbf{s},\mathbf{a})$ is given by Equation 9). Initial work in coordination graphs suggested a variable elimination (VE) technique [8] to solve this problem. However previous work has shown that VE techniques can be slow to solve large coordination graphs, are space intensive, and in addition can be quite complex to implement [14]. Instead, Kok and Vlassis [14] proposed using the *Max-plus* algorithm, which trades solution quality for solution speed.

The Max-plus algorithm is a message-passing algorithm based on loopy belief propagation for Bayesian networks [16, 27, 25]. Instead of probability distributions, agents in Max-plus pass (normalized) values indicating the locally optimal payoff of each agent's actions along edges of the coordination graph (see Figure 2). Max-plus finds the global payoff by having each agent i repeatedly sending messages μ_{ij} to its neighbors:

$$\mu'_{ij} = \max_{a_i}\Big\{ f_i(a_i) + f_{ij}(a_i,a_j) + \sum_{k\in\Gamma(i)\setminus j} \mu_{ki}(a_i)\Big\} - c_{ij} \quad (10)$$

where μ_{ki} is the incoming message, and μ'_{ij} is the outgoing message. The functions $f_i(a_i) = Q_i(\mathbf{s}_i, a_i)$ and $f_{ij}(a_i,a_j) = Q_{ij}(\mathbf{s}_{ij}, a_i, a_j)$ store the Q-values corresponding to the current state and given actions for the indicated agent i and agent-agent pair i,j respectively. All messages are in fact vectors over possible actions. $\Gamma(i) \setminus j$ represents all neighbors of i except j and c_{ij} is

Input: Graph $G = (V, E)$
Output: A vector of actions a^*
Initialize $\mu_{ji} = \mu_{ij} = 0$ for $(i, j) \in E$, $g_i = 0$ for $i \in V$ and $m = -\infty$
while $converged$ = *false and deadline not met* **do**
  $converged$ = true
  foreach *agent* i **do**
    foreach *neighbor* $j \in \Gamma(i)$ **do**
      foreach *action* $a_j \in A_j$ **do**
        Create message $\mu'_{ij}(a_j) = \max_{a_i}\{f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k\in\Gamma(i)\setminus j} \mu_{ki}(a_i)\}$
      end
      $c_{ij} = \frac{1}{|A_j|}\sum_{a_j} \mu'_{ij}(a_j)$
      Normalize: $\mu'_{ij}(a_j) \leftarrow \mu'_{ij}(a_j) - c_{ij}$
      if μ'_{ij} *differs from* μ_{ij} *by a small threshold* **then**
        $converged$ = false
    end
    $a'_i = \arg\max_{a_i}\{f_i(a_i) + \sum_{j\in\Gamma(i)} \mu_{ji}(a_i)\}$
  end
  if $u(a') > m$ **then**
    $a^* = a'$ and $m = u(a')$
  $\mu \leftarrow \mu'$
end
return a^*

Algorithm 1: The centralized anytime Max-plus algorithm.

a normalization factor, calculated after the initial values μ'_{ij} have been found. Max-plus sets this to be the average over all values of the outgoing message: $c_{ij} = \frac{1}{|A_j|}\sum_{a_j} \mu'_{ij}(a_j)$. This prevents messages from exploding in value as multiple iterations of the algorithm proceed. Once messages have converged or a time limit has been reached, each agent chooses the action that maximizes:

$$\arg\max_{a_i}\{f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{j\in\Gamma(i)} \mu_{ji}(a_i)\}$$

for that agent. See Algorithm 1 for details.

4.2 Dynamic Coordination

Although we use an edge-based decomposition (as described above), it is often the case that rewards are received on a per-agent basis instead of a per-edge basis. Thus, we must compute local Q_i functions for each agent in the graph. Following [14], we do this by assuming each Q_{ij} contributes equally to each agent i and j of its edge:

$$Q_i(\mathbf{s}_i, a_i) = \frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) \quad (11)$$

where $\Gamma(i)$ indicates the neighbors of agent i. The sum of all such Q_i functions equals Q in Equation 9. We assume each agent has at least one other neighbor in the coordination graph. It is straightforward to adapt these methods in cases where an agent does not need to coordinate with anyone.

Because our coordination graph is context-specific, to update the Q-function we must use an agent-based update as opposed to an edge-based update. This is because the presence or absence of edges changes from state to state, so we cannot be assured that an edge that is present in the current time step was available in the last time step. To obtain the agent-based update equation for an edge-based decomposition using Advantage Learning, Equation 8 is rewritten using Equation 11 following a derivation similar to that in [14] to get:

$$Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) \xleftarrow{\alpha} \sum_{b\in\{i,j\}} \frac{\frac{1}{k}\left[r_b(\mathbf{s}, \mathbf{a}) + \gamma Q_b(\mathbf{s}'_b, a^*_b)\right] + (1 - \frac{1}{k})Q_b(\mathbf{s}_b, a^*_b)}{|\Gamma(b)|} \quad (12)$$

This update equation propagates the temporal-difference error from all edges that include agents i and j to the local Q-function of each edge (i, j). This update is context-specific, because it does not require the same edges to be present at each time step of the Q-learning algorithm. It only requires that local Q_b functions can be computed for each vertex of the coordination graph, which is done using Equation 11. The notation Q_i and Q_{ij} indicates that the Q-values are agent-based or edge-based respectively. Q-function parameters are shared between agents and edges.

5. REWARD SHAPING

When providing the reward signal to an agent, a reasonable option is to provide each agent with the **full system reward** $G(z)$ for each step. This leads to each agent using that reward to update its value estimates for each action. However, this reward signal is not particularly sensitive to an agent's actions and for large systems, leads to particularly slow learning. Previous work has shown that we can instead provide a *difference reward*, which can significantly outperform agents either receiving a purely local reward or all agents receiving the same system reward [1, 23].

Unlike coordination graphs, which are an explicit form of coordination, difference rewards are an *implicit* form of coordination in that they do not necessarily require agent-to-agent communication. Difference rewards are shaped rewards which may be indirectly provided to the agent through a broadcast signal of the global (system) reward, or determined through direct observation by the agents. The difference reward is given by [1]:

$$D_i(z) = G(z) - G(z - z_i);, \quad (13)$$

where $z - z_i$ specifies the state of the system without agent i[1]. In this instance z is the actions of the agents, and $z - z_i$ represents the actions of all the agents without agent i. Difference rewards are *aligned* with the system reward, in that any action that improves the difference reward will also improve the system reward. This is because the second term on the right hand side of Equation 13 does not depend on agent i's actions, meaning any impact agent i has on the difference reward is through the first term (G) [23]. Furthermore, it is more sensitive to the actions of agent i, reflected in the second term of D, which removes the effects of other agents (i.e., noise) from agent i's reward function. Intuitively this allows the second term of the difference reward to evaluate the performance of the system without agent j and therefore D evaluates the agent's individual contribution to the system performance.

There are two key advantages to using D: First, because the second term removes a significant portion of the impact of other agents in the system, it provides an agent with a "cleaner" signal than G. This benefit has been dubbed *learnability* (agents have an easier time learning) in previous work [1]. Second, because the second term does not depend on the actions of agent j, any action by agent j that improves D, also improves G. This term which measures the amount of alignment between two rewards has been dubbed *factoredness* in previous work [1].

[1]In this paper we will use zero padded vector addition and subtraction to specify the state dependence on specific components of the system.

6. CONTINUOUS ROVER PROBLEM

We implement a variation of the continuous rover problem, as described in [2]. In this problem, there is a set of rovers on a two dimensional plane which are trying to observe points of interests (POIs). A POI has a fixed position on the plane and has a value associated with it. The observation information from observing a POI is inversely related to the distance the rover is from the POI. In this paper the distance metric will be the squared Euclidean norm, bounded by a minimum observation distance, d:

$$\delta_{x,y} = min\{\|x - y\|^2, d^2\} \ . \tag{14}$$

As in [11], we may optionally encourage the formation of teams by requiring two or more rovers to observe a POI simultaneously in order to receive a reward. We do this by defining a maximum observation distance δ_0. Only the closest rovers to a POI within radius δ_0 contribute to the global reward, and so the global evaluation function for a single time step is given by:

$$G(z) = \sum_i \frac{V_i \cdot N \cdot I(|c_i| \geq N)}{\sum_{k=1}^{N} \delta_{i,c_i(k)}}, \tag{15}$$

where V_i is the value of the ith POI, c_i is the set of the agents within viewing range δ_0 of POI i and $c_i(k)$ is the kth closest agent, N is the minimum number of agents required to make a successful observation, and $I(|c_i| \geq N)$ is an indicator function returning 1 if there are enough agents viewing POI i to observe it, 0 otherwise. This equation essentially contributes the value of each successfully observed POI divided by the average distance of the N closest agents observing it to the global reward. The single rover ("local") objective used by each rover only focuses on the value a rover receives for observing a particular POI:

$$L_j(z) = \sum_i \frac{V_i \cdot I(\delta_{i,j} \leq \delta_0) \cdot I(|c_i| \geq N)}{\delta_{i,j}}, \tag{16}$$

where $I(\delta_{i,j} \leq \delta_0)$ is an indicator function returning 1 if agent j is within range to observe POI i, 0 otherwise. This objective promotes selfish behavior only, providing a clear, easy-to-learn signal, but one not necessarily aligned with the system objective as a whole. Finally, the difference objective for a rover provides system-wide beneficial behavior, while remaining sensitive to the actions the agent takes [1]:

$$D_j(z) = \sum_i \begin{cases} \frac{V_i N}{\sum_{k=1}^{N} \delta_{i,c_i(k)}} - \frac{V_i N}{\sum_{k=1,k\neq j}^{N+1} \delta_{i,c_i(k)}} & \\ \qquad \text{if } |c_i| > N \wedge j \in \{c_i(1)...c_i(N)\} & \\ \frac{V_i N}{\sum_{k=1}^{N} \delta_{i,c_i(k)}} \qquad \text{if } |c_i| = N \wedge j \in c_i & \\ 0 \qquad \text{otherwise} \end{cases} \tag{17}$$

This equation has three cases. In the first case, agent j is one of the N agents $\{c_i(1)...c_i(N)\}$ observing a POI i and there is another agent $c_i(N+1)$ close enough to POI i to substitute in for agent j should it disappear. In the second case, agent j is again one of the N agents c_i observing POI i, but there is no agent to substitute for it, thus the contribution of all N agents for POI i to the global reward will be lost should agent j disappear.

At every time step, the rovers sense the world through eight continuous sensors. From a rover's point of view, the world is divided up into four quadrants, with two sensors per quadrant. Unlike the implementation used in [2], we do not change orientation of these quadrants as the rovers move. For each quadrant, the corresponding sensor returns a function of the POIs in that quadrant at that moment. Specifically the first sensor for quadrant q returns the sum of the values of the POIs in its quadrant divided by their squared distance to rover j:

$$s_{1,q,j} = \sum_{i \in I_q} \frac{V_i}{\delta_{i,j}} \tag{18}$$

where I_q is the set of observable POIs in quadrant q. The second sensor returns the sum of square distances from a rover to all the other rovers in the quadrant at time t:

$$s_{2,q,j} = \sum_{j' \in N_q} \frac{1}{\delta_{j',j}} \tag{19}$$

where N_q is the set of rovers in quadrant q.

With four quadrants and two sensors per quadrant, there are a total of eight continuous inputs. This eight dimensional sensor vector constitutes the state space for a rover. At each time step the rover uses its state to compute a two dimensional action $\langle dx, dy \rangle$. The action represents an x,y movement relative to the rover's location and orientation. The mapping from state to action is done using wire-fitting using a feed-forward artificial neural network, with 8 input units, 10 hidden units and 11 wires for 33 output units (movement in x and y, and Q-value). The network uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actions dx and dy are determined by mapping output in the range of $(.1, .9)$ to $(-10, 10)$, 10 being the maximum distance the rover can move in one time step. As the POI and Rover sensor values could also vary widely, they were scaled by the function $\frac{2}{1+e^{-s\zeta}} - 1$ before being input into the neural network, where s is computed from either Equation 18 or 19 and $\zeta = 10$ or 5 is a scaling factor for POIs and rovers respectively.

6.1 Coordination Graphs in Continuous Domains

Several further problems arise when adapting coordination graphs for use in continuous domains such as the Rover domain. For example, it is not always clear from the dynamics of the domain which agents should coordinate with each other. Ideally, all agents should coordinate, but this is impractical. We choose to limit coordination based on two factors: distance, and an additional upper limit on the number of neighbors for each agent. If there are too many potential neighbors within the distance limit, the closest neighbors are chosen. This forces the coordination graph into a limited coverage allowing practical application of Max-Plus algorithm to compute a good joint action in reasonable time.

A further significant issue is that the Max-Plus algorithm requires a finite number of actions that each agent may select from. Unfortunately, in a continuous action space there are an infinite number of possible actions. To remedy this, we select a set of "candidate" actions for each agent from the output of the wire-fitter storing the value function $f_i(\cdot)$ for each agent. Only the top three candidates are selected. There are many other possible ways to select candidates, however we found that this method gave us the best balance of speed and quality of results.

Inputs to the neural network approximating $Q_{ij}(\mathbf{s}_{ij}, a_i, a_j)$, the value function for the coordination graph, needed to be a little different to that used for $Q_i(\mathbf{s}_i, a_i)$ in the previous section. Instead of four rover and four POI sensors, we used four POI sensors for both agents i and j, and two additional distance sensors that measured distance in the x and y coordinates, for a total of 10 inputs to the network. Rover sensors were ignored as the network for $Q_i(\mathbf{s}_i, a_i)$, and the coordination graph itself, are sufficient to handle coordination.

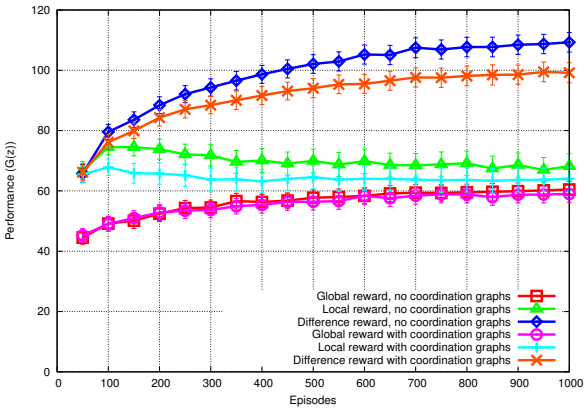


Figure 3: Comparison of results with agents starting in the center, with $N = 1$. In this case, the difference reward performs very well allowing agents to cope with congested conditions, however coordination graphs actually harm convergence in this domain.

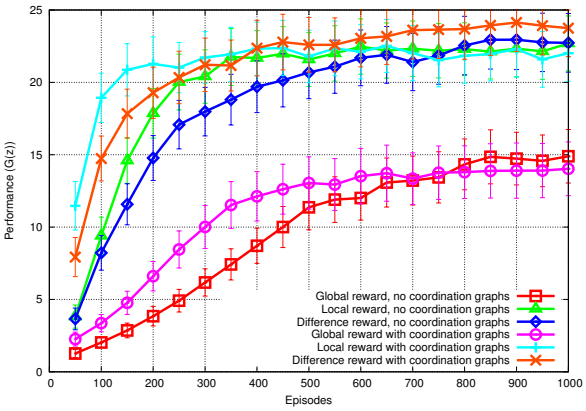


Figure 4: Comparison of results with agents starting in the center, with $N = 3$. Difference rewards again do well in coping with congestion, however here coordination graphs are actually helpful to ensure the correct number of agents observe each POI. Local rewards do surprisingly well: a "selfish" action is often the correct one when you need other agents to also behave selfishly (i.e., observe the same nearby POI).

7. EXPERIMENTAL RESULTS

We conducted experiments with a 10-agent, 30-POI version of the Rover domain discussed in Section 6 with a dynamic, randomly generated distribution of agents and POIs. We performed experiments with local, global, and difference rewards, and with and without coordination graphs. 20 points were plotted for each result, each point the average of 50 episodes. We additionally averaged results over 30 runs for all experiments. Although learning was performed using various reward signals, all results were measured against the global reward $G(z)$, which is the true objective we wish to maximize. Error bars are shown and were calculated using the sample standard error of the mean $\sigma/\sqrt{n}$ where n is the number of runs.

All experiments were conducted on a continuous surface of 100x 100 units, with a "starting" or "center" position at (50,50). POIs were uniformly randomly placed within a 70x70 area surrounding this center position. All POIs were given a value of 5, a minimum observation distance $d = 2$, and a maximum observation distance of $\delta_0 = 5$, thus the reward that any individual POI may contribute to the global is in the range $(.2, 1.25)$, using Equation 14. The number of "thinks" used to provide persistent excitation to the neural networks between steps was set to 1000. The maximum distance a rover could move in one time step was 10. When using coordination graphs, the maximum distance a rover would coordinate with another rover was set to twice this movement speed, or 20, and each rover could coordinate with at most three other rovers.

Learning-related constants were as follows: the smoothing constant $c = 1$, exploration rate $\epsilon = .1$, discount factor $\gamma = .5$, neural network learning rate $\eta = .1$ with no momentum.

Because the locations of the POIs changed locations at each trial, this forced the rovers to create a general solution based on their sensor inputs. This type of problem is common in real world domains, where the rovers typically learn in a simulator and later have to apply their learning to the environment in which they are deployed. Note that learning in this scenario does not depend on the use of multiple trials as the rovers can continuously learn and generalize from their past experience.

We compare experiments with four variations on the rover domain: two experiments with agents starting in very congested conditions, uniformly distributed in 5x5 area at the center position (Figures 3 and 4), and two experiments with the agents uniformly scattered in the same 70x70 area the POIs are scattered in (Figures 5 and 6). These experiments were further varied based on the minimum number of agents required to perform an observation: $N = 1$ (Figures 3 and 5) or $N = 3$ (Figures 4 and 6).

In Figure 3, with agents starting in the center and $N = 1$, we observe that using the difference reward performs very well due to the highly congested conditions. Difference rewards have been shown to perform very well in similar congested conditions in the past [1], and these results confirm that result. However, the addition of coordination graphs adds unnecessary complexity to the learner: coordination graphs are not needed to solve a coordination problem in this domain and thus the extra parameters of the neural network required to store the coordination graph only slow convergence.

In Figure 4, the learning problem is made much harder for the agents by requiring three agents to make an observation simultaneously if reward is to be received. Difference rewards again do well, however in this case the extra coordination provided by coordination graphs is helpful: using both forms of coordination simultaneously provides the best result. Local rewards still do fairly well, however, due to the fact that the "selfish" behavior resulting from such a policy tends to result in good outcome: more than one rover will visit a POI, which in this case is desirable behavior.

In Figure 5, agents are starting widely scattered across the available area, and require only one agent to observe a POI. In this case, it makes little difference whether agents learn using local or difference rewards, or use coordination graphs or not. In general, the best policy is for agents to move to a nearby POI and stay there, avoiding other agents. There is almost no congestion so difference rewards are not needed, and very little close coordination with other agents is needed so coordination graphs can help only a small amount.

In Figure 6, agents are again starting widely scattered, but we require three agents to make an observation. In this case, the learning problem is for agents to join up across their scattered positions and observe a POI together. This is again an uncongested domain, so difference rewards are of very little use here, and can in fact somewhat harm convergence compared to using local rewards. However, we see that coordination graphs are necessary for agents to find

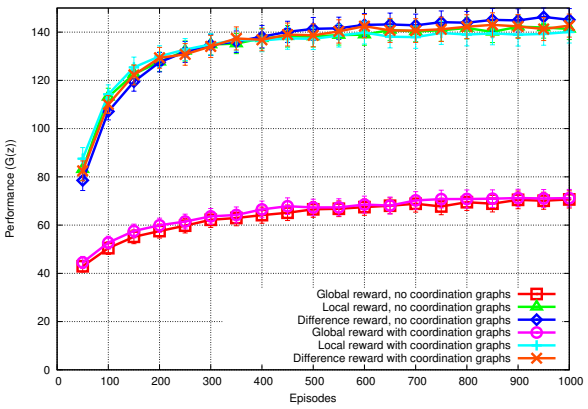


Figure 5: Comparison of results with agents starting widely distributed, with $N = 1$. In this case, choosing between local or difference rewards, or using coordination graphs or not, makes little difference. All these choices perform similarly well in such an uncongested domain. Learning using the global reward is slow to converge, as expected.

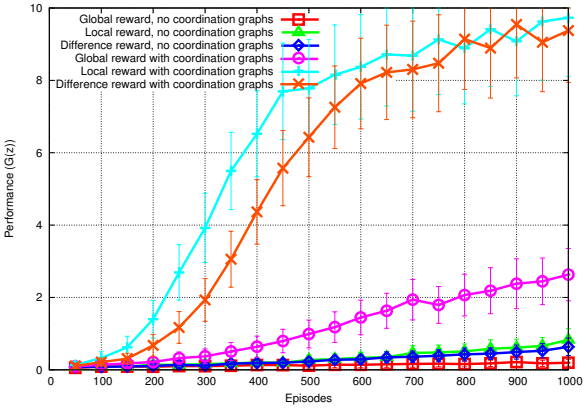


Figure 6: Comparison of results with agents starting in the center, with $N = 3$. Here, coordination graphs are highly useful in allowing scattered agents to meet at a POI to conduct observations: without coordination graphs, performance is very poor. Local rewards converge slightly faster than difference rewards.

each other and navigate to a POI together; without coordination graphs, agents do not learn to converge and observe POIs.

From these tests we conclude that coordination graphs and difference rewards are solving different coordination problems. Difference rewards are ideal for highly-congested domains; coordination graphs are well-suited to cases in which pairs or groups of agents must take specific coordinated actions together to succeed. Including either form of coordination when it is not needed can be harmful, but rarely prevents learning from taking place and may only slow convergence slightly. Cases exist when using both forms of coordination together is helpful, i.e. when a problem requires more than one form of coordination. These conclusions are summarized in Table 1.

8. DISCUSSION

This paper has several contributions. First, we apply wire-fitting to a multiagent domain. Second, we show an implementation of coordination graphs in a continuous domain. To our knowledge neither of these two contributions have been demonstrated before. Several adaptations were needed to allow coordination graphs to apply in a continuous domain: first, we needed to apply advantage learning. Thus, we provide Equation 12, which is an update that extends coordination graphs into a continuous-time domain. Second, as the max-plus algorithm requires each agent to have a finite number of actions, we provide a method for determining a finite set of "candidate actions". In principle, almost any method for determining candidate actions could work, however good performance requires we provide a small set of actions, one of which is likely to be a good compromise between the two agents of an edge of the coordination graph. Our method of choosing the top three actions suggested by the wire-fitter works quite well in practice, though future work could potentially improve on this further.

Our final main contribution is an examination of the conditions under which two popular coordination methods – difference rewards and coordination graphs – can solve coordination problems in a multiagent domain. Both of these techniques seem to solve a similar problem: how can we coordinate the actions of agents so as not to interfere with each other, and achieve a global system objective? Our results show that though these are both coordination methods, they solve different coordination problems. Difference rewards are most useful in highly congested domains, where agents might compete for resources if they behave selfishly, thus harming the global performance. Coordination graphs are most useful when agents must take related actions together to achieve an objective; for example, if they must both navigate to the same area or POI. Difference rewards do not provide enough information to the agent to take tightly-coordinated actions. This conclusion is supported by past work involving both these techniques: coordination graphs have primarily been used for domains in which tight coordination between agents is needed [13, 17, 7, 15], while difference rewards have primarily been applied to domains in which congestion is an important factor [1, 19, 18, 23].

We have shown that in some cases, difference rewards and coordination graphs can complement each other and synergize to solve a complex problem. However, at least for the rover domain we studied, we have found that in most cases we only need one, not both coordination techniques. It is a matter of future work to study domains in which both methods are necessary for coordination.

There are several possibilities for future work that build upon our research here. First of all, while wire-fitting performed very well for us, we found it fairly slow compared to an implementation in a standard discrete grid world. In addition, wire-fitting requires many parameters to be tuned by the researcher, which slows im-

	$N = 1$	$N = 3$
Congested (center start)	(Figure 3) D does well CGs under-perform	(Figure 4) D does well CGs do well
Uncongested (scattered start)	(Figure 5) L or D does well CGs are optional	(Figure 6) L or D does well CGs do well

Table 1: Summary of results. In congested domains, difference rewards perform well, but are unnecessary in uncongested domains. Coordination graphs do well when $N = 3$, requiring close coordination, but are unnecessary or harmful when $N = 1$.

plementation and requires a great deal of testing to find a good set of parameters. Alternative strategies for reinforcement learning in continuous domains exist [24] and it would be useful to test one or more of these to see if learning could be performed any faster.

Of course, coordination graphs and difference rewards are not the only coordination methods available. Assignment-based decomposition [17] is a hierarchical coordination technique that has successfully been applied together with coordination graphs, but is one example among many. It would be useful to extend the comparisons performed in this paper to other such coordination techniques. The main difficulty with doing so is finding a domain in which multiple coordination techniques can be usefully applied: most coordination techniques are domain-specific.

9. REFERENCES

[1] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi Agent Systems*, 17(2):320–338, 2008.

[2] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[3] L. Baird and H. Klopf. Reinforcement learning with high-dimensional continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base, 1993.

[4] W. L. Baker and J. A. Farrell. An introduction to connectionist learning control systems. In D. A. White and D. A. Solfge, editors, *Handbook of Intelligent Control*, pages 35–63. Van Nostrand Reinhold, New York, 1992.

[5] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Athena scientific optimization and computation series. Athena Scientific, 1996.

[6] C. Gaskett. *Q-learning for robot control*. Australian National University, 2002.

[7] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS '01: Proceedings of the Neural Information Processing Systems*, pages 1523–1530. MIT Press, 2001.

[8] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *ICML '02: Proceedings of the 19st International Conference on Machine Learning*, San Francisco, CA, July 2002. Morgan Kaufmann.

[9] C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI '02: Proceedings of the 8th National Conference on Artificial Intelligence*, pages 253–259, Edmonton, Canada, July 2002.

[10] M. E. Harmon and L. C. B. Iii. Residual advantage learning applied to a differential game. In *In Proc. of the International Conference on Neural Networks, Washington D.C*, 1995.

[11] M. Knudson and K. Tumer. Robot coordination with ad-hoc team formation (extended abstract). In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*, Toronto, Canada, May 2010.

[12] J. R. Kok, P. Jan, H. Bram, and B. N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *In Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIGâĂŹ05*, pages 29–36, 2005.

[13] J. R. Kok and N. A. Vlassis. Sparse cooperative Q-learning. In R. Greiner and D. Schuurmans, editors, *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, pages 481–488, Banff, Canada, July 2004. ACM.

[14] J. R. Kok and N. A. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *J. Mach. Learn. Res.*, 7:1789–1828, 2006.

[15] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. *Machine Learning and Knowledge Discovery in Databases*, 5211:656–671, 2008.

[16] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[17] S. Proper and P. Tadepalli. Solving multiagent assignment markov decision processes. In *AAMAS '09: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (to appear)*, pages 681–688, 2009.

[18] S. Proper and K. Tumer. Modeling difference rewards for multiagent learning. In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-12)*, Valencia, Spain, June 2012. (to appear).

[19] C. Roth, M. Knudson, and K. Tumer. Agent fitness functions for evolving coordinated sensor networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Dublin, Ireland, July 2011.

[20] A. Sherstov and P. Stone. Three automated stock-trading agents: A comparative study. In *Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems (AMEC 2004), Lecture Notes in Artificial Intelligence*, pages 173–187. Springer Verlag, Berlin, 2005.

[21] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 2005.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. MIT Press, 1998.

[23] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 330–337, Honolulu,HI, May 2007.

[24] H. van Hasselt and M. Wiering. Reinforcement learning in continuous action spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007 (ADPRL 2007)*, pages 272–279, April 2007.

[25] M. Wainwright, T. Jaakkola, and A. Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166, 2004.

[26] M. P. Wellman, S.-F. Cheng, D. M. Reeves, and K. M. Lochne. Trading agents competing: Performance, progress, and market effectiveness. *IEEE Intelligent Systems*, 18(6):48–53, November/December 2003.

[27] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, pages 239–269, 2003.

Combining Difference Rewards and Hierarchies for Scaling to Large Multiagent Systems

Chris HolmesParker
Oregon State University
442 Rogers Hall
Corvallis, OR 97331-6001
holmespc@onid.orst.edu

Kagan Tumer
Oregon State University
426 Rogers Hall
Corvallis, OR 97331-6001
kagan.tumer@oregonstate.edu

ABSTRACT

Coordinating the actions of agents in multiagent systems presents a challenging problem, especially as the size of the system is increased and predicting the agent interactions becomes difficult. Many approaches to improving coordination within multiagent systems have been developed including organizational structures, shaped rewards, coordination graphs, heuristic methods, and learning automata. However, each of these approaches still have limitations with respect to scalability. The goal of this paper is to combine two such coordination mechanisms (difference rewards and hierarchical organization) to improve scalability. We combine difference rewards and hierarchical organizations in the Defect Combination Problem (DCP) with 10,000 sensing agents. We show that combining these techniques results in significantly improved performance and robustness compared to either approach individually. In particular, we show that combining hierarchical organization with difference rewards can improve both coordination and scalability by decreasing information overhead, structuring agent-to-agent connectivity and control flow, and improving the individual decision making capabilities of agents.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed Systems

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

Multiagent Systems, Coordination, Hierarchical Organization, Teams, Shaped Rewards

1. INTRODUCTION

Coordinating the behavior of agents in multiagent systems such that they collectively optimize a system level objective is a complex control task. Problems of scaling (number of agents in the thousands to tens of thousands), information handling (agents have limited computing capabilities), and robustness (unreliable components) make methods developed for small multiagent systems comprised of reliable devices inadequate [12, 16]. A number of approaches have been presented to address these issues including organizational structures, shaped rewards, learning automata, and coordination graphs [6, 10, 15, 17, 19]. Although significant progress has been made towards improving coordination and scalability with each of these methods individually, relatively little work has focused on leveraging the complementary benefits of these approaches. We propose combining two of these methods (hierarchical organization and reward shaping) together in such a way that the benefits of each approach are mutually beneficial to system performance. In this work, we show that combining hierarchical organization to reduce each agent's communication overhead with reward shaping techniques that attempt to make optimal use of local information can improve coordination and scalability more than either method achieves independently.

Reward shaping has been shown to drastically improve coordination, scalability, and performance in multiagent systems [1]. The specific shaped rewards studied in this work are based upon the difference reward structure, which has been shown to be robust to scaling in a number of domains including air traffic control, rover navigation, and distributed sensor networks [1, 16]. Difference rewards were designed to promote coordination and scalability by filtering the information each agent receives, extracting only information relevant to each agent specifically. However, as scaling increases, the amount of information each agent must process increases, reducing the effectiveness of the filter provided by difference rewards. We address this shortcoming by introducing hierarchical organization into the system, which reduces the amount of information each agent must receive and process (to our knowledge no one has combined difference rewards with hierarchical organization).

Hierarchies have shown a lot of promise in decreasing information sharing and processing requirements, improving robustness, and increasing performance in large multiagent systems [6, 11]. These structures focus primarily upon organizing the control flow and reducing information sharing and processing overheads, to reduce the coordination complexity between agents in the system [6]. Controlling these factors alone is not always enough to achieve good system performance. The underlying decision making process of each agent also heavily impacts the system performance. In this work, we combine hierarchies with learning agents using shaped rewards which promote good agent decision making.

Although both hierarchical organization and reward shaping methods have been heavily researched, relatively little work has been done to demonstrate the complementary nature of these two approaches. Generally speaking, hierarchies establish the system control flow and reduce the amount of information that each agent must receive and process [6]. Shaped rewards on the other hand attempt to optimize each agent's decision making given that information

[16]. We demonstrate the complementary nature of these approaches in the Defect Combination Problem (DCP) introduced in [2].

The key contributions of this paper are to:

- Combine difference rewards (which promote coordination) and hierarchical organization (which defines the roles of agents and agent-to-agent connectivity) to reduce information sharing and processing requirements, improve coordination, and improve scalability in a multiagent system.

- to show the robustness of combining difference rewards and hierarchal organization to agent failures.

The remainder of this paper is organized as follows. Section 2 provides background material on hierarchical systems, reward shaping, and the Defect Combination Problem (DCP). Section 3 describes the two variations of the Defect Combination Problem. Section 4 describes the learning algorithms, rewards, and hierarchical organization used in this work. Section 5 contains experimental results, empirically demonstrating the benefits of coupling hierarchies and shaped rewards. Finally, Section 6 provides a discussion of this work.

2. BACKGROUND AND RELATED WORK

The hierarchical organization of a multiagent system can be defined as the collection of roles, relationships, and authority structures which govern its behavior [6]. The structure of a hierarchy guides how its members interact with one another, influencing authority relationships, data flow, resource allocation, coordination patterns, and other system characteristics [5]. Hierarchies have been shown to improve system performance in a number of domains including distributed sensor networks, autonomous aerial vehicle coordination, and rover coordination [6, 7, 20]. In many cases, hierarchical organization reduces coordination complexity and increases system level performance by providing an explicit structure and control flow [6, 7]. In this work, we use a simple 2-layer hierarchy to decrease coordination requirements by governing the agent-to-agent interactions for agents within the DCP domain.

Reward shaping is the practice of replacing an agent's reward function with an alternative reward that changes its learning [16]. Frequently, reward shaping is used to improve system performance or to make a problem easier to solve [1, 4]. Reward shaping has been used to increase performance by speeding up convergence rates and improving coordination in problems involving reinforcement learning [1, 18]. Shaped rewards provide agents with more specific feedback, but can result in severely suboptimal policies if not designed properly [4]. In this work, we focus on difference rewards, which are designed to be aligned with the system objective and to effectively filter the information each agent receives to improve decision making [1, 9]. Unfortunately, this filter has limited scalability and there comes a point beyond which as scaling increases agents still receives a noisy learning signal.

Previous work involving the Defect Combination Problem (DCP) included [2], where statistical physics was utilized to derive the theoretical optimal performance of an N sensor system and the corresponding ratio of active sensors, but it did not include a non-exhaustive search method for finding the actual subset of sensors to use. Using learning agents with difference rewards in a nonhierarchical setting was proposed as a method for finding a good subset of devices in [16]. In that work, difference rewards were shown to improve system performance in the DCP in a nonhierarchical setting involving 1000 sensors [16]. However, as our work shows, difference rewards alone are not sufficient to address the increased coordination complexities and increased signal noise present when scaling increases. To address this shortcoming, we couple difference rewards with hierarchical organization which restricts the amount of information each agent in the system receives and reduces the agent-to-agent coordination complexity.

2.1 Difference Rewards

In this work, we consider difference rewards which are shaped rewards of the form [1]:

$$D_j \equiv G(z) - G(z - z_j) \ , \tag{1}$$

where G is the system objective, z is the complete system state vector, and z_j is the action of agent j. Intuitively this allows the second term of the difference reward to evaluate the performance of the system without agent j and therefore D evaluates the agent's individual contribution to the system performance.

There are two key advantages to using D: First, because the second term removes a significant portion of the impact of other agents in the system, it provides an agent with a "cleaner" signal than G [1, 16]. Second, because the second term does not depend on the actions of agent j, any action by agent j that improves D, also improves G (the derivatives of D and G with respect to j are the same) [1, 16].

We also consider the Estimated Difference Reward (EDR) which is given by:

$$EDR_j \equiv G(z) - E_{z_j}[G(z)|z_{-j}] \tag{2}$$

where $E_{z_j}[G(z)|z_{-j}]$ gives the expected value of G over the possible actions of agent j. Because this term does not depend on the immediate actions of j, this utility is still aligned with G [16]. Furthermore, because it removes noise from an agent's private utility, EDR yields far better learnability than does G [16]. This noise reduction is due to the subtraction which (to a first approximation) eliminates the impact of states that are not affected by the actions of agent j. The major difference between EDR and D is in how they handle z_j. EDR provides an estimate of agent j's impact by sampling all possible actions of agent j whereas D simply removes agent j from the system.

3. DEFECT COMBINATION PROBLEM

Many real world sensing applications require large sets of disparate sensing devices to coordinate their actions in order to collectively optimize their network attenuation, coverage areas, and sensing schedules [3, 13, 18]. In the domain used in this work, a set of 10,000 sensing devices must coordinate their sensing schedules in order to optimize their aggregated attenuation. This work focuses on the Defect Combination Problem (DCP) domain introduced in [2]. This problem assumes that there exists a set of imperfect sensors $\mathbf{X}$ which have constant attenuations due to manufacturing defects or imperfections. Each of the sensors x_i has an associated attenuation a_i (which can be positive or negative) in its reading, such that if it is taking a measurement of A (actual

value) it measures $A + a_i$ where a_i is the device's individual error. The problem then becomes how to best choose a subset of the $\mathbf{X}$ sensors that minimizes the aggregated attenuation of the combined readings:

$$G = \frac{\left|\sum_{i=1}^{N} n_i a_i\right|}{\sum_{i=1}^{N} n_i} \quad (3)$$

where G is the aggregated attenuation of the combined sensor readings, a_i is the attenuation of a particular sensor i, N is the number of sensors, and $n_i \in \{0, 1\}$ based upon whether the sensor chooses to be "on" or "off".

This is an NP-complete optimization problem [2, 16] and simply choosing the single sensor with the best attenuation is an inadequate solution, as is choosing the best K sensors ($1 \leq K \leq N$). To illustrate this, consider the case where there are 6 sensing devices whose attenuations are $a_1 = -0.19$, $a_2 = 0.54$, $a_3 = 0.1$, $a_4 = -0.14$, $a_5 = -0.05$, and $a_6 = 0.21$. Choosing only the best sensor a_5 would yield an aggregated attenuation of $|0.05|$, while choosing sensors a_3, a_4, and a_5 will yield an aggregated attenuation of $|0.03|$, which is better than the single best sensing device a_5 alone. This is still not the optimal solution in this 6 sensor case however, as combining sensors a_1 and a_6 results in an aggregated attenuation of $|0.01|$. In this problem, individual sensors acting independently without coordinating their actions can drastically decrease the system performance, consider the case where sensors a_1 and a_6 are turned on in conjunction with sensor a_2, the aggregated attenuation jumps to from $|0.01|$ to $|0.18|$. Finding good solutions requires a great deal of coordination between sensors, as any one sensor can heavily impact the system performance.

4. AGENTS AND COORDINATION

In this domain, we used a multiagent approach in which each agent was an ϵ-greedy reinforcement learner which used a standard value update [14]:

$$V_{new}(a) = V_{old}(a) + \alpha(R - V_{old}(a)) \quad (4)$$

where a is the agents' action selection, R is the reward received for taking action a, α is the learning rate, and V is the value associated with taking action a.

4.1 Rewards

Throughout this work, agents receive learning signals via three different reward structures: global, difference, and estimated difference rewards. Global rewards provide agents with a learning signal that is equivalent to the system performance (in the team-based experiments, agents receive rewards based upon their teams performance). Global rewards are in-line with the system objective (i.e. if agents maximize their global rewards, they concurrently optimize the system objective), but they provide agents with a noisy learning signal. This is because all agents receiving a global reward signal get the same feedback regardless of their actions, meaning that they may receive a good reward for taking a poor action, or a bad reward for taking a good action (their reward is coupled to the reward of all other agents). Difference rewards address this shortcoming by filtering the noise off of the global reward signal and provide agents with

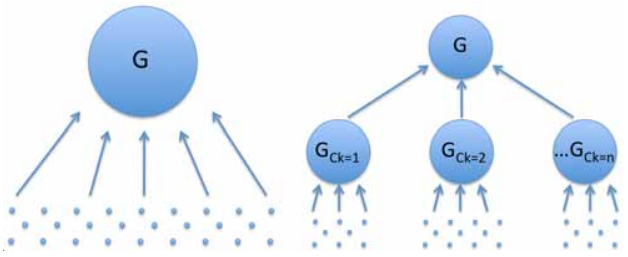


Figure 1: When no teams or hierarchical organization is present (left), agents are required to coordinate directly with all other agents to optimize the global objective G. We reduce this coordination requirement by adding in a two-layer hierarchical structure (right). Here, sensing agents are partitioned into separate teams and coordinate to optimize their team objective G_{c_k} (Equation 5). Then, a control agent is assigned over each team, and the control agents coordinate to optimize the system level objective G (Section 4.2).

specific feedback on how their actions impacted the system performance (Section 2.1).

Here, we derive the difference reward for the DCP problem when no hierarchies or teams are present.[1] Here, each agent is required to coordinate directly with all other agents in the system. The difference reward D_j for agent j is derived by combining Equations 1 and 3:

$$D_j = \begin{cases} \frac{\left|\sum_{i=1}^{N} n_i a_i\right|}{\sum_{i=1}^{N} n_i} - \frac{\left|\sum_{i\neq j}^{N} n_i a_i\right|}{\sum_{i\neq j}^{N} n_i}, & \text{if } n_j = 1 \\ 0, & \text{if } n_j = 0 \end{cases}$$

In the DCP, D_j only provides agents with a clear learning signal if $n_j \neq 0$. If an agent chooses to be turned "off" ($n_j = 0$), it receives a reward of 0 (it had no impact on the system) whether that action was good or bad for the system performance, meaning half of the actions an agent takes will effectively be random as far as the system performance is concerned.

Next, we derived the Estimated Difference Reward (Section 3) for the DCP problem by combining Equations 2 and 3. Consider the case where the the probabilities are equivalent for each action "on" and "off", $P_{n_j=0} = 0.50$ and $P_{n_j=1} = 0.50$, EDR_j becomes:

$$EDR_j = \begin{cases} 0.50\frac{\sum_{i\neq j}^{N} n_i a_i - a_j}{\sum_{i\neq j} n_i - 1} - 0.50\frac{\sum_{i=1}^{N} n_i a_i}{\sum_{i=1} n_i}, & \text{if } n_j = 1 \\ 0.50\frac{\sum_{i\neq j}^{N} n_i a_i + a_j}{\sum_{i\neq j} n_i + 1} - 0.50\frac{\sum_{i=1}^{N} n_i a_i}{\sum_{i=1} n_i}, & \text{if } n_j = 0 \end{cases}$$

EDR_j provides a clear learning signal: if it is positive, the action taken by agent j was beneficial to system performance, and if EDR_j is negative, the action was harmful to

[1] D and EDR rewards can be derived for the team-based sensing agents and the hierarchical control agents (Section 4.2), but are excluded here for brevity.

Algorithm 1 Instantiate Hierarchical Organization

1: Randomly partition N sensing agents into M equal teams of size C
2: Assign a "local" team objective to each individual team (Equation 5)
3: Assign one control agent per team
4: Assign control agents an objective (Equation 6)

system performance. Agents trying to maximize EDR_j will implicitly maximize system performance simultaneously.

4.2 Teams and Hierarchical Organization

Though the simplest way to organize a multiagent system is to have no teams and no hierarchical organization, there comes a point beyond which as agent scaling increases, coordination becomes too complex for a nonhierarchical system to be effective. We address this shortcoming by incorporating teams and hierarchical organization into the system.

Uncoordinated Teams

In contrast to the standard DCP problem approach in which all N agents observed each other and acted as a single group, we introduced a team-based approach. Here, we randomly partitioned the N agents into teams of C agents, where each agent could only be a member of a single team. Random teams were assigned due to the NP-complete nature of the problem, the computational expense of intelligently assigning teams would be too high. Since there are no obvious ways of decomposing the system objective with respect to the teams, each team is treated as a separate DCP. The goal of each team is to optimize the aggregated attenuation of its C sensing devices (Equation 5). Agents within each team attempted to optimize their aggregated team attenuation:[2]

$$G_{c_k} = \frac{|A_{c_k}|}{N_{c_k}} = \frac{\left|\sum_{i=1}^{C} n_i a_i\right|}{\sum_{i=1}^{C} n_i} \tag{5}$$

where G_{c_k} is the objective of team c_k, A_{c_k} is the aggregated attenuation of team c_k, N_{c_k} is the total number of active devices in team c_k, C is the number of agents in each team, $n_i \in \{0, 1\}$ depending on whether sensor i chose to participate in sensing, and a_i is the attenuation of sensor i. A team approach is advantageous because it can reduce the coordination complexity of individual agents within the system by reducing the number of devices with which each agent has to communicate. Although this team formation approach reduces the coordination complexity and information overhead of the system, it may not lead to good system performance. This is because each team acts to optimize its own independent team objective G_{c_k}, without taking into account how its actions impact the overall system performance. However, we test this method because each team of C agents will have relatively low aggregated attenuations and by statistically averaging many teams with low aggregation an even lower attenuation may result.

[2]In the team-based experiments (Sections 5.2-5.4), sensing agents' global, difference, and estimated difference rewards were based upon the team objective (Equation 5).

Algorithm 2 Learning in Hierarchically Coordinated Teams

```
Instantiate Hierarchical Organization (Algorithm 1)
for Run = 1 → Run_Max do
  for T = 1 → Tmax_teams do
    Sensing agents learn (Section 4)
    Control agents fix policies to "on"
  end for
  for T = 1 → Tmax_control do
    Sensing agents use learned policies (fixed)
    Control agents learn (Section 4)
  end for
end for
```

Hierarchically Coordinated Teams

As seen in the previous section, creating teams can decrease agent-to-agent coordination complexity and reduce information overheads. However, creating individual teams can be harmful to system performance if these teams fail to coordinate their actions well. We address this problem by superimposing a hierarchical control layer on top of each team (Algorithm 1). In this setting, individual teams are treated as though they were a single sensor and each "team sensor" is controlled by a single control agent. This results in a 2-layer hierarchical network structure (Figure 1, *right*) which reduces agent-to-agent coordination complexity and information overhead within the system. Agents in the bottom layer attempted to optimize the attenuation of their individual teams for a single reading (Equation 5), while the control agents dictated whether or not each team would participate in the aggregated system reading. Thus, instead of turning "on" or "off" like the sensing agents, the control agents each turned a team on or off (Algorithm 2). The top level control agents coordinated in order to optimize the system objective (Equation 3):

$$G_H = \frac{\left|\sum_{k=1}^{K} A_{c_k} n_k\right|}{\sum_{k=1}^{K} N_{c_k} n_k} = \frac{\left|\sum_{i=1}^{N} n_i a_i\right|}{\sum_{i=1}^{N} n_i} \tag{6}$$

where G_H is the objective of the control agents in the hierarchical system for the standard DCP, A_{c_k} is the aggregated attenuation of team c_k, N_{c_k} is the total number of active devices in team c_k, K is the total number of teams (N/C), $n_k \in \{0, 1\}$ depending on whether the agent i governing team k chose to turn team c_k *on* or *off*.[3]

5. EXPERIMENTS AND RESULTS

In this paper, we conduct the following set of experiments:

1. The DCP with no teams.
2. The DCP with uncoordinated teams.
3. The DCP with hierarchically coordinated teams.
4. The DCP with failures using hierarchically coordinated teams.

[3]The goal of the control agents G_H is to combine the team attenuations A_{C_k} and participations N_{C_k} in such a way that they optimize the system level attenuation G

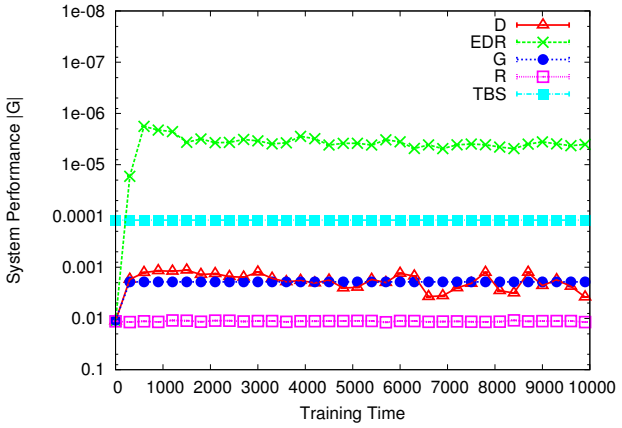


Figure 2: 10,000 sensors Defect Combination Problem (no hierarchies). Agents must choose whether to be "on" or "off". As seen, agents using EDR obtain significantly better aggregated attenuation than the best single sensor TBS (Section 5).

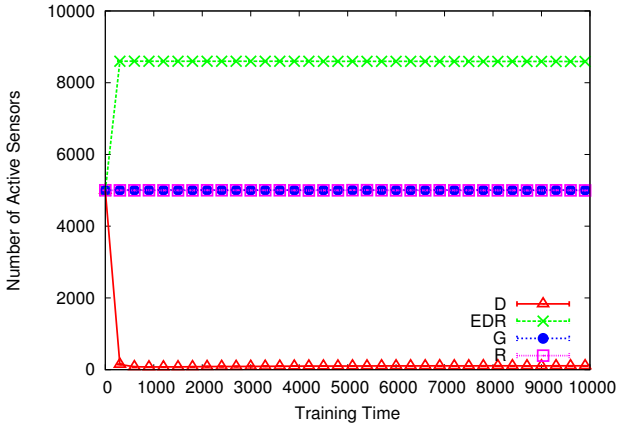


Figure 3: Number of sensors used when 10,000 sensing agents solved the standard DCP. G and R both use nearly the optimal number of sensors 50% [2], but achieve poor performance. Although D and EDR are both difference rewards, they lead to very different policies. D uses few sensors, while EDR uses nearly 80% of the sensing devices.

There were five different types of agents used. The first type of agents coordinate such that only the single-best sensing device was turned on each time step (TBS). Although selecting the best sensor is conceptually simple, it is a centralized algorithm and requires global coordination. Second, we consider the case where the behavior of the agents is completely random (R). The next three types of agents are learning agents attempting to optimize global (G), difference (D), or estimated difference reward (EDR) structures. These rewards were derived separately for the no teams, uncoordinated teams, and hierarchically coordinated teams experiments.

In these experiments at the beginning of each run the attenuations a_i for each agent were drawn from a Gaussian distribution of zero mean and unit variance. All experiments had 10,000 episodes, contained N=10,000 sensing agents (in hierarchical experiments there were also 100 control agents, resulting in 10,100 total agents), were averaged over $r = 1000$ statistical runs, and plotted with the error of the mean $\sigma/\sqrt{r}$ (the error in the mean is plotted in Figures 2-7, but it is so small that it is not visible). The results are statistically significant as we performed a t-test with $p = 0.05$ for all experiments. The learning rate was set to $\alpha = 0.10$ (performance was not overly sensitive to α) and the exploration rate for all agents was set to $\epsilon = 0.01$. All value tables and Q-tables were initialized to zero. For all agents, for the first 20 time steps, learning was turned off and agents chose random action selections. After the first 20 steps, learning was turned on for 60 agents at a time until all of the agents were learning, in the mean time agents who had not been switched on continued performing randomly.[4]

5.1 No Teams

The first experiment shows the performance of agents solving the DCP problem using learning without teams or hierarchical organization. Here, each agent must coordinate directly with all other agents. In this experiment (Figure 2) random action selection R utilizes approximately half of the sensors each time step, but performs poorly since the selection of which sensors is completely random. Similarly, agents using a global reward G turn on approximately half of the sensing devices and make better decisions selecting which sensors to turn on. This results in G outperforming R by nearly an order of magnitude. However, agents using G still have difficulty differentiating the impact of their own actions on their reward signal from the actions of other agents. This is because with G, all agents receive the system performance as their reward signal, regardless of how their own actions impacted the system performance. This makes it difficult for these agents to coordinate their actions, inhibiting system performance.

Difference rewards D and EDR address this shortcoming by effectively filtering out the impact of other agents on an agents' reward signal and accounting for each agent's individual contribution to the system performance. However, as seen in Figure 2, agents using D perform poorly in this experiment. This is because D only provides constructive feedback when the agent elects to be active. Agents using D receive a reward of 0 when they choose to remain off so they do not receive enough feedback to successfully coordinate their actions (Section 4.1). The Estimated Difference Reward EDR reward does not give a 0 reward to an agent for being on or off, instead it gives an estimated value of the agents cumulative impact on the system over time based upon its historic action selections (Section 4.1), resulting in better performance than D in this case (Figure 2).

It is clear from this experiment that the way an agent handles the information it receives drastically impacts the performance. Agents using G, D, and EDR received the exact same information, yet agents using difference rewards were able to significantly outperform agents using a traditional global rewards. Here, both D and EDR reduce the

[4]Allowing all agents to begin learning simultaneously created a "spike" into the system which significantly slowed down learning. The gradual introduction of the learning agents is softens this discontinuity in learning [16].

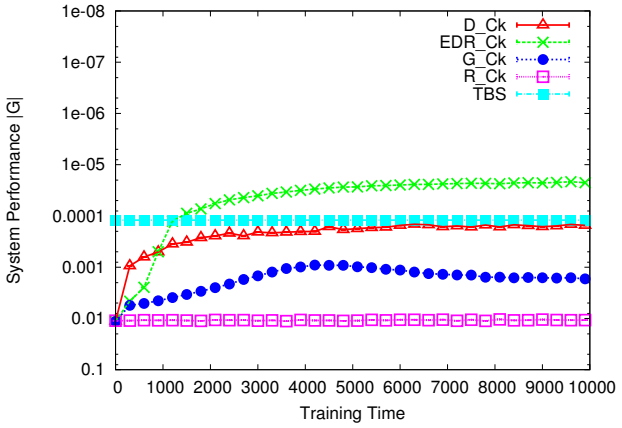


Figure 4: 10,000 sensors DCP with teams of 100 agents (Section 4.2). Teams of agents using D_{c_k} and EDR_{c_k} far outperform those using a standard global reward G_{c_k}. Unfortunately, regardless of the reward used the teams are unable to coordinate together, resulting in poor performance.

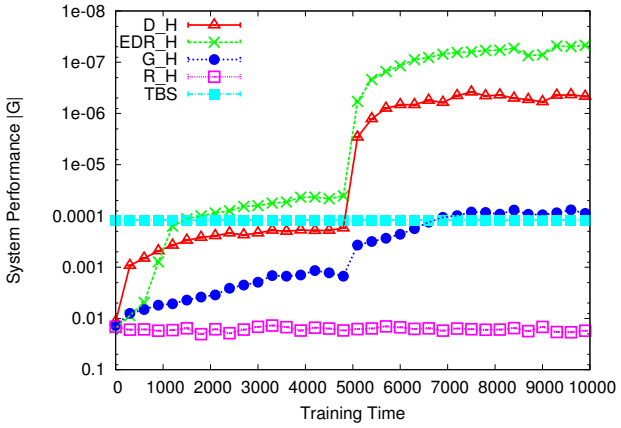


Figure 5: 10,000 sensors DCP with hierarchical teams (Section 4.2). Sensors are divided into teams of 100 and a control agent is placed over each team. The discontinuity at 5000 time steps is due to hierarchical learning (Section 5.3). As seen, D_H and EDR_H outperform all other approaches (Figure 2). The hierarchy dictates the control flow and information overheads, while difference rewards improve agent decision making.

overall coordination complexity for individual agents by filtering much of the noise of other agents' actions from each agent's reward signal (Section 2.1). It is interesting to note that D and EDR arrive at very different policies. Agents using D typically learn to use around 1% of the overall sensors, while agents using EDR use nearly 80% (Figure 3). Both of these solutions used far from the theoretical optimal number of sensors, which was determined to be 50% in [2]. This tells us that shaped rewards alone may not be enough to maximize system performance for this problem.

5.2 Uncoordinated Teams

Next, we integrate random teams into the problem (Section 4.2). Here, the 10,000 sensing agents are randomly divided into teams of 100, where each agent can only be a member of a single team. The goal of each team G_{c_k} is to optimize the aggregated attenuation of its own 100 sensing agents (Equation 5). Each team acts independently to optimize its own attenuation and there is no coordination between the teams (there are no control agents present to coordinate the actions of the teams together to optimize the system performance G). By creating teams of agents, we effectively reduce the information sharing and processing requirements and coordination complexity for agents within the system. Agents now only need to coordinate with the other sensing devices in their team.

As seen in Figure 4, team based agents using G_{c_k} continue to perform poorly compared to team based agents using difference rewards D_{c_k} and EDR_{c_k} in the standard DCP. Here, the amount of information each agent receives is reduced by two orders of magnitude, but agents using G_{c_k} still don't perform well. This is because although the information overhead is reduced one hundred fold to that of a 100 agent system and it is easier for agents to deduce their individual impact on their rewards, the teams are not working together in an organized way, and are frequently interfering with each other.

Agents using D_{c_k} perform better in this case than they did without teams (Figure 2), primarily because the amount of exploration was increased allowing agents using D_{c_k} to stumble across higher quality solutions more frequently, resulting in more sensors being active. Neither agents using D_{c_k} nor EDR_{c_k} performed as well as agents using EDR difference rewards in the standard case (Figure 2) because teams are not coordinated; each team is individually trying to optimize its own 100 agent team objective G_{c_k} without accounting for how its actions impact the overall system performance G. Here, the agents are effectively optimizing the 100 sensor DCP problem one hundred times and summing the results in a suboptimal manner. The performance could vastly be improved if the teams were allowed to coordinate their actions to mutually benefit system performance. In the next experiment we address this by superimposing hierarchical control agents onto each team.

5.3 Hierarchically Coordinated Teams

Finally, we implemented a 2-layer hierarchy into the DCP (Section 4.2). This hierarchical approach assigns a control agent for each team which determines how the team participates in sensing. This approach addresses the two key issues that inhibited the performance in the two previous nonhierarchical experiments. First, it reduces the agent-to-agent coordination complexity by adding structure and organization to the system. Agents now only need to coordinate with other agents in their team (agents in the top level of the hierarchy form their own team). The presence of a control layer on top of the teams solves the team coordination issue from Experiment 2 (Figure 4). Secondly, the information sharing and processing requirements are reduced by approximately one hundred fold for all agents within the system. Here, individual sensing agents continue to optimize their local team objective G_{c_k}, while the hierarchical agents directly optimize their own reward G_H (which is the system

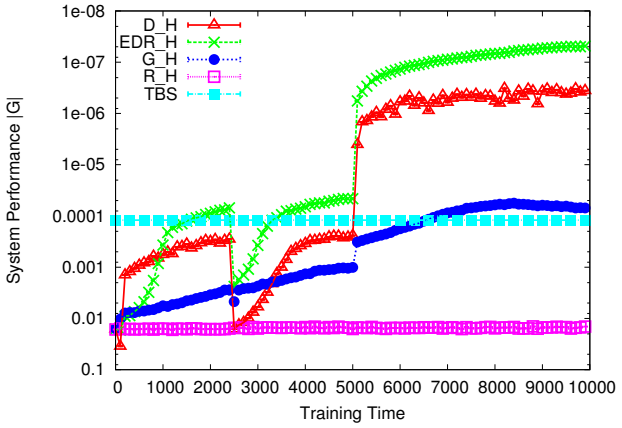


Figure 6: 10,000 sensors solving the DCP with hierarchical teams (Section 4.2). 10% of the bottom layer agents fail after 2500 time steps. The discontinuity at 5000 time steps is due to hierarchical learning (Section 5.3). As seen, when 10% of the sensing devices fail, the remaining 90% are able to coordinate to adapt their behavior and recover most of the lost system performance.

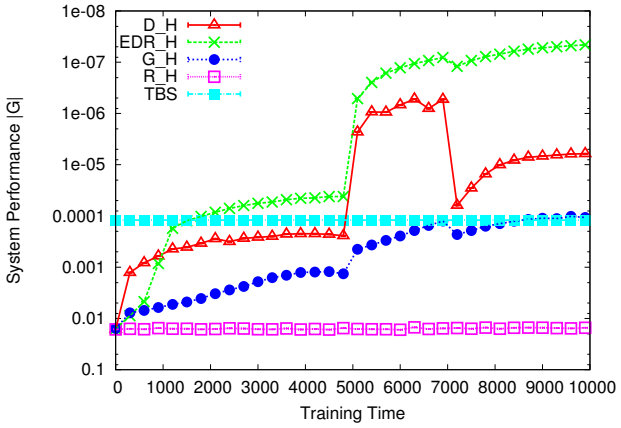


Figure 7: 10,000 sensors DCP with hierarchical teams (Section 4.2). 10% of top layer agents fail after 7500 time steps. The discontinuity at 5000 time steps is due to hierarchical learning (Section 5.3). As seen, a combination of shaped rewards and hierarchies proves robust to top layer failures. Agents using D_H and EDR_H far outperform agents using a standard global reward G_H.

objective G in these experiments).

Here, the top and bottom level teams were trained separately (Algorithm 2). First, the bottom level agents learned for 5000 time steps while the top level agents took random actions and did not learn. Then, the bottom level agents' learning was turned off and they followed their learned policies while the top layer agents' learned for the next 5000 time steps. This was done for two key reasons: 1) the actions of the agents in the bottom layer were independent of agents in the top layer, and 2) the top layer could not make optimal decisions until the actions of the bottom layer were set. The separate training of the top and bottom levels of the hierarchy is responsible for the learning spike at 5000 time steps in Figures 5-7.

As seen in Figure 5, hierarchical organization benefits agents using the G, D, and EDR reward structures. Agents optimizing global rewards (G_H) achieve nearly an order of magnitude better performance when a hierarchical structure was added to the system. This is because, in addition to reducing the information overhead, the hierarchical structure allows teams to coordinate their actions together to improve system performance. Despite the benefits from the addition of a hierarchical structure, agents using a traditional global reward G_H were still unable to achieve the same performance as agents using shaped rewards without a hierarchical structure (Figures 2 and 5).

This shows that simply adding hierarchical organization to the system may not be enough to maximize system performance. Adding a hierarchy reduces coordination complexity and information overheads, but it does not attempt to optimize agent decision making given the information each agent receives. This is why agents using D and EDR rewards in a nonhierarchical setting where the information overhead and coordination complexity remain high still outperform a traditional global reward G_H in a hierarchical structure. Agents using a combination of a hierarchical structure and shaped rewards outperform nonhierarchical approaches by orders of magnitude (Figure 5), which supports the fact that hierarchical structures and shaped rewards offer complimentary benefits in large scale multiagent systems. Hierarchical organization dictates the control flow and reduces the information overheads, while shaped rewards improve agent decision making given the information each received.

5.4 Hierarchically Coordinated Teams with Failures

Now that we have established that a combination of shaped rewards and hierarchical organization can dramatically improve the performance of large multiagent systems, we want to demonstrate the robustness of our approach to component failures. In the context of this experiment an agent (controller or sensor) getting stuck *on* will constitute a failure. Since failures in the top and bottom layers of the hierarchy may impact the system differently, we perform a separate experiment for each case. In the first experiment 10% of the bottom level sensors fail after 2500 time steps (Figure 6), while the other sensors continue learning. In the next experiment, 10% of the top level sensors fail at time step 7500, while the others continue learning (Figure 7). In both cases, the top level hierarchical agents do not begin learning until time step 5000 (Algorithm 2).

As seen in Figure 6, a combination of difference rewards and hierarchies is robust to failures within each individual team. Here, 10% of the agents in each individual team fail after 2500 time steps and the remaining sensing devices need to coordinate their actions with these defective devices in order to recover system performance. Due to the reduced coordination requirements imposed by the hierarchical organization, team-based sensing agents only need to coordinate their actions with 100 other agents. These reduced coordination requirements coupled with agents using difference rewards enable them to coordinate in order to regain

the performance lost due to failures. In the next experiment (Figure 7), 10% of the control agents failed, each one impacting an entire team of sensing agents. However, since the individual teams maintained relatively low attenuations, when control agents failed and remained on, the remaining control agents were still able to coordinate their actions in order to achieve good performance even in the presence of failures.

6. DISCUSSION

In very large multiagent systems, agents frequently encounter two key problems: 1) increased coordination requirements, and 2) increased information sharing and processing requirements. We address both of these issues by combining two well known coordination mechanisms, hierarchical organization and shaped difference rewards. Hierarchies dictate the control flow and information handling, lowering the 'per agent' coordination complexity in the system. On the other hand, difference rewards act to optimize information processing, serving as an information filter (extracting only the specific information relative to a particular agent) and promote agent coordination. Our results show that a combination of shaped difference rewards and hierarchical organization can improve coordination, scalability, and performance in large multiagent systems. Combining these approaches led to approximately three orders of magnitude improvement over either method individually in the DCP, and shows promise for other large multiagent domains including sensor networks, aerial vehicle coordination, and network traffic management.

This work showed the potential advantages of combining coordination algorithms in ways that leverage their benefits. Although many coordination algorithms exist throughout the literature, they have primarily been used independently and relatively little work has focused on the performance increases attainable by combing them. Future work would include finding new combinations of coordination algorithms that can be used to improve both agent-to-agent coordination as well as overall scalability. In particular, selecting coordination mechanisms that are synergistic and not only work well together but actually magnify each others benefits. This could be done by defining a set of metrics and characteristics of coordination mechanisms, which could then be used to determine when coordination mechanisms may be merged together to improve performance.

7. REFERENCES

[1] A. Agogino and K. Tumer. Analyzing and visualizing multi-agent rewards in dynamic and stochastic domains. In *Journal of Autonomous Agents and Multi-Agent Systems*, pages 320–338, 2008.

[2] D. Challet and N. Johnson. Optimal combination of imperfect objects. *Physics Review Letters 89*, 2002.

[3] A. Farinelli, A. Rogers, and N. Jennings. Maximising sensor network efficiency through agent-based coordination of sense/sleep schedules. *Workshop on Energy in Wireless Sensor Networks*, 2008.

[4] M. Grzes and D. Kudenko. Online learning of shaping rewards in reinforcement learning. *Neural Networks*, 23, 2010.

[5] S. Hayden, C. Carrick, and Q. Yang. A catalog of agent coordination patterns. *In Proceedings of the 3rd Annual Conference on Autonomous Agents*, 1999.

[6] B. Horling and V. Lesser. A survey of multiagent organizational paradigms. In *Knowledge Engineering Review*. Cambridge University Press, 2005.

[7] B. Horling, R. Mailler, and V. Lesser. A case study of organizational effects in a distributed sensor network. *In Proceedings of the International Conference on INtelligent Agent Technology*, 2004.

[8] E. Howley and J. Duggan. Investing in the commons: A study of openness and the emergence of cooperation. *Advances in Complex Systems*, 14, 2011.

[9] M. Knudson and K. Tumer. Coevolution of heterogeneous multi-robot teams. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2010.

[10] J. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 2006.

[11] N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich. Automatic discovery and transfer of maxq hierarchies. *In Proceedings of the 25th International Conference on Machine Learning*, 2008.

[12] L. Panait and S. Luke. Cooperative multi-agent learning - the state of the art. *In the Journal of Autonomous Agents and MultiAgent Systems*, 2005.

[13] A. Rogers, A. Farinelli, and N. Jennings. Self-organising sensors for wide area surveillance using the max-sum algorithm. *Lecture Notes in Computer Science. Self-Organizing Architectures*, 2010.

[14] R. Sutton and A. Barto. *Reinforcement Learning An Introduction*. MIT Press, Cambridge, MA, 1998.

[15] M. Tambe, E. Bowring, H. Jung, G. Kaminka, R. Maheswaran, J. Marecki, P. Modi, R. Nair, S. Okamoto, J. Pearce, P. Paruchuri, D. Pynadath, P. Scerri, N. Schurr, and P. Varakantham. Conflicts in teamwork - hybrids to the rescue. *In Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems*, 2005.

[16] K. Tumer. Designing agent utilities for coordinated, scalable, and robust multiagent systems. In P. Scerri, R. Mailler, and R. Vincent, editors, *Challenges in the Coordination of Large Scale Multiagent Systems*. Springer, 2005.

[17] P. Vrancx, K. Verbeeck, and A. Nowe. Decentralized learning in markov games. *IEEE Transations on Systems, Man, and Cybernetics - Part B: Cybernetics*, 38(4), August 2008.

[18] S. Williamson, E. Gerding, and N. Jennings. Reward shaping for valuing communications during multi-agent coordination. *In Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[19] Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara. An integrated token-based algorithm for scalable coordination. *In Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems*, 2005.

[20] C. Zhang, S. Abdallah, and V. Lesser. Integrating organizational control into multi-agent learning. *In Procceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

Reinforcement Learning of Throttling for DDoS Attack Response

Kleanthis Malialis
Department of Computer Science,
University of York, UK
malialis@cs.york.ac.uk

Daniel Kudenko
Department of Computer Science,
University of York, UK
kudenko@cs.york.ac.uk

ABSTRACT

Distributed denial of service attacks (DDoS) constitute a serious and evolving threat in the current Internet. The most common type of these attacks is the flooding DDoS attack, which is designed to exhaust computer or network resources. Router throttling is a popular approach in the battle against these attacks, which views the flooding DDoS problem as a resource management or congestion problem. In this paper, we introduce a learning throttling approach which provides a highly adaptive response to such attacks. We compare our proposed approach against two other throttling approaches from the literature. It is shown that our approach effectively mitigates the impact of flooding DDoS attacks, and that it overcomes potential stability and convergence problems that the two throttling approaches suffer from.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security

Keywords

Reinforcement Learning, Network Security, DDoS Attacks, Throttling

1. INTRODUCTION

The increasing adoption of technologies and applications makes computer security a vital and essential part of every organisation. Computer security is based on the principles of confidentiality, integrity and availability; or the CIA triad as they are usually referred to as [11]. Confidentiality ensures that classified information is not exposed to unauthorised individuals. Individuals have also the full control of all the information relating to them. Integrity ensures that information is not modified or destroyed by unauthorised individuals. It also ensures that a system performs its intended functionality. Finally, availability ensures that a system's service is provided in a timely and reliable manner to its legitimate users.

One of the most serious threats in the current Internet is posed by flooding distributed denial of service (DDoS) attacks, which target the availability of a system [10]. A DDoS attack is a highly coordinated attack where the attacker (or attackers) takes under his control a large number of hosts, called the botnet (network of bots), which start bombarding the target when they are instructed to do so. Such an attack is designed to exhaust a server's resources or congest a network's infrastructure, and therefore renders the victim incapable of providing services to its legitimate users.

Research on DDoS defence is focused on three areas, namely, detection, traffic classification and response [8]. Detection aims at identifying the existence of an attack, while classification distinguishes between legitimate and malicious network packets. Response aims at stopping the flood, or reducing it to acceptable levels. DDoS response, among others, include filtering, rate limiting or throttling and traffic redirection. Due to the distributed nature of these attacks, an effective DDoS defence requires a distributed coordinated defence mechanism, where defensive nodes at different network locations cooperate to stop the flood, or mitigate its effectiveness.

Router throttling is a popular method applied by telecommunication companies to regulate traffic during periods of congestion, most commonly known as "traffic shaping" [9]. Extensions of router throttling to tackle the flooding DDoS problem requires these attacks to be viewed as a resource management or congestion problem [13]. Throttling provides an easy, quick and cheap solution against the DDoS threat.

In this paper, we introduce a novel router throttling approach, which applies multi-agent reinforcement learning (MARL). MARL's task is to learn router throttles in order to keep the server under protection in a working condition. It is also responsible for allowing as much legitimate traffic as possible towards the server. Due to the high complexity and multidimensionality of the DDoS problem, MARL creates an automated and effective response against DDoS attacks. The environment is highly dynamic and unpredictable, and our approach provides adaptive behaviours over frequent environmental changes.

We compare our proposed approach against a baseline approach, and a popular throttling approach found in the literature [13]. MARL is shown to overcome stability and convergence problems, which the other two approaches suf-

fer from. It is also shown that it outperforms the baseline approach, and has a similar performance with the second approach. To the best of our knowledge, this is the first time that learning is incorporated in the throttling component of a DDoS defence mechanism.

The organisation of this paper is as follows. Section 2 describes the background of reinforcement learning, DDoS attacks and router throttling. Section 3 presents our proposed approach in great detail. We describe our experiments and present the results in section 4. In section 5 we discuss about the advantages and limitations of our approach. Finally, section 6 provides a conclusion and direction towards our future work.

2. BACKGROUND

This section describes all relevant work upon which this work is based.

2.1 Reinforcement Learning

Reinforcement learning is a paradigm in which an active decision-making agent interacts with its environment and learns from reinforcement, that is, a numeric feedback in the form of reward or punishment [12]. The feedback received is used to improve the agent's actions. Typically, reinforcement learning uses a Markov Decision Process (MDP) as a mathematical model.

An MDP is a tuple $\langle S, A, T, R\rangle$, where S represents the state space, A represents the action space, $T(s, a, s') = Pr(s'|s, a)$ is the transition probability function which returns the probability of reaching state s' when action a is executed in state s, and $R(s, a, s')$ is the reward function which returns the immediate reward r when action a executed in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e. a mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and reward function) are available, this task can be solved using dynamic programming [2].

In most real-world domains, the environment dynamics are not available and therefore the assumption of perfect problem domain knowledge makes dynamic programming to be of limited practicality. The concept of an iterative approach constitutes the backbone of the majority of reinforcement learning algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states, $V(s)$, or state-action, $Q(s, a)$, pairs. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. The SARSA algorithm is such a method [12]. After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \quad (1)$$

where α is the rate of learning and γ is the discount factor. It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'.

The exploration-exploitation trade-off constitutes a critical issue in the design of a reinforcement learning agent. It aims to offer a balance between the exploitation of the agent's knowledge and the exploration through which the agent's knowledge is enriched. A common method of doing so is ϵ-greedy, where the agent behaves greedily most of the time, but with a probability ϵ it selects an action randomly. To get the best of both exploration and exploitation, it is advised to reduce ϵ over time [12].

Applications of reinforcement learning to multi-agent systems typically take one of two approaches; multiple individual learners or joint action learners [5]. The former is the deployment of multiple agents each using a single-agent reinforcement learning algorithm. The latter is a group of multi-agent specific algorithms designed to consider the existence of other agents.

Multiple individual learners assume any other agents to be a part of the environment and so, as the others simultaneously learn, the environment appears to be dynamic as the probability of transition when taking action a in state s changes over time. To overcome the appearance of a dynamic environment, joint action learners were developed that extend their value function to consider for each state the value of each possible combination of actions by all agents.

The consideration of the joint action causes an exponential increase in the number of values that must be calculated with each additional agent added to the system. Typically, joint action learning algorithms have only been demonstrated in trivial problem domains whilst applications in complex systems most often implement multiple individual learners [6]. For these reasons, this work will focus on multiple individual learners and not joint action learners.

To model a multi-agent system, the single-agent MDP becomes inadequate and instead the more general Stochastic Game (SG) is required [4]. A SG of n agents is a tuple $\langle S, A_i, T, R_i\rangle, i \in [1, n]$, where S is the state space, A_i is the action space of agent i, $T(s, A, s') = Pr(s'|s, A)$ is the probability that joint action A in state s will lead to state s', and $R_i(s, a, s')$ is the immediate reward r received by agent i when action a taken in state s results in a transition to state s'.

2.2 Distributed Denial of Service Attacks

DDoS attacks [10] constitute a major and evolving problem in the current Internet. Such an attack targets the availability of computer or network resources, thus not allowing its legitimate users to access resources in an efficient manner, or even make the resources completely inaccessible to them. A DDoS attack is a highly coordinated attack; the strategy behind it is represented by the agent-handler model [10] as shown in figure 1.

The model consists of four elements, the clients, handlers, agents and victim. The handlers (or masters) and the agents

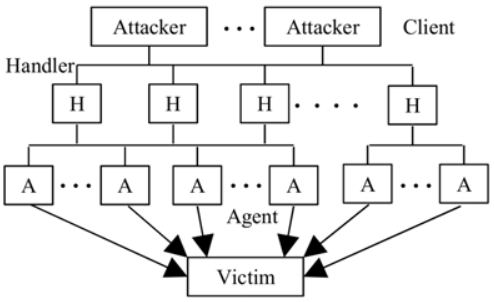


Figure 1: The agent-handler model (from [10])

(or daemons or zombies) are hosts compromised by the attackers. Specifically, the clients install a software on vulnerable hosts to compromise them, thus being able to communicate with and control them. The clients communicate with the handlers, which in turn control the agents in order to launch a DDoS attack. Typically, the users of the agent systems are not aware that their system is involved in a coordinated DDoS attack. Moreover, not all available agents are needed for the attack to take place.

The most common type of DDoS attacks is the flooding attacks, where the attacker starts bombarding the victim by sending large volumes of traffic towards it, thus causing severe congestion to the victim. Research on flooding DDoS defence is focused on three areas, namely, detection, traffic classification and response [8]. Detection aims at identifying whether an attack takes place. Accurate detection occurs when performed in the vicinity of the victim, where the traffic is highly aggregated. Therefore the system can observe a possible performance degradation.

Traffic classification aims at the distinction between legitimate and malicious network packets. Ideal classification occurs when performed near the source, where it experiences a moderate traffic volume. Sophisticated traffic profiling requires advanced statistics gathering, thorough packet inspection and needs to devote many resources. As a consequence, a classification unit can be very expensive and customers may find the cost unaffordable.

Response to the attack aims at stopping the flood or reducing it to acceptable levels. Responding to an attack near the victim causes heavy collateral damage. Collateral damage occurs when legitimate traffic is punished along with the DDoS traffic. Accurate response occurs when performed upstream, that is, close to the sources or in the intermediate (between the sources and the victim). Some locations are more beneficial than others, thus requiring less deployment points.

It's clear that to combat the distributed nature of these attacks, a distributed defence mechanism is necessary, where many defensive nodes, across different locations cooperate in order to stop or reduce the flood [8].

2.3 Router Throttling

Router throttling is a popular method used by telecommunication companies to perform "traffic shaping" [9] during periods of congestion. Extensions of router throttling as a DDoS defence mechanism requires flooding DDoS attacks to be seen as a resource management or congestion problem. We adopt the system model from Yau et al [13].

A network is a connected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. All leaf nodes are hosts and denoted by H. Hosts can be traffic sources, and are not trusted. An internal node represents a router, which forwards or drops traffic received from its connected hosts or peer routers. The set of routers are denoted by R, and they are assumed to be trusted, i.e. not to be compromised.

The set of hosts $H = V - R$ is partitioned into the set of legitimate or good users H_g, and the set of attackers H_a. A leaf node denoted by S represents the victim server. A good user sends packets towards the server S at a rate r_g, and an attacker at a rate r_a. We assume that the attacker's rate is significantly higher than that of a good user, that is, $r_a >> r_g$. This assumption is based on the rationale that if an attacker sends at a similar rate to a good user, then

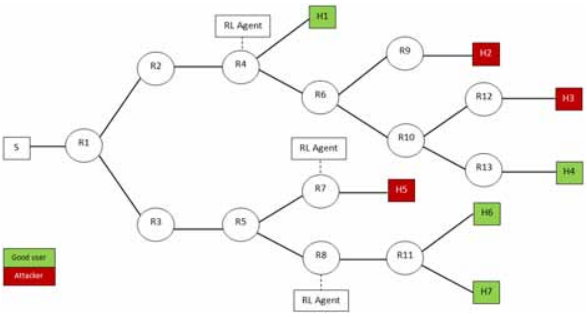


Figure 2: Network topology

the attacker (the client) must recruit a considerably larger number of hosts in order to launch an attack with a similar effect [13].

A server S is assumed to be working normally if its load r is below a specified threshold U_s, that is, $r \leq U_s$. The rate r_g of a good user is significantly lower than the upper boundary i.e. $r_g << U_s$, where U_s can be determined by observing how users normally access the server.

The authors adopt a proactive, server-initiated approach. When a server S operates below an upper boundary Us, it needs no protection. When the server experiences heavy load, it requests from upstream routers to install a throttle. In case the server load r is still over the upper boundary Us, the server asks from upstream routers to increase the throttle. If the server load r drops below a lower boundary Ls, the server asks the upstream routers to relax the throttle. The goal is to keep the server load within the boundaries during a DDoS attack.

Furthermore, the deployment locations are determined by a positive integer k, and are given by $R(k) \subseteq R$. $R(k)$ is defined as the set of routers that are either k hops away from the server, or less than k hops away but are directly attached to a host [13]. Consider for example the network shown in figure 2. It consists of 21 nodes, these are, the victim server denoted by S, 13 routers denoted by $R1 - R13$ and seven hosts denoted by $H1 - H7$, which are traffic sources towards the server. The set $R(4)$ consists of routers $R4, R7, R8$. Router $R4$ is included in the set $R(4)$, although it is 3 hops away, because it is directly attached to the host $H1$. For this reason, router $R6$ is not included in the set.

The authors present a baseline throttle approach in which all routers in $R(k)$ throttle traffic for S, by forwarding only a fraction of the traffic. This approach penalises all routers in $R(k)$ equally, irrespective of whether they are well behaving or not, and is therefore not a fair approach. The authors propose a fair throttle approach which installs a uniform leaky bucket rate at each router in $R(k)$. The algorithm achieves level-k max-min fairness [13]; a form of equal-share fairness [8], where the parent divides its bandwidth allocation equally among all its children [1]. Both approaches have their drawbacks:

1. *Stability*: System oscillations in order to settle the server load to a desired level within the lower L_s and upper U_s boundaries, can cause stability, and therefore

[1]The alternative design is proportional-share, where the parent takes into consideration its children needs, and divides its allocation amongst them proportionally.

convergence problems. Performing throttling becomes challenging as the range $[L_s, U_s]$ becomes smaller.

2. *Convergence Time*: Even if convergence is obtained, oscillations can cause an increase in the time required for the server load to converge to the desired level.

3. *Router Locations*: Determining the deployment locations using the parameter k may be inappropriate. For some network topologies, the approach can cause severe collateral damage. Consider for example, the upper branch (i.e. the one having $R2$ as its root) in the network topology shown in figure 2. For $k \geq 3$, only router $R4$ will be selected because it is directly connected to a host (collateral damage occurs to some extend anyway, because no traffic classification is performed).

4. *Static Router Selection*: Currently, all the available defensive routers take part in the throttling mechanism. Consequently, there is an increased amount of communication between the server and routers. This has a direct impact on cost, and also increases the chances of communication signal exploitation.

We emphasise that router throttling has detection and response capabilities, but lacks the traffic classification capability. In general, the effectiveness of throttling increases with an increasing value of k. Throttling provides an easy, quick and relatively cheap solution against the DDoS threat.

3. LEARNING ROUTER THROTTLING

MARL is a suitable candidate to provide a distributed coordinated defence, which is essential to tackle the DDoS attacks. It offers an automated, effective and adaptive response against the distributed nature of these attacks. A reinforcement learning agent is deployed on each of the routers in $R(k)$. For example, in the network topology shown in figure 2, a reinforcement learning agent is deployed on the set $R(4)$ i.e. on routers $R4, R7, R8$. Each defensive router that performs throttling requires the monitoring of its load, that is, the arrival rate of the traffic destined to the server under protection. In other words, the state space of each reinforcement learning agent consists of only one state feature, which is its router load.

Each router applies throttling via probabilistic traffic dropping. For example action 0.4 means that the router will drop (approximately) 40% of its incoming traffic towards the server, thus allowing (approximately) only 60% of it to reach the server. The action is applied throughout the monitor window. Completing shutting off the incoming traffic destined to the server by a router is prohibited, that is, the action 1.0 (which corresponds to 100% drop probability) is not included in the action space of none of the routers. The reason being that the incoming traffic likely contains some legitimate traffic as well, and therefore dropping all the incoming traffic facilitates the task of the attacker, which is to deny legitimate users access to the server.

According to Mahajan et al [7] "there is no useful, policy-free equivalent of max-min fairness [2] when applied to aggregates; no one would recommend for best-effort traffic that we give ... an equal-share of the bandwidth in a time of high congestion. Instead the goal is to ... protect the other traffic on the link". The first objective of our system is therefore to allow as much legitimate traffic as possible to reach the server during a period of congestion. The idea behind this is that (the level of) "fairness" is an emerging property of the aforementioned objective. The second objective is to keep the server under working conditions, that is, to keep its server load below a certain threshold $C(= U_s)$. The two objectives are encoded in the reward function of each agent, and all agents receive the same reward.

A network is very dynamic, and most of all, unpredictable. Applying a learning algorithm in this environment is challenging. Recall the SARSA update rule shown in equation 1. The update rule performs bootstrapping, that is, it updates estimates for the current state-action pair ($Q(s, a)$), based on previous estimates of it ($Q(s, a)$) and the next state-action pair ($Q(s', a')$). Due to the unpredictable nature of a network, the dependency on the next state-action pair ($Q(s', a')$) is clearly undesirable. Consider the following example to explain why.

Let s_0 be the state perceived at time t_0, and s_1 be the state perceived at time t_1 (where $t_0 < t_1$). At time t_0 an agent decides to perform an action a based on state s_0. Let t_m be a time between the two time steps, i.e. $t_0 < t_m < t_1$. Suppose at time t_m more legitimate users make their appearance. Or suppose at time t_m an attack is initiated. It is obvious that the perceived state s_1 will have low correlation with the action a executed in s_0 (unless the action will have an impact on the attacker's behaviour, that is, the case where the attacker changes its behaviour according to the defensive action).

To deal with this unpredictable behaviour, we modified the SARSA algorithm in such a way to remove the dependency on the next state-action pair, thus resulting in the update rule shown in equation 2. This is equivalent to setting the discount factor to $\gamma = 0$. We have in fact found that training the agents using the original SARSA algorithm, can be hurtful.

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r - Q(s, a)] \tag{2}$$

4. SIMULATIONS

To evaluate our proposed approach we carried out simulations using the ns-2 network simulator [1]. ns-2 is an advanced network simulator which offers abstraction, scenario generation and extensibility [3]. Abstraction refers to the simulation at different levels of granularity. Scenario generation refers to the creation of complex traffic patterns, topologies and dynamic events. Extensibility allows users to add new functionality to the simulator. Despite these advantages, network simulators cannot always capture important details, which can be captured if wide-area testbeds or small-scale labs are used. Such alternatives though are very expensive, and are difficult to reconfigure. Therefore, the ns-2 is a good compromise. We do not claim to have created realistic network scenarios or to have stressed our proposed approach. The simulations are instead intended to illustrate the basic functionality of our approach.

During the training of the system, we keep track of, and distinguish between legitimate and bad traffic (requirement for the reward function). However, this is not the case during the evaluation of our system. The rationale behind this

[2] Another algorithm for equal-share fairness.

Algorithm 1 Reward function

```
if (server_load_{t+1} > C) then
    // Punishment
    reward = −5
else
    // Reward in [0, 1]
    reward = legitimate_reached_server/total_legitimate
end if
```

is that the defensive system can be trained in simulation, or in any other controlled physical environment e.g. small-scale lab, where legitimate and malicious packets can be identified, and then deployed in a realistic network, where such identification is impossible. Finally, as a convention, bandwidth and traffic rates are measured in $Mbit/s$.

4.1 Training

The network topology used for both training and evaluation purposes, is shown in figure 2. All the links have a bandwidth of 100, a delay (link propagation) of $10ms$ and they all use the "Drop Tail" mechanism.

Our purpose is to create a highly dynamic environment, in which we demonstrate the applicability of MARL. Before the start of each new episode, an attacker is selected with a probability of p, and a good user with a probability of $q = 1 - p$. We have chosen p and q to be 0.4 and 0.6 respectively, as in [13]. The network shown in figure 2 is therefore an instance of the model, consisting of 4 good sources ($H1, H4, H6, H7$) and 3 attackers ($H2, H3, H5$). Each normal flow is composed of UDP traffic at constant rates in the range $[0, 2]$. DoS flows are also composed of UDP traffic at constant rates in the range $[10, 12]$. The packet size for both normal and DoS flows is 512 *bytes*.

The MARL approach uses a linear decreasing ϵ-greedy exploration strategy with an initial $\epsilon = 0.2$ and the learning rate is set to $a = 0.1$. The reward function of each agent is shown in algorithm 1; all the routers receive the same reward. The monitor window is set to $2s$. We have discretised the state space of each router using steps of 5, and set the number of states to be 8. For example, states $1, 2$ and 8 correspond to a router load being in the range of $[0, 5], (5, 10]$ and $(35, \infty)$ respectively. The action space is also discretised, thus each agent has five available actions to choose from, these are, $0, 0.6, 0.7, 0.8, 0.9$ which correspond to $0\%, 60\%, 70\%, 80\%, 90\%$ traffic dropping. An action is applied throughout the monitor window. To facilitate learning, action 0.0 is hardwired in the system, that is, a router drops no traffic if it experiences a load which is lower than the threshold C. The state-action value estimates are stored in a tabular form.

The system is trained for 15000 episodes, each of a duration of 90s. Traffic starts at time $t = 0$ and finishes at $t = 90$. Approximately 50% of the episodes are trained only with legitimate traffic, while the rest are trained with both legitimate and DDoS traffic. Note that because of the dynamic nature of the training model, the episodes appear with a different sequence in every 15000-episode training. Figure 3 therefore presents the training results after a single 15000-episode training. A similar graph is obtained after three training repetitions.

Given such a dynamic environment, the system is able to converge to the optimal policy. The single state feature per router (i.e. its load) is inadequate to perform traffic classification and therefore collateral damage is inevitable in some episodes. This is the reason the reward received per episode may vary between a minimum and maximum value, as shown in figure 3.

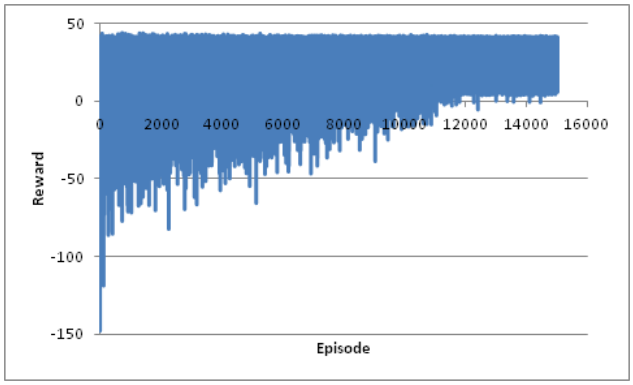


Figure 3: Training results

4.2 Evaluation

The baseline and fair approaches use a lower threshold $L_s = 8$ and an upper threshold $U_s = 10$. Other control parameters are set based on values and ranges recommended by [13], however no parameter optimisation is performed. The MARL approach use a threshold of $C = 10$. Furthermore, each agent uses its policy learnt during the training period. All approaches use the same monitor window as previously. Defencive nodes for all approaches are deployed on $R(4)$, i.e. routers $R4, R7, R8$.

Evaluation is conducted on the network topology shown in figure 2, and runs for $180s$. There are four legitimate sources, and three attackers. Legitimate users and attackers send at a constant rate of 1 and 10 respectively. Legitimate traffic flows throughout the interval, while an attack is initiated at time $t = 45s$ and stopped at time $t = 135s$. We evaluated our approach by comparing it with the baseline and fair approaches in two different network scenarios. Each scenario is repeated five times by each of the three approaches.

In the first scenario, the good and bad sources are $H1, H4, H6, H7$ and $H2, H3, H5$ respectively (as in figure 2). Because of the way the baseline and fair approaches work, oscillations can sometimes become a bottleneck. Oscillations cause stability problems, and a possibility of the server load not to converge within the lower and upper boundaries.

Figure 4 shows how the server load is altered throughout the scenario. Both the baseline and fair approaches suffer from stability problems; however, we note that these were not observed at all times. The MARL never experienced any stability problems. Moreover, as soon as the attack is initiated, the MARL approach responds rapidly, thus not allowing the attack to reach its full peak, as opposed to both the baseline and fair approaches.

Figure 5 shows the average legitimate traffic that reached the server during the scenario. The baseline approach allows only 60% of the total legitimate traffic to reach the server. Clearly, the baseline approach performs worse than

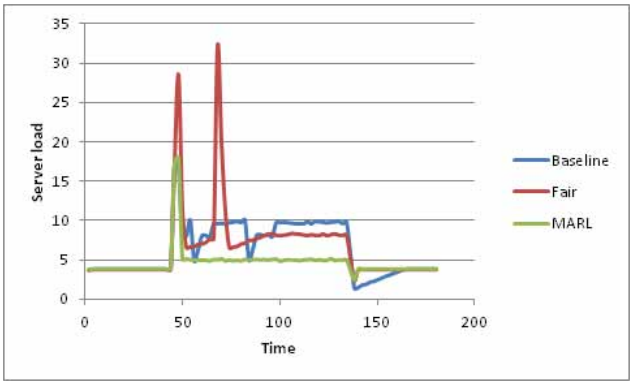


Figure 4: Scenario 1 - Server load

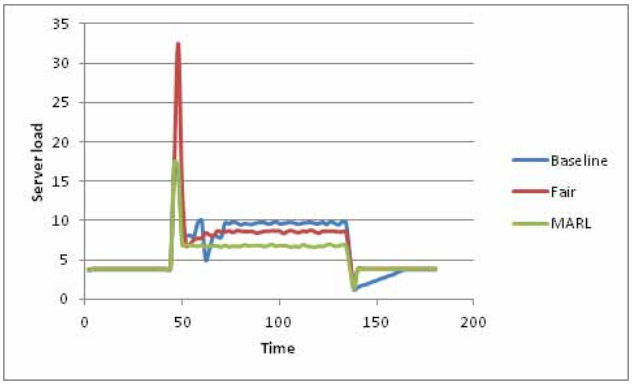


Figure 6: Scenario 2 - Server load

the others because it penalises traffic irrespective of whether routers are well behaving or not. Recall that in this scenario none of the hosts $H6$ and $H7$ misbehave, but still, router $R8$ penalises traffic from them. The fair and MARL approaches have a very similar performance, allowing 79% and 77% of legitimate traffic respectively. Although the amount of legitimate traffic is the primary evaluation criterion, for completeness we also present the average amount of attack traffic that managed to reach the server. Figure 5 also shows these results, indicating that our approach allows less bad traffic to reach the server compared to the other approaches.

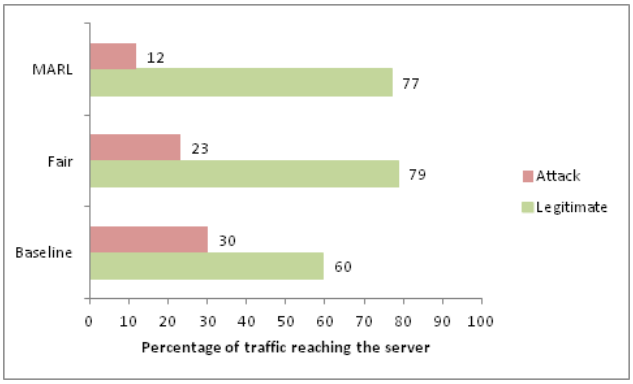


Figure 5: Scenario 1 - Percentage of traffic reaching the server for each of the three approaches

In the second scenario, the good and bad sources are $H1, H2, H4, H7$ and $H3, H5, H6$ respectively. The attackers are chosen in such a way that all three defensive nodes experience some bad traffic. Figure 6 shows how the server load is altered throughout the scenario. We didn't observe any stability problems in the fair approach for this scenario. The MARL approach didn't experience any stability problems as well. Again, as soon as the attack is initiated, the MARL approach responds rapidly, thus not allowing the attack to reach its full peak, as opposed to the other two approaches.

The average legitimate traffic allowed to reach the server is $60\%, 62\%$ and 63% for the baseline, fair and MARL approaches respectively, as presented in figure 7. As expected, the baseline approach performs almost as good as the other two approaches. Again, for completeness, figure 7 also presents the average attack traffic that reached the server, which shows that MARL again allows less attack traffic compared to the other approaches.

In summary, the experiments show promising results to support our claim that our approach is more stable than the baseline and fair approaches. This is achieved by the ability of our system to learn the router throttles during its training. Similarly to the fair approach, evidence suggests that our approach outperforms the baseline approach, as it does not penalise well-behaved users.

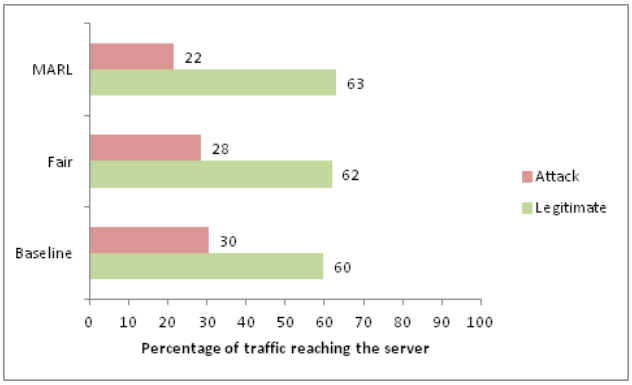


Figure 7: Scenario 2 - Percentage of traffic reaching the server for each of the three approaches

5. DISCUSSION

It is of particular importance to emphasise the challenges encountered. Firstly, throttling does not completely solve the DDoS problem, in the sense that no traffic classification is performed, and therefore the system cannot distinguish between DDoS and normal traffic. Responding to a DDoS attack, by only taking into account measurements of the server and routers' load, makes throttling itself a challenging task.

Furthermore, the highly dynamic nature of the environment makes learning difficult. Moreover, the unpredictable network behaviour requires a modification of the original

SARSA update rule (shown in equation 1), to the one with removed dependency on the next state-action pair (shown in equation 2). An extra challenge imposed by the environment is that a throttling action corresponds to the probabilistic dropping of traffic. By executing for example action 0.5, it is expected that 50% of the traffic will be dropped; this however is just the expected outcome, and not necessarily the actual one. This makes the learning procedure even more difficult. Also, an action by the router is applied throughout the monitor window.

Despite all these challenges, it is apparent that the MARL algorithm is capable of learning optimal or nearly optimal policies. An advantage of our approach is that it only needs to be provided with one threshold C regarding the server load, as opposed to the baseline and fair approaches which need to be provided with lower L_s and upper U_s thresholds (where $C = U_s$). Therefore stability and convergence problems due to oscillations are avoided.

Existing work on machine learning detection involves the monitoring of network traffic, in order to learn whether to raise an alarm or not. Our system performs throttling only when the server load exceeds its threshold C. Furthermore, this work has a narrow but well-defined scope by focusing on flooding DDoS attacks, as opposed to the majority of work, which employs a machine learning technique (or techniques) to detect any kind of intrusion.

Our approach currently makes an additional assumption that if the server becomes overloaded (its load exceeds C), this means that the corresponding router (or routers) experiences a traffic load of more than C towards the server. This assumption is not irrational, since a defensive router can be deployed at a key location, for example at an ingress router [3]. This however may lead to deception of our system. Consider for example the network shown in figure 2. The defensive mechanism will not be initiated in case routers in $R(k)$ each experience a load of 9. We will address this issue in our future work.

6. CONCLUSION AND FUTURE WORK

Concluding, we have presented a novel learning throttling approach against flooding DDoS attacks, which uses MARL. We view the flooding DDoS problem as a resource management or congestion problem. MARL learns router throttles in such a way to ensure that the server under protection is not overloaded, and as much legitimate traffic as possible reaches the server. Our approach creates an automated response, and mitigates the effectiveness of flooding DDoS attacks. It is also shown that it overcomes stability and convergence problems, that other approaches found in the literature suffer from.

As a future work, we plan to investigate the incorporation of more defensive routers, for example routers $R4$, $R6$, $R7$, $R8$, $R10$ in figure 2. We anticipate that an agent will learn when to activate a throttle, and to what extend. This would result to a flexible, dynamic throttling approach, which will offer more advantages. Collateral damage will be reduced since throttling will be performed closer to the sources. Cost effectiveness is another advantage, since only activated routers will take part in the defence mechanism. As a consequence this will reduce communication costs as well. Additionally, less communication means fewer chances of communication signal exploitation.

Despite these advantages, some challenges arise. Such a defence will definitely need to be collaborative, due to the distributed location of the routers. Specifically, when a downstream router does not experience high traffic load, it must be able to distinguish whether this is because of attack absence, or because of upstream router throttling. We plan to investigate information sharing mechanisms to address this issue. We also intend to study the scalability of our approach.

[3] An ingress router is the place where incoming traffic arrives.

7. REFERENCES

[1] The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[3] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. Mccanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

[4] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, Mar. 2008.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *In Proceedings of National Conference on Artificial Intelligence (AAAI-98*, pages 746–752, 1998.

[6] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Taipei, Taiwan*. ACM Press, 2011.

[7] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *Computer Communication Review*, 32(3):62–73, 2002.

[8] J. Mirkovic, M. Robinson, P. Reiher, and G. Oikonomou. Distributed defense against ddos attacks. Technical report, University of Delaware, 2006.

[9] K. J. O'Briend. Putting the breaks on web-surfing speeds. *New York Times*, Nov. 13 2011.

[10] S. M. Specht. Distributed denial of service: taxonomies of attacks, tools and countermeasures. In *Proceedings of the International Workshop on Security in Parallel and Distributed Systems, 2004*, pages 543–550, 2004.

[11] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010.

[12] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press Cambridge, MA, USA, 1998.

[13] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. In *IEEE/ACM Transactions on Networking*, pages 29–42, 2005.

Safe Opponent Exploitation*

Sam Ganzfried and Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
{sganzfri, sandholm}@cs.cmu.edu

ABSTRACT

We consider the problem of playing a finitely-repeated two-player zero-sum game safely—that is, guaranteeing at least the value of the game per period in expectation regardless of the strategy used by the opponent. Playing a stage-game equilibrium strategy at each time step clearly guarantees safety, and prior work has conjectured that it is impossible to simultaneously deviate from a stage-game equilibrium (in hope of exploiting a suboptimal opponent) and to guarantee safety. We show that such profitable deviations are indeed possible—specifically, in games where certain types of 'gift' strategies exist, which we define formally. We show that the set of strategies constituting such gifts can be strictly larger than the set of iteratively weakly-dominated strategies; this disproves another recent conjecture which states that all non-iteratively-weakly-dominated strategies are best responses to each equilibrium strategy of the other player. We present a full characterization of safe strategies, and develop efficient algorithms for exploiting suboptimal opponents while guaranteeing safety. We also provide the analogous results for sequential perfect- and imperfect-information games, and present safe exploitation algorithms and full characterizations of safe strategies for those settings as well. We present experimental results in Kuhn poker, a canonical test problem for game-theoretic algorithms. Among other things, the experiments show that aggressive safe exploitation strategies significantly outperform adjusting the exploitation within equilibrium strategies only.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; J.4 [**Social and Behavioral Sciences**]: Economics

General Terms

Algorithms, economics, theory

Keywords

Game theory, opponent exploitation, multiagent learning

*This material is based upon work supported by the National Science Foundation under grants IIS-0964579, IIS-0905390, and CCF-1101668. We also acknowledge Intel Corporation and IBM for their machine gifts. A longer version of this paper will appear in the proceedings of the ACM Conference on Electronic Commerce [4]. Proofs that have been omitted in this version can be found there.

1. INTRODUCTION

In repeated interactions against an opponent, an agent must determine how to balance between *exploitation* (maximally taking advantage of weak opponents) and *exploitability* (making sure that he himself does not perform too poorly against strong opponents). In two-player zero-sum games, an agent can simply play a minimax strategy, which guarantees at least the value of the game in expectation against any opponent. However, doing so could potentially forego significant profits against suboptimal opponents. Thus, an equilibrium strategy has low (zero) exploitability, but achieves low exploitation. On the other end of the spectrum, agents could attempt to learn the opponent's strategy and maximally exploit it; however, doing so runs the risk of being exploited in turn by a deceptive opponent. This is known as the "get taught and exploited problem" [9]. Such deception is common in games such as poker; for example, a player may play very aggressively initially, then suddenly switch to a more conservative strategy to capitalize on the fact that the opponent tries to take advantage of his aggressive 'image', which he now leaves behind. Thus, pure opponent modeling potentially leads to a high level of exploitation, but at the expense of exploitability. Respectively, the game solving community has, by and large, taken two radically different approaches: finding game-theoretic solution and opponent modeling/exploitation.

In this paper, we are interested in answering a fundamental question that helps shed some light on this tradeoff:

> **Is it possible to play a strategy that is not an equilibrium in the stage game while simultaneously guaranteeing at least the value of the game in expectation in the worst case?**

If the answer is no, then fully safe exploitation is not possible, and we must be willing to accept some increase in worst-case exploitability if we wish to deviate from equilibrium in order to exploit suboptimal opponents. However, if the answer is yes, then safe opponent exploitation would indeed be possible.

Recently it was proposed that safe opponent exploitation is not possible [3]. The intuition for that argument was that the opponent could have been playing an equilibrium all along, and when we deviate from equilibrium to attempt to exploit him, then we run the risk of being exploitable ourselves. However, that argument is incorrect. It does not take into account the fact that our opponent may give us a *gift* by playing an identifiably suboptimal strategy, such as

one that is strictly dominated.[1] If such gift strategies are present in a game, then it turns out that safe exploitation can be achieved; specifically, we can deviate from equilibrium to exploit the opponent provided that our worst-case exploitability remains below the total amount of profit won through gifts (in expectation).

Is it possible to obtain such gifts that do not correspond to strictly dominated strategies? What about other forms of dominance, such as weak, iterated, and dominance by mixed strategies? Recently it was conjectured that all non-iteratively-weakly-dominated strategies are best responses to each equilibrium strategy of the other player [10]. This would suggest that such undominated strategies cannot be gifts, and that gift strategies must therefore be dominated according to some form of dominance. We disprove this conjecture and present a game in which a non-iteratively-weakly-dominated strategy is not a best response to an equilibrium strategy of the other player. Safe exploitation is possible in the game by taking advantage of that particular strategy. We define a formal notion of gifts, which is more general than iteratively-weakly-dominated strategies, and show that safe opponent exploitation is possible specifically in games in which such gifts exist.

Next, we provide a full characterization of the set of safe exploitation strategies, and we present several efficient algorithms for converting any opponent modeling algorithm (that is arbitrarily exploitable) into a fully safe opponent exploitation procedure. One of our algorithms is similar to a procedure that guarantees safety in the limit as the number of iterations goes to infinity [8]; however, the algorithms in that paper can be arbitrarily exploitable in the finitely-repeated game setting, which is what we are interested in. The main idea of the algorithm is to play an ϵ-safe best response (a best response subject to the constraint of having exploitability at most ϵ) at each time step rather than a full best response, where ϵ is determined by the total amount of gifts obtained thus far from the opponent. Safe best responses have also been studied in the context of Texas Hold'em poker [6], though that work did not use them for real-time opponent exploitation. We also present several other safe algorithms which alternate between playing an equilibrium and a best response depending on how much has been won so far in expectation.

It turns out that safe opponent exploitation is also possible in sequential games, though we must redefine what strategies constitute gifts and must make pessimistic assumptions about the opponent's play in game states off the path of play. We present efficient algorithms for safe exploitation in games of both perfect and imperfect information, and fully characterize the space of safe strategies in these game models.

We compare our algorithms experimentally on Kuhn poker [7], a simplified form of poker which is a canonical problem for testing game-solving algorithms and has been used as a test problem for opponent-exploitation algorithms [5]. We observe that our algorithms obtain a significant improvement over the best equilibrium strategy, while also guaranteeing safety in the worst case. Thus, in addition to providing theoretical advantages over both minimax and fully-exploitative strategies, safe opponent exploitation can be effective in practice.

[1]We thank Vince Conitzer for pointing this out to us.

2. GAME THEORY BACKGROUND

In this section, we briefly review relevant definitions and prior results from game theory and game solving.

2.1 Strategic-form games

The most basic game representation, and the standard representation for simultaneous-move games, is the *strategic form*. A *strategic-form game* (aka matrix game) consists of a finite set of players N, a space of *pure strategies* S_i for each player, and a utility function $u_i : \times S_i \rightarrow \mathbb{R}$ for each player. Here $\times S_i$ denotes the space of *strategy profiles* — vectors of pure strategies, one for each player.

The set of *mixed strategies* of player i is the space of probability distributions over his pure strategy space S_i. We will denote this space by Σ_i. Define the *support* of a mixed strategy to be the set of pure strategies played with nonzero probability. If the sum of the payoffs of all players equals zero at every strategy profile, then the game is called *zero sum*. In this paper, we will be primarily concerned with two-player zero-sum games. If the players are following strategy profile σ, we let σ_{-i} denote the strategy taken by player i's opponent, and we let Σ_{-i} denote the opponent's entire mixed strategy space.

2.2 Extensive-form games

An *extensive-form* game is a general model of multiagent decision making with potentially sequential and simultaneous actions and imperfect information. As with perfect-information games, extensive-form games consist primarily of a game tree; each non-terminal node has an associated player (possibly *chance*) that makes the decision at that node, and each terminal node has associated utilities for the players. Additionally, game states are partitioned into *information sets*, where the player whose turn it is to move cannot distinguish among the states in the same information set. Therefore, in any given information set, a player must choose actions with the same distribution at each state contained in the information set. If no player forgets information that he previously knew, we say that the game has *perfect recall*. A (behavioral) *strategy* for player i, $\sigma_i \in \Sigma_i$, is a function that assigns a probability distribution over all actions at each information set belonging to i.

2.3 Nash equilibria

Player i's *best response* to σ_{-i} is any strategy in

$$\arg\max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i}).$$

A *Nash equilibrium* is a strategy profile σ such that σ_i is a best response to σ_{-i} for all i. An *ϵ-equilibrium* is a strategy profile in which each player achieves a payoff of within ϵ of his best response.

In two player zero-sum games, we have the following result which is known as the *minimax theorem*:

$$v^* = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} u_1(\sigma_1, \sigma_2) = \min_{\sigma_2 \in \Sigma_2} \max_{\sigma_1 \in \Sigma_1} u_1(\sigma_1, \sigma_2).$$

We refer to v^* as the *value* of the game to player 1. Sometimes we will write v_i as the value of the game to player i. It is important to note that *any* equilibrium strategy for a player will guarantee an expected payoff of at least the value of the game to that player.

Define the exploitability of σ_i to be the difference between the value of the game and the performance of σ_i against its

nemesis, formally:

$$\text{expl}(\sigma_i) = v_i - \min_{\sigma_{-i}} u_i(\sigma_i, \sigma_{-i}).$$

For any $\epsilon \geq 0$, define SAFE(ϵ) to be the set of strategies with exploitability at most ϵ. Define the ϵ-safe best response of player i to σ_{-i} to be

$$\text{argmax}_{\sigma_i \in \text{SAFE}(\epsilon)} u_i(\sigma_i, \sigma_{-i}).$$

2.4 Repeated games

In repeated games, the *stage game* is repeated for a finite number T of iterations. At each iteration, players can condition their strategies on everything that has been observed so far. In strategic-form games, this generally includes the full mixed strategy of the agent in all previous iterations, as well as all actions of the opponent (though not his full strategy). In extensive-form games, generally only the actions of the opponent along the path of play are observed; in games with imperfect information, the opponent's private information may also be observed in some situations.

3. SAFETY

One desirable property of strategy for a repeated game is that it is *safe* — that it guarantees at least v_i per period in expectation. Clearly playing a minimax strategy at each iteration is safe, since it guarantees at least v_i in each iteration. However, a minimax strategy may fail to maximally exploit a suboptimal opponent. On the other hand, deviating from stage-game equilibrium in an attempt to exploit a suboptimal opponent could lose the guarantee of safety and may result in an expected payoff below the value of the game against a deceptive opponent (or if the opponent model is incorrect).

3.1 A game in which safe exploitation is not possible

Consider the classic game of Rock-Paper-Scissors (RPS), whose payoff matrix is depicted in Figure 1. The unique equilibrium σ^* is for each player to randomize equally among all three pure strategies.

	R	P	S
R	0	-1	1
P	1	0	-1
S	-1	1	0

Figure 1: Payoff matrix of Rock-Paper-Scissors.

Now suppose that our opponent has played Rock in each of the first 10 iterations (while we have played according to σ^*). We may be tempted to try to exploit him by playing the pure strategy Paper at the 11th iteration. However, this would not be safe; it is possible that he has in fact been playing his equilibrium strategy all along, and that he just played Rock each time by chance (this will happen with probability $\frac{1}{3^{10}}$). It is also possible that he will play Scissors in the next round (perhaps to exploit the fact that he thinks we are more likely to play Paper having observed his actions). Against such a strategy, we would actually have a negative expected total profit — 0 in the first 10 rounds and -1 in the 11th. Thus, our strategy would not be safe. By similar reasoning, it is easy to see that any deviation from σ^* will not be safe, and that safe exploitation is not possible in RPS.

3.2 A game in which safe exploitation is possible

Now consider a variant of RPS in which player 2 has an additional pure strategy T. If he plays T, then we get a payoff of 4 if we play R, and 3 if we play P or S. The payoff matrix of this new game RPST is given in Figure 2. Clearly the unique equilibrium is still for both players to randomize equally between R, P, and S. Now suppose we play our equilibrium strategy in the first game iteration, and the opponent plays T; no matter what action we played, we receive a payoff of at least 3. Now suppose we play the pure strategy R in the second round in an attempt to exploit him (since R is our best response to T). In the worst case, our opponent will exploit us in the second round by playing P, and we will obtain payoff -1. But combined over both time steps, our payoff will be positive no matter what the opponent does at the second iteration. Thus, our strategy constituted a safe deviation from equilibrium. This was possible because of the existence of a 'gift' strategy for the opponent; no such gift strategy is present in standard RPS.

	R	P	S	T
R	0	-1	1	4
P	1	0	-1	3
S	-1	1	0	3

Figure 2: Payoff matrix of RPST.

4. CHARACTERIZING GIFTS

What exactly constitutes a gift? Does it have to be a strictly dominated pure strategy, like T in the preceding example? What about weakly-dominated strategies? What about iterated-dominance, or dominated mixed strategies?

Recent work has conjectured the following:

CONJECTURE 1. *[10] An equilibrium strategy makes an opponent indifferent to all non-[weakly]-iteratively dominated strategies. That is, to tie an equilibrium strategy in expectation, all one must do is play a non-[weakly]-iteratively dominated strategy.*

This conjecture would seem to imply that gifts correspond to strategies that put weight on pure strategies that are weakly-iteratively dominated. However, consider the game shown in Figure 3.

	L	M	R
U	3	2	10
D	2	3	0

Figure 3: A game with a gift strategy that is not weakly iteratively dominated.

It can easily be shown that this game has a unique equilibrium, in which P1 plays U and D with probability $\frac{1}{2}$, and P2 plays L and M with probability $\frac{1}{2}$. The value of the game to player 1 is 2.5. If player 1 plays his equilibrium strategy and player 2 plays R, player 1 gets expected payoff of 5, which exceeds his equilibrium payoff; thus R constitutes a gift, and player 1 can safely deviate from equilibrium to try to exploit him. But note that R is not dominated under any form of dominance. This disproves the conjecture, and causes us to rethink our notion of gifts.

PROPOSITION 1. *It is possible for a strategy that survives iterated weak dominance to obtain expected payoff worse than the value of the game against an equilibrium strategy.*

We define a gift strategy as follows:

DEFINITION 1. *A strategy σ_{-i} is a* gift strategy *if there exists an equilibrium strategy σ_i^* for the other player such that σ_{-i} is not a best response to σ_i^*.*

PROPOSITION 2. *Assuming we are not in a trivial game in which all of player i's strategies are minimax strategies, then non-stage-game equilibrium safe strategies exist if and only if there exists at least one gift strategy for the opponent.*

5. SAFETY ANALYSIS OF SOME NATURAL EXPLOITATION ALGORITHMS

Now that we know it is possible to safely deviate from equilibrium in certain games, can we construct efficient procedures for implementing such safe exploitative strategies? In this section we analyze the safety of several natural exploitation algorithms.

5.1 Risk What You've Won (RWYW)

The 'Risk what you've won' algorithm (RWYW) is quite simple; essentially, at each iteration it risks only the amount of profit won so far. More specifically, at each iteration t, RWYW plays an ϵ-safe best response to a model of the opponent's strategy (according to some opponent modeling algorithm M), where ϵ is our current cumulative payoff minus $(t-1)v^*$. Pseudocode is given in Algorithm 1.

Algorithm 1 Risk What You've Won (RWYW)

$v^* \leftarrow$ value of the game to player i
$k^1 \leftarrow 0$
for $t = 1$ to T **do**
  $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}_{(k^t)}} M(\pi)$
  Play action a_i^t according to π^t
  Update M with opponent's actions, a_{-i}^t
  $k^{t+1} \leftarrow k^t + u_i(a_i^t, a_{-i}^t) - v^*$
end for

PROPOSITION 3. *RWYW is not safe.*

PROOF. Consider RPS, and assume our opponent modeling algorithm M says that the opponent will play according to his distribution of actions observed so far. Since initially $k^1 = 0$, we must play our equilibrium strategy σ^* at the first iteration, since it is the only strategy with exploitability of 0. Without loss of generality, assume the opponent plays R in the first iteration. Our expected payoff in the first iteration is 0, since σ^* has expected payoff of 0 against R (or any strategy). Suppose we had played R ourselves in the first iteration. Then we would have obtained an actual payoff of 0, and would set $k^2 = 0$. Thus we will be forced to play σ^* at the second iteration as well. If we had played P in the first round, we would have obtained a payoff of 1, and set $k^2 = 1$. We would then set π^2 to be the pure strategy P, since our opponent model dictates the opponent will play R again, and P is the unique k^2-safe best response to R. Finally, if we had played S in the first round, we would have obtained an actual payoff of -1, and would set $k^2 = -1$; this would require us to set π^2 equal to σ^*.

Now, suppose the opponent had actually played according to his equilibrium strategy in iteration 1, plays the pure strategy S in the second round, then plays the equilibrium in all subsequent rounds. As discussed above, our expected payoff at the first iteration is zero. Against this strategy, we will actually obtain an expected payoff of -1 in the second iteration if the opponent happened to play R in the first round, while we will obtain an expected of 0 in the second round otherwise. So our expected payoff in the second round will be $\frac{1}{3} \cdot (-1) + \frac{2}{3} \cdot 0 = -\frac{1}{3}$. In all subsequent rounds our expected payoff will be zero. Thus our overall expected payoff will be $-\frac{1}{3}$, which is less than the value of the game; so RWYW is not safe. □

RWYW is not safe because it does not adequately differentiate between whether profits were due to skill (i.e., from gifts) or to luck.

5.2 Risk What You've Won in Expectation (RWYWE)

A better approach than RWYW would be to risk the amount won so far *in expectation*. Ideally we would like to do the expectation over both our randomization and our opponent's, but this is not possible in general since we only observe the opponent's action, not his full strategy. However, it would be possible to do the expectation only over our randomization. It turns out that we can indeed achieve safety using this procedure, which we call RWYWE. Pseudocode is given in Algorithm 2. Here $u_i(\pi_i^t, a_{-i}^t)$ denotes our expected payoff of playing our mixed strategy π_i^t against the opponent's observed action a_{-i}^t.

Algorithm 2 Risk What You've Won in Expectation (RWYWE)

$v^* \leftarrow$ value of the game to player i
$k^1 \leftarrow 0$
for $t = 1$ to T **do**
  $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}_{(k^t)}} M(\pi)$
  Play action a_i^t according to π^t
  The opponent plays action a_{-i}^t according to unobserved distribution π_{-i}^t.
  Update M with opponent's actions, a_{-i}^t
  $k^{t+1} \leftarrow k^t + u_i(\pi_i^t, a_{-i}^t) - v^*$
end for

LEMMA 1. *Let π be updated according to RWYWE, and suppose the opponent plays according to π_{-i}. Then for all $n \geq 0$,*

$$E[k^{n+1}] = \sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*.$$

LEMMA 2. *Let π be updated according to RWYWE. Then for all $t \geq 1$, $k^t \geq 0$.*

PROPOSITION 4. *RWYWE is safe.*

PROOF. By Lemma 1,

$$\sum_{t=1}^{T} u_i(\pi_i^t, \pi_{-i}^t) = E[k^{T+1}] + Tv^*.$$

By Lemma 2, $k^{T+1} \geq 0$, and therefore $E[k^{T+1}] \geq 0$. So

$$\sum_{t=1}^{T} u_i(\pi_i^t, \pi_{-i}^t) \geq Tv^*,$$

and RWYWE is safe. □

RWYWE is similar to the Safe Policy Selection Algorithm (SPS), proposed in [8]. The main difference is that SPS uses an additional decay function f, using the update step

$$k^{t+1} \leftarrow k^t + f(t+1) + u_i(\pi^t, a_{-i}^t) - v^*.$$

They are able to show that SPS is safe in the limit as $T \to \infty$;[2] however SPS is arbitrarily exploitable in finitely repeated games. Furthermore, even in infinitely repeated games, the SPS can lose a significant amount; it is merely the average loss that approaches zero. We can think of RWYWE as SPS but using $f(t) = 0$ for all t.

5.3 Best equilibrium strategy

Given an opponent modeling algorithm M, we could play the best Nash equilibrium according to M at each time step:

$$\pi^t = \text{argmax}_{\pi \in \text{SAFE}_{(0)}} M(\pi).$$

This would clearly be safe, but can only exploit the opponent as much as the best equilibrium can, and potentially leaves a lot of exploitation on the table.

5.4 Regret minimization between an equilibrium and an opponent modeling algorithm

We could use a no-regret algorithm (e.g., [1]) to select between an equilibrium and opponent modeling algorithm M at each iteration. As pointed out in [8], this would be safe in the limit as $T \to \infty$. However, this would not be safe in finitely-repeated games. Note that even in the infinitely-repeated case, no-regret algorithms only guarantee that average regret goes to 0 in the limit; in fact, total regret can still grow arbitrarily large.

5.5 Regret minimization in the space of equilibria

Regret minimization in the space of equilibria is safe, but again would potentially miss out on a lot of exploitation against suboptimal opponents. This procedure was previously used to exploit opponents in Kuhn poker [5].

5.6 Best equilibrium followed by full exploitation (BEFFE)

The BEFFE algorithm works as follows. We start off playing the best equilibrium strategy according to some opponent model M. Then we switch to playing a full best response for all future iterations if we know that doing so will keep our strategy safe in the full game (in other words, if we know we have accrued enough gifts to support full exploitation in the remaining iterations). Pseudocode is given in Algorithm 3.

This algorithm is similar to the DBBR algorithm [3], which plays an equilibrium for some fixed number of iterations, then switches to full exploitation. However, BEFFE automatically detects when this switch should occur, which has several advantages. First, it is one fewer parameter required by the algorithm. More importantly, it enables the algorithm to guarantee safety.

PROPOSITION 5. *BEFFE is safe.*

One possible advantage of BEFFE over RWYWE is that it potentially saves up exploitability until the end of the game, when it has the most accurate information on the opponent's strategy (while BEFFE does exploitation from the start when the opponent model has noisier data). On the other hand, BEFFE possibly misses out on additional rounds of exploitation by waiting until the end, since it may accumulate additional gifts in the exploitation phase that it did not take into account. Furthermore, by waiting longer before turning on exploitation, one's experience of the opponent can be from the wrong part of the space, that is, the space that is reached when playing equilibrium but not when exploiting. Consequently, the exploitation might not be as effective because it may be based on less data about the opponent in the pertinent part of the space. This issue has been observed in opponent exploitation in Heads-Up Texas Hold'em poker [3].

[2]We recently discovered a mistake in their proof of safety in the limit; however, the result is still correct.

Algorithm 3 Best Equilibrium Followed by Full Exploitation (BEFFE)

```
v* ← value of the game to player i
k^1 ← 0
for t = 1 to T do
  π^t_BR ← argmax_π M(π)
  ε ← v* − min_{π_−i} u_i(π^t_BR, π_−i)
  if k^t >= (T − t + 1)(v* − ε) then
    π^t ← π^t_BR
  else
    π^t ← argmax_{π∈SAFE_(0)} M(π)
  end if
  Play action a^t_i according to π^t
  The opponent plays action a^t_−i according to unobserved distribution π^t_−i.
  Update M with opponent's actions, a^t_−i
  k^{t+1} ← k^t + u_i(π^t_i, a^t_−i) − v*
end for
```

5.7 Best equilibrium and full exploitation when possible (BEFEWP)

BEFEWP is similar to BEFFE, but rather than waiting until the end of the game, we play a full best response at each iteration where its exploitability is below k^t; otherwise we play the best equilibrium. Pseudocode is given in Algorithm 4.

Algorithm 4 Best Equilibrium and Full Exploitation When Possible (BEFEWP)

```
v* ← value of the game to player i
k^1 ← 0
for t = 1 to T do
  π^t_BR ← argmax_π M(π)
  ε ← v* − min_{π_−i} u_i(π^t_BR, π_−i)
  if ε <= k^t then
    π^t ← π^t_BR
  else
    π^t ← argmax_{π∈SAFE_(0)} M(π)
  end if
  Play action a^t_i according to π^t
  The opponent plays action a^t_−i according to unobserved distribution π^t_−i.
  Update M with opponent's actions, a^t_−i
  k^{t+1} ← k^t + u_i(π^t_i, a^t_−i) − v*
end for
```

Like RWYWE, BEFEWP will continue to exploit a suboptimal opponent throughout the match provided the opponent keeps giving us gifts. It also guarantees safety, since we are still playing a strategy with exploitability at most k^t at each iteration. However, playing a full best response rather than a safe best response early in the match may not be the greatest idea, since our data on the opponent is still quite noisy.

PROPOSITION 6. *BEFEWP is safe.*

6. A FULL CHARACTERIZATION OF SAFE STRATEGIES IN STRATEGIC-FORM GAMES

In the previous section we saw a variety of opponent exploitation algorithms, some which are safe and some which are unsafe. In this section, we fully characterize the space of safe algorithms. Informally, it turns out that an algorithm will be safe if at each time step it selects a strategy with exploitability at most k^t, where k is updated according to the RWYWE procedure. Note that this does not mean that RWYWE is the only safe algorithm, or that safe algorithms must explicitly use the given update rule for k^t; it just means that the exploitability at each time step must be bounded by the particular value k^t, assuming that k had hypothetically been updated according to the RWYWE rule.

DEFINITION 2. *An algorithm for selecting strategies is* expected-profit-safe *if it satisfies the rule*

$$\pi^t \in SAFE(k^t)$$

at each time step t from 1 to T, where initially $k^1 = 0$ and k is updated using the rule

$$k^{t+1} \leftarrow k^t + u_i(\pi^t, a^t_{-i}) - v^*$$

PROPOSITION 7. *A strategy π (for the full game, not the stage game) is safe if and only if it is expected-profit-safe.*

7. SAFE EXPLOITATION IN SEQUENTIAL GAMES

In sequential games, we cannot immediately apply RWYWE (or the other safe algorithms that deviate from equilibrium), since we do not know what the opponent would have done at game states off the path of play (and thus cannot evaluate the expected payoff of our mixed strategy).

7.1 Sequential games of perfect information

In sequential games of perfect information, it turns out that to guarantee safety we must assume pessimistically that the opponent is playing a nemesis off the path of play (while playing his observed action on the path of play). This pessimism potentially limits our amount of exploitation when the opponent is not playing a nemesis, but is needed to guarantee safety.

Algorithm 5 Sequential RWYWE

$v^* \leftarrow$ value of the game to player i
$k^1 \leftarrow 0$
for $t = 1$ to T **do**
  $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}_{(k^t)}} M(\pi)$
  Play action a^t_i according to π^t
  The opponent plays action a^t_{-i} according to unobserved distribution π^t_{-i}.
  Update M with opponent's actions, a^t_{-i}
  $\tau^t_{-i} \leftarrow$ strategy for the opponent that plays a^t_{-i} on the path of play, and plays a best response to π^t off the path of play.
  $k^{t+1} \leftarrow k^t + u_i(\pi^t_i, \tau^t_{-i}) - v^*$
end for

PROPOSITION 8. *Sequential RWYWE is safe.*

We now provide a full characterization of safe exploitation algorithms in sequential games—similarly to what we did for strategic-form games earlier in the paper.

DEFINITION 3. *An algorithm for selecting strategies in sequential games of perfect information is* expected-profit-safe *if it satisfies the rule*

$$\pi^t \in SAFE(k^t)$$

at each time step t from 1 to T, where initially $k^1 = 0$ and k is updated using the same rule as Sequential RWYWE.

PROPOSITION 9. *A strategy π in a sequential game of perfect information is safe if and only if it is expected-profit-safe.*

7.2 Sequential games of imperfect information

In sequential games of imperfect information, not only do we not see the opponent's action off of the path of play, but sometimes we do not even see his private information. We consider the two cases—when his private information is observed and unobserved—separately.

7.2.1 Opponent's private information is observed at the end of the game

When the opponent's private information is observed at the end of each game iteration, we can play a procedure similar to Sequential RWYWE. Here, we must pessimistically assume that the opponent would have played a nemesis at every information set off of the path of play (though we do not make any assumptions regarding his play along the path of play other than that he played action a^t_{-i} with observed private information θ^t_{-i}). Pseudocode for this procedure is given in Algorithm 6.

Algorithm 6 Safe exploitation algorithm for sequential games of imperfect information where opponent's private information is observed at the end of the game

$v^* \leftarrow$ value of the game to player i
$k^1 \leftarrow 0$
for $t = 1$ to T **do**
  $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}_{(k^t)}} M(\pi)$
  Play action a^t_i according to π^t
  The opponent plays action a^t_{-i} with observed private information ρ^t_{-i}, according to unobserved distribution π^t_{-i}.
  Update M with opponent's actions, a^t_{-i}, and his private information, θ^t_{-i}
  $\tau^t_{-i} \leftarrow$ strategy for the opponent that plays a best response to π^t subject to the constraint that it plays a^t_{-i} on the path of play with private information θ^t_{-i}.
  $k^{t+1} \leftarrow k^t + u_i(\pi^t_i, \tau^t_{-i}) - v^*$
end for

PROPOSITION 10. *Algorithm 6 is safe.*

DEFINITION 4. *An algorithm for selecting strategies in sequential games of imperfect information is* expected-profit-safe *if it satisfies the rule*

$$\pi^t \in SAFE(k^t)$$

at each time step t from 1 to T, where initially $k^1 = 0$ and k is updated using the same rule as Algorithm 6.

PROPOSITION 11. *A strategy π in a sequential game of imperfect information is safe if and only if it is expected-profit-safe.*

7.2.2 Opponent's private information is not observed

Unfortunately we must be extremely pessimistic if the opponent's private information is not observed, though it can still be possible to detect gifts in some cases. We can only be sure we have received a gift if the opponent's observed action would have been a gift for any possible private information he may have. Thus we can run an algorithm similar to Algorithm 6, where we redefine τ^t_{-i} to be the opponent's best response subject to the constraint that he plays a^t_{-i} with *some* private information.

The approaches from this subsection and the previous subsection can be combined if we observe some of the opponent's private information afterwards but not all. Again, we must be pessimistic an assume he plays a nemesis subject to the restriction that we plays the observed actions with the observed part of his private information.

7.3 Detecting gifts within a game iteration

In some situations, we could detect gift actions early in the game that would allow us to risk trying to exploit him even in the middle of a single game iteration. We can use a variant of the Sequential RWYWE update rule to detect gifts during a game iteration, where we redefine τ^t_{-i} to be the opponent's best response to π^t_i subject to the constraint that he has taken the observed actions along the path of play thus far. This would allow us to safely deviate from equilibrium to exploit him even during a game iteration.

8. EXPERIMENTS

We ran experiments using the sequential imperfect-information variants of several of the safe algorithms presented in Section 5. The domain we consider is Kuhn poker [7], a simplified form of poker which has been frequently used as a test problem for game-theoretic algorithms [5].

8.1 Kuhn poker

Kuhn poker is a two-person zero-sum poker game, consisting of a three-card deck and a single round of betting. The full rules can be found in the papers cited above. The value of the game to player 1 is $-\frac{1}{18} \approx -0.0556$. Player 2 has a unique equilibrium strategy, while player 1 has infinitely many equilibrium strategies parameterized by a single value.

8.2 Experimental setup

We experimented using several of the safe strategies described in Section 5 — RWYYE, Best equilibrium, BEFFE, and BEFEWP. For all algorithms, we used a natural opponent modeling algorithm similar to prior work [3, 5]. We also compare our algorithms to a full best response using the same opponent modeling algorithm. This strategy is not safe and is highly exploitable in the worst case; but it provides a useful metric for comparison.

Our opponent model assumes the opponent plays according to his observed frequencies so far in the game, where we assume that we observe his hand at the end of each game iteration as prior work on exploitation in Kuhn poker has done [5]. We initialize our model by assuming a Dirichlet prior of 5 fictitious hands at each information set at which the opponent has played according to his unique equilibrium strategy.

We adapted all five algorithms to the imperfect-information setting by using the pessimistic update rule described in Algorithm 6. We ran the algorithms against three general classes of opponents. The first class of opponent selects an action uniformly at random at each information set (random opponents were used previously in Kuhn poker [5]). The second opponent class is also static but more sophisticated; at each information set it selects each action at each information with probability chosen randomly but within 0.2 of the equilibrium probability (recall that player 2 has a unique equilibrium strategy). Thus, these opponents play relatively close to optimally, and are perhaps more indicative of realistic suboptimal opponents. The third class of opponents is dynamic. Opponents in that class play the first 100 hands randomly, and then play a true best response (i.e., nemesis strategy) to our player's strategy. So, after the first 100 hands, we make the opponent more powerful than any real opponent could be in practice.

We ran all five algorithms against (the same) 800 random opponents, 2400 sophisticated static opponents, and 800 dynamic opponents. Each match against a single opponent consisted of 1000 hands, and we assume that the hands for both players were dealt identically for each of the algorithms against a given opponent (to reduce variance). For example, suppose algorithm A1 is dealt a K and opponent O is dealt a Q in the first hand of the match. Then in the runs of all other algorithms A against O, A is dealt a K and O is dealt a Q in the first hand.

8.3 Experimental results

The results from our experiments are given in Table 1. Against random opponents, the ordering of the performances of the safe algorithms was RWYWE, BEFEWP, BEFFE, Best Nash (and all of the individual rankings are statistically significant using 95% confidence intervals). Against sophisticated static opponents the rankings of the algorithms' performances were identical, though the only significant result at the 95% level was that RWYWE outperformed Best Nash. (Recall that the value of the game to player 1 is $-\frac{1}{18} \approx -0.0556$, so a negative win rate is not necessarily indicative of losing). In summary, against static opponents, our most aggressive safe exploitation algorithm outperforms the other safe exploitation algorithms that either stay within equilibrium strategies or use exploitation only when enough gifts have been accrued to use full exploitation. Against the dynamic opponents, our algorithms are indeed safe as the theory would predict, while the best response algorithm does very poorly (and worse than the value of the game).

Table 1: Win rate in $/hand of the five algorithms against opponents from each class. The ± given is the 95% confidence interval.

	Random	Sophisticated	Dynamic
RWYWE	0.363 ± 0.003	-0.0096 ± 0.0017	-0.021 ± 0.003
BEFEWP	0.353 ± 0.003	-0.0105 ± 0.0017	-0.020 ± 0.003
BEFFE	0.199 ± 0.003	-0.0111 ± 0.0017	-0.041 ± 0.003
Best Nash	0.143 ± 0003	-0.0135 ± 0.0017	-0.035 ± 0.003
Best response	**0.470 ± 0.003**	**0.0563 ± 0.0018**	**-0.121 ± 0.003**

In some matches, RWYWE steadily accumulates gifts along the way, and k^t increases throughout the match. An example of the graph of profit and k^t for one such opponent is

given in Figure 4. In this situation, the opponent is frequently giving us gifts, and we quickly start playing (and continue to play) a full best response according to our opponent model.

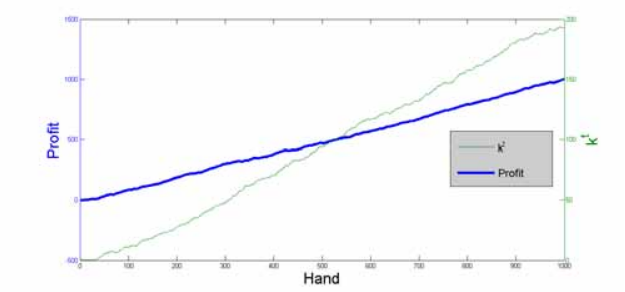


Figure 4: Profit and k^t over the course of a match of RWYWE against a random opponent. Profits are denoted by the thick blue line using the left Y axis, while k^t is denoted by the thin green line and the right Y axis. Against this opponent, both k^t and profits steadily increase.

In other matches, k^t remains very close to 0 throughout the match, despite the fact that profits are steadily increasing; one such example is given in Figure 5. Against this opponent, we are frequently playing an equilibrium or an ϵ-safe best response for some small ϵ, and only occasionally playing a full best response. Note that k^t falling to 0 does not necessarily mean that we are losing or giving gifts to the opponent; it just means that we are not completely sure about our worst-case exploitability, and are erring on the side of caution to ensure safety.

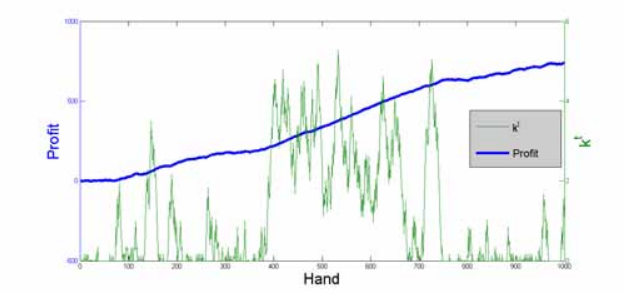


Figure 5: Profit and k^t over the course of a match of RWYWE against a random opponent. Profits are denoted by the thick blue line using the left Y axis, while k^t is denoted by the thin green line and the right Y axis. Against this opponent, k^t stays relatively close to 0 throughout the match, while profit steadily increases.

9. CONCLUSIONS AND FUTURE RESEARCH

We showed that safe opponent exploitation is possible in certain games, disproving a recent conjecture. Specifically, profitable deviations from stage-game equilibrium are possible in games where 'gift' strategies exist for the opponent, which we define formally and fully characterize. We considered several natural opponent exploitation algorithms and showed that some guarantee safety while others do not; for example, risking the amount of profit won so far is not safe in general, while risking the amount won so far *in expectation* is safe. We described how some of these algorithms can be used to convert any opponent modeling algorithm (that is arbitrarily exploitable) into a fully safe opponent exploitation procedure. Next we provided a full characterization of safe algorithms for strategic-form games, which corresponds to precisely the algorithms that are expected-profit safe. We also provided algorithms and full characterizations of safe strategies in sequential games of perfect and imperfect information. In our experiments, against static opponents, several safe exploitation algorithms significantly outperformed an algorithm that selects the best Nash equilibrium strategy; thus we conclude that safe exploitation is feasible and potentially effective in realistic settings. Our most aggressive safe exploitation algorithm outperformed the other safe exploitation algorithms that use exploitation only when enough gifts have been accrued to use full exploitation. In experiments against an overly strong dynamic opponent that plays a nemesis strategy after 100 iterations, our algorithms are indeed safe as the theory would predict, while the best response algorithm does very poorly (and worse than the value of the game).

Several challenges must be confronted before applying safe exploitation algorithms to larger sequential games of imperfect information, such as Texas Hold'em poker. First, the best known technique for computing ϵ-safe best responses involves solving a linear program on par with performing a full equilibrium computation; performing such computations in real time, even in a medium-sized abstracted game, is not feasible in Texas Hold'em. In addition, perhaps performance can be improved if we integrate our algorithms with lower-variance estimators of our expected profits [2].

10. REFERENCES

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 2002.

[2] M. Bowling, M. Johanson, N. Burch, and D. Szafron. Strategy evaluation in extensive games with importance sampling. *ICML*, 2008.

[3] S. Ganzfried and T. Sandholm. Game theory-based opponent modeling in large imperfect-information games. *AAMAS*, 2011.

[4] S. Ganzfried and T. Sandholm. Safe opponent exploitation. *ACM-EC*, 2012.

[5] B. Hoehn, F. Southey, R. C. Holte, and V. Bulitko. Effective short-term opponent exploitation in simplified poker. *AAAI*, 2005.

[6] M. Johanson, M. Zinkevich, and M. Bowling. Computing robust counter-strategies. *NIPS*, 2007.

[7] H. W. Kuhn. Simplified two-person poker. *Contributions to the Theory of Games*, 1950.

[8] P. McCracken and M. Bowling. Safe strategies for agent modelling in games. *AAAI Fall Symposium on Artificial Multi-agent Learning*, 2004.

[9] T. Sandholm. Perspectives on multiagent learning. *Artificial Intelligence*, 171, 2007.

[10] K. Waugh. Abstraction in large extensive games. Master's thesis, University of Alberta, 2009.

Intrinsically Motivated Model Learning for a Developing Curious Agent

Todd Hester
University of Texas at Austin
Department of Computer Science
Austin, TX 78751
todd@cs.utexas.edu

Peter Stone
University of Texas at Austin
Department of Computer Science
Austin, TX 78751
pstone@cs.utexas.edu

ABSTRACT

Reinforcement Learning (RL) agents could benefit society by learning tasks that require learning and adaptation. However, learning these tasks efficiently typically requires a well-engineered reward function. Intrinsic motivation can be used to drive an agent to learn useful models of domains with limited or no external reward function. The agent can later plan on its learned model to perform tasks in the domain if given a reward function. This paper presents the TEXPLORE with Variance-And-Novelty-Intrinsic-Rewards algorithm (TEXPLORE-VANIR), an intrinsically motivated model-based RL algorithm. The algorithm learns models of the transition dynamics of a domain using decision trees. It calculates two different intrinsic rewards from this model: one to explore where the model is uncertain, and one to acquire novel experiences that the model has not yet been trained on. This paper presents experiments demonstrating that the combination of these two intrinsic rewards enables the algorithm to learn an accurate model of a domain with no external rewards and that the learned model can be used afterward to perform tasks in the domain. While learning the model, the agent explores the domain in a developing and curious way, progressively learning more complex skills. In addition, the experiments show that combining the agent's intrinsic rewards with external task rewards enables the agent to learn faster than using external rewards alone.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Reinforcement Learning, Artificial Curiosity, Developmental Learning

1. INTRODUCTION

Reinforcement Learning (RL) agents could be useful in society because of their ability to learn and adapt to new environments and tasks. However, they typically require a well-engineered reward function to specify the task to be learned and enable it to be learned efficiently. Intrinsic motivation can be used to make agents learn more efficiently by augmenting external rewards in a task. They can also be used in the absence of external rewards to drive an agent to learn as much as possible about the world. This learned knowledge can be used later to perform tasks in the world.

This paper presents an intrinsically motivated model-based RL algorithm, called TEXPLORE with Variance-And-Novelty-Intrinsic-Rewards (TEXPLORE-VANIR), that learns a model of a domain without external rewards. The agent is based on a typical model-based RL framework. TEXPLORE-VANIR combines model learning through the use of decision trees with two unique intrinsic rewards calculated from the tree models. TEXPLORE-VANIR uses its decision tree models to calculate two different intrinsic rewards. The first reward is based on *variance* in its models' predictions to drive the agent to explore where its model is uncertain. The second reward drives the agent to *novel* states which are the most different from what its models have been trained on so far. The combination of these two rewards enables the agent to explore in a developing curious way, learning progressively more complex skills in the domain. In addition, the agent can learn an accurate and useful model of the domain.

This paper presents two main contributions:

1. Novel methods for obtaining intrinsic rewards from a decision tree based model of the world.
2. The TEXPLORE-VANIR algorithm for intrinsically motivated model learning.

Section 4 presents experiments showing that the algorithm: 1) explores in a developing, curious way; 2) learns a more accurate model than other approaches; and 3) can use its learned model later to perform tasks specified by a reward function. In addition, it shows that the agent can use the intrinsic rewards in conjunction with external rewards to learn a task faster than if using external rewards alone.

2. BACKGROUND

This section presents background in two main areas. Section 2.1 covers background material on Reinforcement Learning (RL), which is used as the framework for TEXPLORE-VANIR. Section 2.2 describes background on other approaches to Intrinsic Motivation (IM).

2.1 Reinforcement Learning

We adopt the standard Markov Decision Process (MDP) formalism for this work [23]. An MDP is defined by a tuple $\langle S, A, R, T \rangle$, which consists of a set of states S, a set of actions A, a reward function $R(s, a)$, and a transition function $T(s, a, s') = P(s'|s, a)$. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives

a reward $R(s,a)$ and reaches a new state s', determined from the probability distribution $P(s'|s,a)$. Many domains utilize a factored state representation, where the state s is represented by a vector of n state variables: $s = \langle x_1, x_2, ..., x_n \rangle$. A policy π specifies for each state which action the agent will take.

The value $Q^\pi(s,a)$ of a given state-action pair (s,a) is an estimate of the expected future reward that can be obtained from (s,a) when following policy π. The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's lifetime. The optimal value function $Q^*(s,a)$ provides maximal values in all states and is determined by solving the Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a'), \quad (1)$$

where $0 < \gamma < 1$ is the discount factor. The optimal policy π is then as follows:

$$\pi(s) = \text{argmax}_a Q^*(s,a). \quad (2)$$

RL methods fall into two general classes: model-based and model-free methods. Model-based RL methods learn a model of the domain by approximating $R(s,a)$ and $P(s'|s,a)$ for each state and action. The agent can then calculate a policy (i.e. plan) using this model. Model-free methods update the values of actions only when taking them in the real task. One of the advantages of model-based methods is their ability to plan multi-step exploration trajectories. The agent can plan a policy to reach intrinsic rewards added into its model to drive exploration to interesting state-actions.

2.2 Intrinsic Motivation

This section presents background on Intrinsic Motivation (IM). Work on intrinsically motivated agents originally came from two different goals [16]. The first goal is for the intrinsic motivation to drive the agent to maximize its knowledge about the world and its ability to control it. The second goal is to enable cumulative, open-ended learning on robots.

Work towards the first goal fits nicely within the RL framework. Intrinsic rewards can be used with model-free RL agents to drive them to update their value function (and thus learn a good policy) as quickly as possible [5]. For model-based agents, rewards can be used to drive the agent to learn an accurate model as efficiently as possible [2].

These different approaches demonstrate that the correct intrinsic motivation is dependent on the type of algorithm. For example, with a Q-LEARNING agent [25], it makes sense to give intrinsic rewards for where the value backups will have the largest effect, as done in [5]. When learning with a tabular model, the agent must gain enough experiences in each state-action to learn an accurate model of it. Thus it makes sense to use intrinsic motivation to drive the agent to acquire these experiences, as done by R-MAX [2].

This work takes the approach of using a model-based RL algorithm in a domain with no external rewards. This approach can be thought of as a pure exploration problem, where the agent's goal is simply to learn as much about the world as possible. Bayesian methods such as Duff's optimal probe [7] attempt to solve this problem optimally, but are computationally intractable. TEXPLORE-VANIR extends a model-based RL algorithm called TEXPLORE to use intrinsic motivation. Its intrinsic rewards should drive it to quickly learn an accurate model in a domain with no external rewards.

3. ALGORITHM

This section presents the TEXPLORE with Variance-And-Novelty-Intrinsic-Rewards algorithm (TEXPLORE-VANIR), an algorithm for an intrinsically motivated curious agent. First we present some essential properties that the agent should have, before presenting its specific approach to model learning and intrinsic motivation.

Our goal is to develop an RL agent that uses intrinsic rewards to explore in a developing curious way and learn a useful model of the domain's transition dynamics. To this end, we have the following desiderata for such an algorithm:

1. The algorithm should be model-based, both to enable multi-step exploration trajectories and to allow the agent to use the learned model later to perform tasks.
2. It should incorporate generalization into its model learning to learn the model quickly.
3. It should not be required to visit every state-action in the task.
4. It should decide which areas of the state space are interesting to it without having to sample them first.

Now, we present TEXPLORE-VANIR, an algorithm that has all of these desired properties. It uses decision trees to learn models that generalize predictions across state-actions in the domain, similar to SPITI [6] and TEXPLORE [10]. It incorporates intrinsic rewards that drive the agent towards two types of state-actions: ones where its model is uncertain and ones where its model may have generalized incorrectly. TEXPLORE-VANIR extends the TEXPLORE model-based RL algorithm [10] by incorporating intrinsic rewards that motivate the agent to explore in a developing, curious way to learn useful models of domains with no external rewards.

TEXPLORE-VANIR follows the typical approach of a model-based RL agent. It plans a policy using its learned model (including intrinsic rewards), takes actions following that policy, acquiring new experiences which are used to improve its model, and repeats. In order to be applicable to learning on robots, TEXPLORE-VANIR uses the Real-Time Model Based Architecture presented in [9]. This architecture uses approximate planning with UCT [13] and parallelizes the model learning, planning, and acting such that the agent can take actions in real-time at a specified frequency.

The following section presents details on TEXPLORE's model learning approach, which is used by TEXPLORE-VANIR. Then, Section 3.2 describes how the learned models are used in calculating intrinsic rewards for exploration by TEXPLORE-VANIR.

3.1 Model Learning

Making the intrinsically motivated agent model-based enables it to: 1) plan multi-step exploration trajectories; 2) learn faster than model-free approaches; and 3) use the learned model to solve tasks given to it after its learning. It is desirable for the model to generalize the learned transition and reward dynamics across state-actions. This generalization enables the model to make predictions about unseen or infrequently visited state-actions, and therefore not

have to visit each and every one. Thus, TEXPLORE-VANIR approaches the model learning task as a supervised learning problem, with the current state and action as the input, and the next state as the output to be predicted. In order to improve generalization, TEXPLORE-VANIR follows the approach of [15] and [11] in predicting the relative transitions $(s' - s)$ between states rather than absolute outcomes.

Like Dynamic Bayesian Network (DBN) based RL algorithms [8, 4], TEXPLORE-VANIR learns a model of the factored domain by learning a separate prediction for each of the n state features. TEXPLORE-VANIR assumes that each of the state variables transitions independently, as DBN-based methods do. Therefore, the separate feature predictions can be combined to create a prediction of the complete state vector.

TEXPLORE-VANIR follows the approach of TEXPLORE [10] by using a decision tree to predict the change in each state feature. The decision trees are learned using an implementation of Quinlan's C4.5 algorithm [18]. The C4.5 algorithm builds a tree that splits on the inputs (current state features and action), with each leaf of the tree making a different prediction. The splits are formed by choosing the feature with the highest information gain. Decision trees were chosen because they generalize broadly at first, but can be refined with training to make accurate predictions for individual state-actions.

Another desirable property for the model learning algorithm to have is a measure of uncertainty in the model's predictions. TEXPLORE-VANIR accomplishes this goal by building multiple hypotheses of the true model of the domain in the form of a random forest. The variance of the different trees' predictions can be used as a measure of the uncertainty in the model. A random forest is a collection of m decision trees, each of which differ because they are trained on a random subset of experiences and have some randomness when choosing splits at the decision nodes. The agent then plans over the average of the predictions made by each tree in the forest. Random forests have been proven to converge with less generalization error than individual tree models [3]. A complete diagram of TEXPLORE-VANIR's model learning process is shown in Figure 1.

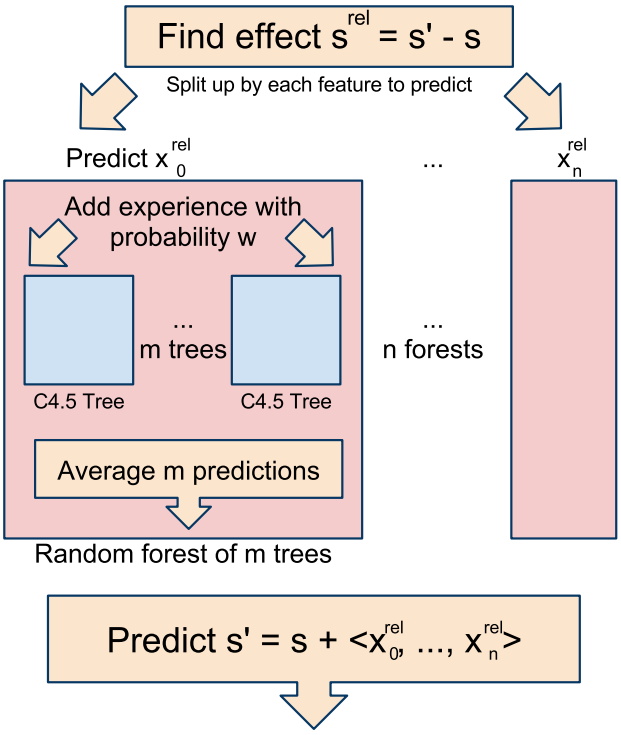


Figure 1: *Model Learning.* This is how the algorithm learns a model of the domain. The agent calculates the difference between s' and s as the transition effect s^{rel}. Then it splits up the state vector and learns a random forest to predict each state feature. Each random forest is made up of stochastic decision trees, which get each new experience with probability w. The random forest's predictions are made by averaging each tree's predictions, and then the predictions for each feature are combined into a complete model of the domain.

3.2 Intrinsic Motivation

The main contribution of this paper is a method for extending the model-based TEXPLORE algorithm for learning specific RL tasks to the TEXPLORE-VANIR algorithm for exploration by a curious agent. This curiosity is operationalized via intrinsic rewards for 1) preferring to explore areas of the state space where there is a large degree of uncertainty in the model, and 2) preferring regions of the state space that are far from previously explored areas (regardless of how certain the model is).

The variance of the predictions of each of the trees in the forest can be used to motivate the agent towards the state-actions where its models disagree. These state-actions are the ones where there are still multiple hypotheses of the true model of the domain that make differing predictions. TEXPLORE-VANIR calculates a measure of the variance in the predictions of the change in each state feature for a given state-action:

$$D(s,a) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{m} D_{KL}(P_j(x_i^{rel}|s,a)||P_k(x_i^{rel}|s,a)), \quad (3)$$

where for every pair of models (j and k) in the forest, it sums the KL-divergences between the predicted probability distributions for each feature i. This measure $D(s,a)$ tells us how much the predictions of the different models disagree. This measure is different than just measuring where the predictions are noisy, as $D(s,a)$ will be 0 if all the tree models predict the same stochastic outcome distribution. An intrinsic reward proportional to this variance measure, called the VARIANCE-REWARD, can then be incorporated into the agent's model for planning:

$$R(s,a) = vD(s,a), \quad (4)$$

where v is a coefficient determining how big this reward should be. This reward can be combined with other rewards (intrinsic or extrinsic) to drive the agent.

This reward will drive the agent to the state-actions where its models have not yet converged to a single hypothesis of the world's true dynamics. However, there will still be cases where all of the agent's models are making incorrect predictions. Therefore, TEXPLORE-VANIR also needs a measure of how likely it is for the model's predictions to be incorrect. For the decision tree model that TEXPLORE-VANIR uses, the model is more likely to be incorrect when it has to general-

State Features	ID, X, Y, KEY, LOCKED, RED-E, RED-W, RED-N, RED-S, GREEN-E, GREEN-W, GREEN-N, GREEN-S, BLUE-E, BLUE-W, BLUE-N, BLUE-S,
Actions	EAST, WEST, NORTH, SOUTH, PRESS, PICKUP

Table 1: Properties of the Light World domain.

ize its predictions farther from the experiences it is trained on. Therefore, TEXPLORE-VANIR utilizes a second intrinsic reward based on the L_1 distance in feature space from a given state-action and the nearest one that the model has been trained on. If X is a set of all seen state-actions, then the L_1 distance from a given state-action to the nearest seen state-action is:

$$\delta(s,a) = \min_{\langle s_x, a_x \rangle \in X} || \langle s,a \rangle - \langle s_x, a_x \rangle ||_1. \quad (5)$$

A reward proportional to this distance, called the NOVELTY-REWARD, drives the agents to explore the state-actions that are the most novel compared to the state-actions it has seen before:

$$R(s,a) = n\delta(s,a), \quad (6)$$

where n is a coefficient determining how big this reward should be. One nice property of this reward is that given enough time, it will drive the agent to explore *all* the state-actions in the domain, as any unvisited state-action is different in some feature from the visited ones. However, it will start out driving the agent to explore the state-actions that are the most different from ones it has seen.

The TEXPLORE with Variance-And-Novelty-Intrinsic-Rewards algorithm (TEXPLORE-VANIR) is completed by combining these two intrinsic rewards together. They can be combined with different weightings of their coefficients (v and n) to drive the agent to both explore novel state-actions where its model may have generalized incorrectly and state-actions where its model is uncertain. In addition, these two intrinsic rewards can be combined with an external reward defining a task. A combination of these two intrinsic rewards should drive the agent to explore in a developing and curious way: seeking out novel and interesting state-actions, while exploring increasingly complex parts of the domain.

4. EMPIRICAL RESULTS

Evaluating an agent's curiosity is not as straightforward as evaluating a standard RL agent on a specific task. Rather than attempting to accrue reward on a specific task, a curious agent's goal is better stated as preparing itself for any task. We therefore evaluate TEXPLORE-VANIR in four ways on a complex domain with no external rewards. First, we measure the accuracy of the agent's learned model in predicting the domain's transition dynamics. Second, we test whether the learned model can be used to perform tasks in the domain when given a reward function. Third, we examine the agent's exploration to see if it is exploring in a developing, curious way. Finally, we demonstrate that TEXPLORE-VANIR can combine its intrinsic rewards with external rewards to learn faster than if it was given only external rewards. These results demonstrate that the intrinsic rewards and model learning approach TEXPLORE-VANIR uses are sufficient for the agent to explore in a developing curious way and to learn a transition model that is useful for performing tasks on the domain.

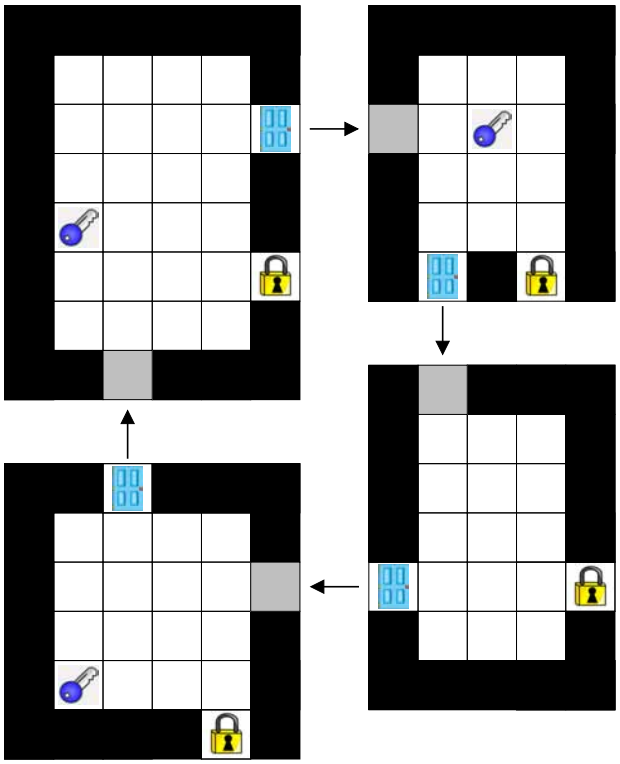


Figure 2: The Light World domain. In each room, the agent must navigate to the key, pickup they key, navigate to the lock, press it, and then navigate to and exit through the door to the next room.

The agent is tested on the Light World domain, presented in [14], and shown in Figure 2. In this domain, the agent goes through a series of rooms. Each room has a door, a lock, and possibly a key. The agent must go to the lock and press it to open the door, at which point it can then leave the room. It cannot go back through the door in the opposite direction. If a key is present, it must pickup the key before pressing the lock. Open doors, locks, and keys each emit a different color light that the agent can see. The agent has sensors that detect each color light in each cardinal direction. The sensors have a maximal value of 1 when the agent is at the light, and their values go down to 0 when the light is 20 steps away. The agent's state is made up of 17 different features: its X and Y location in the room, the ID of the room it is in, whether it has the KEY, whether the door is LOCKED, as well as the values of the 12 light sensors, which detect each of the three lights in the four cardinal directions. The agent can take six possible actions: it can move in each of the four cardinal directions, PRESS the lock, or PICKUP the key. The first four actions are stochastic; they move the agent in the intended direction with probability 0.8 and to either side with probability 0.1 each. The PRESS and PICKUP actions are only effective when the agent is on top of the lock and the key, respectively. The agent starts in a random state in the top left room in the domain, and can proceed through the rooms indefinitely. The states features and actions of the domain are listed in Table 1.

This domain is well-suited for this task because the domain has a rich feature space and complex dynamics. There

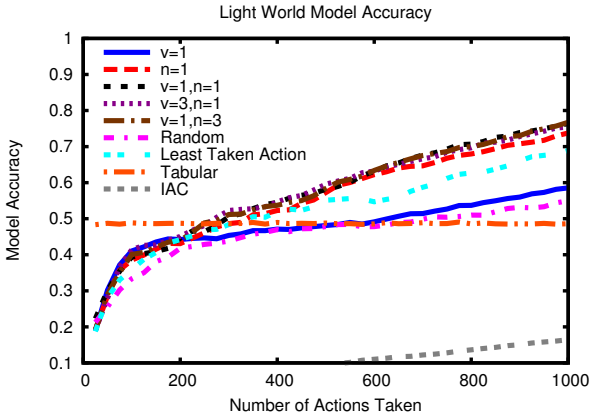


Figure 3: Model accuracy of each algorithm plotted versus number of steps the agent has taken, averaged over 30 trials. The versions of TEXPLORE-VANIR with $n >= 1$ learn the most accurate models.

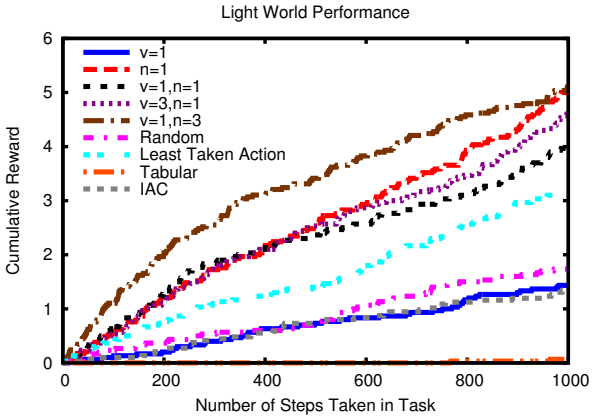


Figure 4: Cumulative rewards received by each algorithm over the 1000 steps of the task, averaged over 30 trials. TEXPLORE-VANIR with $v = 1, n = 3$ receives the most reward.

are simple actions that move the agent in different directions, as well as more complex actions (PICKUP and PRESS) that interact with objects in different ways. There is a progression of the complexity of the uses of these two actions. Picking up the key is easier than pressing the lock, as the lock requires the agent to have already picked up the key and not yet unlocked the door.

Five versions of the algorithm using the following sets of coefficients are compared:

1. v = 1, n = 0
2. v = 0, n = 1
3. v = 1, n = 1
4. v = 3, n = 1
5. v = 1, n = 3

In addition, TEXPLORE-VANIR is tested against the following agents:

1. Agent which selects actions *randomly*
2. Agent which selects the *least taken action* at each state
3. Agent which acts randomly with a *tabular* model
4. Intelligent Adaptive Curiosity (IAC) [17]

These four algorithms provide two different ways to explore using TEXPLORE-VANIR's tree model, as well as an algorithm using a tabular model, and an existing model for intrinsically motivated learning (IAC).

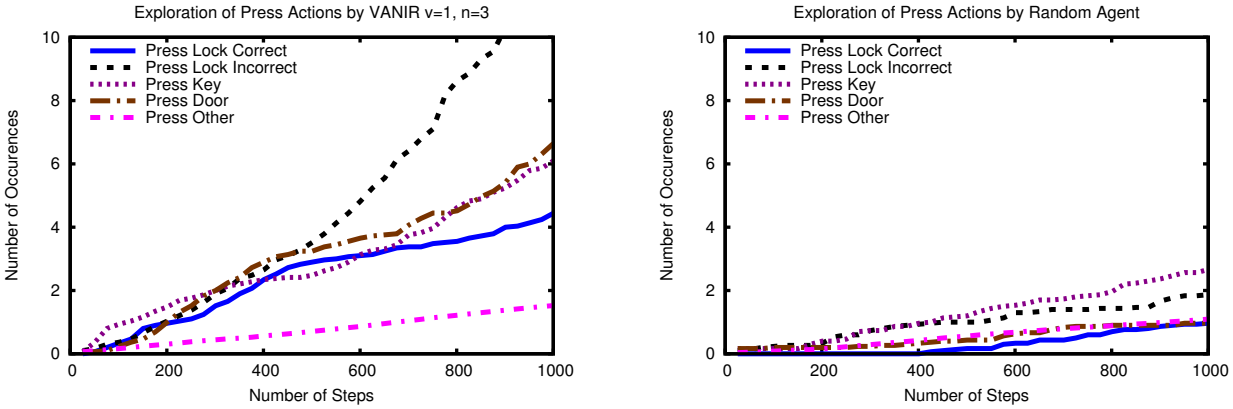
Intelligent Adaptive Curiosity (IAC) [17] is a method for providing intrinsic reward to encourage a developing agent to explore. Their approach does not adopt the RL framework, but is similar in many respects. IAC splits the state space into regions and attempts to learn a model of the transition dynamics in each region. It maintains an error curve for each region and uses the slope of this curve as the intrinsic reward for the agent, driving the agent to explore the areas where its model is improving the most. Since this approach is not using the RL framework, it selects actions only to maximize the immediate reward, rather than the discounted sum of future rewards. Another drawback of this approach is that before intrinsic rewards from any region can be calculated, the agent must take actions from that part of the domain a few times to build an error curve for its model of that region.

All the algorithms are run in the Light World domain for 1000 steps without any external reward. During this phase, the agent is free to play and explore in the domain, all the while learning a model of the dynamics of this world. For some of the experiments, a second phase of the experiment is run with external rewards to see if the agent's learned model is useful. All of the algorithms use the RTMBA parallel architecture [9] and take actions at 1 Hz.

First, the accuracy of the agent's learned model is examined. After every 25 steps, 5000 state-actions from the domain are randomly sampled and the model's predicted probabilities of each next state are compared with the true next state probabilities. Figure 3 shows 1.0 minus the average error in the predicted probabilities of the model learned by each agent. This figure shows that the versions of TEXPLORE-VANIR with $n >= 1$ learn the most accurate models, with the version with $v = 1, n = 3$ having the highest accuracy. IAC, the TABULAR model, and the random agent have the poorest model accuracy.

While all the versions of TEXPLORE-VANIR with $n >= 1$ reach similar levels of model accuracy, it is more important for the algorithm to be accurate in the interesting and useful parts of the domain than for it to be accurate about every state-action. Therefore, we want to test if the learned models are useful to perform a task. After the algorithms learned models without rewards for 1000 steps, they are provided with a reward function for a task. The task is for the agent to continue moving through the rooms (requiring it to use the keys and locks). The reward function is a reward of 1 for moving from one room to the next, and a reward of 0 for all other actions. The agents are tested using their learned transition models, the given external reward function, and no intrinsic rewards for 1000 steps on the task.

Figure 4 shows the cumulative external reward received



(a) TEXPLORE-VANIR with $v = 1, n = 3$

(b) Random Agent.

Figure 5: This plot shows the number of times that TEXPLORE-VANIR with $v = 1, n = 3$ and a Random Agent select the press action in various states over 1000 steps in the task with no external rewards, averaged over 30 trials. Note that the random agent attempts the press action much less than TEXPLORE-VANIR does. TEXPLORE-VANIR starts out trying to press the key, which is the easiest object to find, and eventually does learn to press the lock, but has difficulty learning when to press the lock (it must be with the key but without the door already being open). The agent does not try calling the press action on random states very often. In contrast, the random agent calls press action on random states more often than it calls it correctly on the lock.

by each algorithm over the 1000 steps of the task. For comparison, an optimal policy would be expected to receive a cumulative reward of 60.2 over the 1000 steps. This figure shows more variation in the abilities of each agent to perform the task. Although all the versions of TEXPLORE-VANIR with $n >= 1$ have similar model accuracies, TEXPLORE-VANIR with $v = 1, n = 3$ has the greatest cumulative reward for most of the steps. At the end of the 1000 steps, however, TEXPLORE-VANIR with $v = 1, n = 3$ is equaled by TEXPLORE-VANIR with $n = 1$. In comparison, the versions exploring randomly or with $v = 1$ perform much worse than TEXPLORE-VANIR with $n >= 1$ and all the methods using tree models perform much better than the tabular model.

Next, the exploration of the TEXPLORE-VANIR agent with $v = 1, n = 3$ is examined. In addition to learning an accurate and useful model of the task, we desire the agent to exhibit a developing curiosity. Precisely, the agent should progressively learn more complex skills in the domain, rather than explore randomly or exhaustively. In this domain, the agent should first learn to pick up the key, then learn how to use the key at the lock, and then how to go through the door to the next room.

Figures 5(a) and 5(b) show the number of times that TEXPLORE-VANIR with $v = 1, n = 3$ and the random agent select the PRESS action in various states over 1000 steps in the task with no external rewards, averaged over 30 trials. Comparing the two figures clearly shows that TEXPLORE-VANIR calls the PRESS action many more times than the random agent. Figure 5(a) also shows that TEXPLORE-VANIR tries PRESS on objects more often than on random states in the domain. In contrast, Figure 5(b) shows that the random agent tries PRESS on arbitrary states in the domain more often than it uses it correctly.

Analyzing the exploration of TEXPLORE-VANIR further, Figure 5(a) shows that it initially tries PRESS on the key, which is the easiest object to access, then tries it on the lock, and then on the door (which is not visible until unlocked). The figure also shows that TEXPLORE-VANIR takes longer to learn the correct dynamics of the lock, as it continues to PRESS the lock incorrectly, either without the key or with the door already unlocked. These plots show that TEXPLORE-VANIR is acting in an intelligent, curious way, trying actions on the objects in order from the easiest to hardest to access, and going back to the lock repeatedly to learn its more complex dynamics.

Finally, not only should the agent's intrinsic rewards be useful when learning in task without external rewards, they should also make an agent in a domain with external rewards learn more efficiently. For this experiment, the algorithms are run for 1000 steps with their intrinsic rewards added to the previously used external reward function that rewards moving between rooms. Figure 6 shows the cumulative external reward received by each agent over the 1000 steps of the task. The versions of TEXPLORE-VANIR with both $n >= 1$ and $v >= 1$ receive the most reward, with TEXPLORE-VANIR with $v = 1, n = 3$ once again performing the best, receiving an average of 4.2 reward over the 1000 steps. In contrast, the agent using only external rewards (TEXPLORE-VANIR with $v = 0, n = 0$) receives only an average of 0.07 reward over the 1000 steps, and the agent using a tabular model receives 0 reward. These results demonstrate that TEXPLORE-VANIR's intrinsic rewards can also be used to speed up learning in tasks that already provide external rewards.

These results show that TEXPLORE-VANIR, in particular with $v = 1, n = 3$, out-performs all other algorithms on the four evaluations. It learns more accurate models than the other approaches when given no external reward and can use its learned model to perform better on a task than the other methods as well. It explores the domain in a curious manner progressing from state-actions with easier dynamics to those that are more difficult. Finally, TEXPLORE-VANIR

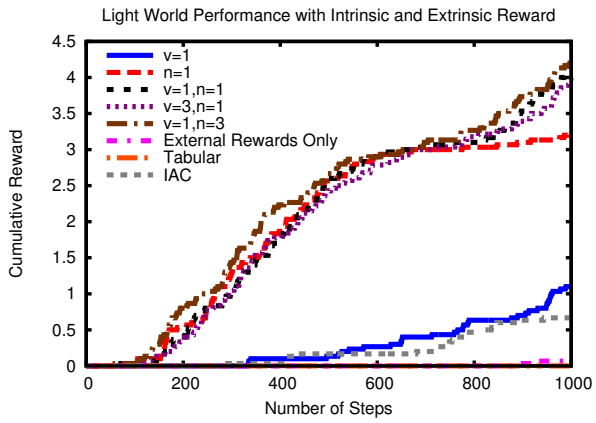


Figure 6: Cumulative rewards received by each algorithm, using intrinsic and external rewards combined, over the 1000 steps of the task, averaged over 30 trials. The versions of TEXPLORE-VANIR with both $n >= 1$ and $v >= 1$ receive the most reward, while the agent using only external rewards performs very poorly.

can use its intrinsic rewards to speed up learning in a case where it is given external rewards.

5. RELATED WORK

This section reviews related work on intrinsic motivation, with a focus on work which is built within the RL framework.

Schmidhuber [19] tries to drive the agent to where the model has been improving the most, rather than trying to estimate where the model is poorest. The author takes a traditional model-based RL method, and adds a confidence module, which is trained to predict the absolute value of the error of the model. This module could be used to create intrinsic rewards encouraging the agent to explore high-error state-action pairs, but then the agent would be attracted to noisy states in addition to poorly-modeled ones. Instead the author adds another module that is trained to predict the changes in the confidence module outputs. Using this module, the agent is driven to explore the parts of the state space that most improve the model's prediction error.

IAC [17] (described in the previous section) and later Robust-IAC [1] are based on a similar idea. They are both methods that encourage a developing agent to explore areas of the state space where the model is improving the most. These approaches do not use the RL framework and have no way of incorporating external rewards, but could be used to provide intrinsic rewards to an existing RL agent. Both of these last two algorithms require the agent to have sampled part of the state space before it can judge if intrinsic rewards should be provided in that region.

R-MAX [2] is a reinforcement learning method that explores efficiently through the use of intrinsic rewards. The algorithm learns a maximum-likelihood tabular model of the task. The model used by the algorithm provides the maximum reward in the domain, r_{max}, as an intrinsic reward to any state-actions that have been visited less than m times. This reward drives the agent to visit each state-action m times so that the agent can learn an accurate model of each state-action. The algorithm is guaranteed to learn an optimal policy in a number of time steps polynomial in the number of state-actions in the domain.

Jonsson and Barto [12] take a similar approach to TEXPLORE-VANIR, in that they also learn trees to model the domain. Their method learns conditional trees using Bayesian Information Criterion to perform splits. Since having a uniform distribution over input values will provide the best information for making splits in the tree, their method provides intrinsic motivation for actions that would increase the uniformity of the inputs to the tree. This reward only drives local exploration, but does enable the agent to quickly learn accurate models of certain tasks. This work was extended to perform more global exploration by adding options to set each state feature to any possible value [24]. The agent could then select options to set features to values where it could then take actions to better improve the uniformity of input features to its trees. However, this approach assumes that the agent can set each feature of the domain independently and learn options to do so.

Singh, Barto, and Chentanez [20] present an approach to learning a broad set of reusable skills in a playroom domain. They learn option models for a variety of skills and show that the agent progresses from learning easier to more difficult skills. However, the skills the agent is to learn are predefined, rather than being entirely intrinsically motivated.

Simsek and Barto [5] present an approach for the pure exploration problem, where there is no concern with receiving external rewards. They provide a Q-LEARNING agent [25] with intrinsic rewards for where its value function is most improving. This reward speeds up the agent's learning of the true task. However, such a reward to make the agent perform more value backups on its value function is not necessary for model-based agents, which can perform all the necessary backups using its model without having to re-visit each state-action. Stout and Barto [22] extend this work to the case where the agent is learning multiple tasks and must balance the intrinsic rewards that promote the learning of each skill. These algorithms still require an external reward, as the intrinsic reward is speeding up the learning of the task defined by the external reward function.

Singh et al. [21] present an interesting perspective on intrinsically motivated learning. They argue that in nature, intrinsic rewards come from evolution and exist to help us perform any task. Agents using intrinsic rewards combined with external rewards should be able to perform better on tasks than those using solely external rewards. For two different algorithms and tasks, they search over a broad set of possible task and agent specific intrinsic rewards and find rewards that make the agent learn faster than if it solely used external rewards.

6. DISCUSSION

This paper presents the TEXPLORE-VANIR algorithm for intrinsically motivated learning. This algorithm combines decision tree based model learning with two novel intrinsic rewards. One reward drives the agent to where the model is uncertain in its predictions, and the second drives the agent to acquire novel experiences that its model has not been trained on. Experiments show empirically that TEXPLORE-VANIR can learn accurate and useful models in a domain with no external rewards. In addition, TEXPLORE-VANIR's intrinsic rewards drive the agent to learn in a developing

and curious way, progressing from learning easier to more difficult skills. TEXPLORE-VANIR can also combine its intrinsic rewards with external task rewards to learn a task faster than using external rewards alone.

One of the major advantages of TEXPLORE-VANIR in comparison to others is that it does not require the agent to sample each region of the state space first to determine if they are interesting. It can decide certain states are interesting because of the novelty of their state features without having to visit that region of the state space first. The agent takes an intelligent approach to exploration, building multiple hypotheses of the true dynamics of the domain and driving itself through the variance motivation to explore where the various hypotheses make different predictions. In addition, the agent continues seeking out novel states that may provide different dynamics from what it has seen before.

Our main interest in the pursuit of this research is to apply it to developmental learning on robots. Therefore, one area of future work is to extend the algorithm to work in large continuous state spaces, such that it can apply on a robot. Once the algorithm is extended in this way, we plan to perform experiments on humanoid robots such as the Aldebaran Nao.

7. REFERENCES

[1] A. Baranes and P. Y. Oudeyer. R-IAC: Robust Intrinsically Motivated Exploration and Active Learning. *IEEE Transactions on Autonomous Mental Development (TAMD)*, 1(3):155–169, Oct. 2009.

[2] R. Brafman and M. Tennenholtz. R-Max - a general polynomial time algorithm for near-optimal reinforcement learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 953–958, 2001.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] D. Chakraborty and P. Stone. Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning (ICML)*, June 2011.

[5] O. Şimşek and A. G. Barto. An intrinsic reward mechanism for efficient exploration. In *ICML*, pages 833–840, 2006.

[6] T. Degris, O. Sigaud, and P.-H. Wuillemin. Learning the structure of factored Markov Decision Processes in reinforcement learning problems. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML)*, pages 257–264, 2006.

[7] M. Duff. Design for an optimal probe. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pages 131–138, 2003.

[8] C. Guestrin, R. Patrascu, and D. Schuurmans. Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML)*, pages 235–242, 2002.

[9] T. Hester, M. Quinlan, and P. Stone. RTMBA: A real-time model-based reinforcement learning architecture for robot control. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[10] T. Hester and P. Stone. Real time targeted exploration in large domains. In *Proceedings of the Ninth International Conference on Development and Learning (ICDL)*, August 2010.

[11] N. Jong and P. Stone. Model-based function approximation for reinforcement learning. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2007.

[12] A. Jonsson and A. G. Barto. Active learning of dynamic bayesian networks in markov decision processes. In *SARA*, pages 273–284, 2007.

[13] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML)*, 2006.

[14] G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 895–900, 2007.

[15] B. Leffler, M. Littman, and T. Edmunds. Efficient reinforcement learning with relocatable action models. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 572–577, 2007.

[16] M. Lopes and P.-Y. Oudeyer. Guest editorial: Active learning and intrinsically motivated exploration in robots: Advances and challenges. *IEEE Transactions on Autonomous Mental Development (TAMD)*, 2(2):65–69, 2010.

[17] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Trans. Evolutionary Computation*, 11(2):265–286, 2007.

[18] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[19] J. Schmidhuber. Curious model-building control systems. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1458–1463. IEEE, 1991.

[20] S. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS) 17*, 2005.

[21] S. P. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development (TAMD)*, 2(2):70–82, 2010.

[22] A. Stout and A. Barto. Competence progress intrinsic motivation. In *Proceedings of the Ninth International Conference on Development and Learning (ICDL)*, pages 257–262, 2010.

[23] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[24] C. M. Vigorito and A. G. Barto. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development (TAMD)*, 2(2), 2010.

[25] C. Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Informed Initial Policies for Learning in Finite Horizon Dec-POMDPs

Landon Kraemer and Bikramjit Banerjee
School of Computing
University of Southern Mississippi
118 College Dr. #5106
Hattiesburg, MS 39406-0001
landon.kraemer@eagles.usm.edu, bikramjit.banerjee@usm.edu

ABSTRACT

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a formal model for planning in cooperative multiagent systems where agents operate with noisy sensors and actuators, and local information. Prevalent Dec-POMDP solution techniques have mostly been centralized and have assumed knowledge of the model. In real world scenarios, however, solving centrally may not be an option and model parameters may be unknown. To address this, we propose a distributed, model-free algorithm for learning Dec-POMDP policies, in which agents take turns learning, with each agent not currently learning following a static policy. For agents that have not yet learned a policy, this static policy must be *initialized*. We propose a principled method for learning such initial policies through interaction with the environment. We show that by using such *informed initial policies*, our alternate learning algorithm can find near-optimal policies for three benchmark problems.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Experimentation

Keywords

Reinforcement learning, Decentralized POMDPs

1. INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a formal model for planning in cooperative multiagent systems where agents operate with noisy sensors and actuators, and local information. While many techniques have been developed for solving Dec-POMDPs exactly and approximately, they have been primarily centralized and reliant on full knowledge of the model parameters. But in real world scenarios, model parameters may not be known a priori, and centralized solvers fail to decentralize the policy computation.

Simple distributed reinforcement learning, particularly Q-learning [15], can address both of the above limitations. Q-learning agents can learn mappings from their own action-observation histories (H_i) to their own actions (A_i), via a quality function ($Q_i : H_i \times A_i \mapsto \mathbb{R}$) that evaluates the long-term effects of selecting an action after observing an individual action-observation history. Since agents cannot observe states or each others' actions directly, *independent learning* [5] is a more reasonable approach than joint learning. Independent learners ignore the actions and observations of the other learners and simply learn their own Q-functions directly.

We investigate two ways of applying independent Q-learning to solving finite-horizon Dec-POMDP problems: one in which agents learn concurrently (Q-Conc), and one in which agents take turns to learn the best responses to each other's policies (Q-Alt). In the latter method, each agent that has not yet taken a turn learning needs an initial policy to follow. There are simple ways to choose an arbitrary initial policy in constant time, e.g., choose a policy at random or follow a uniform stochastic policy, but such methods do not consider the transition, observation, or reward structure of the Dec-POMDP. As the main contribution of this paper, we propose a simple and principled approach to building a finite initial joint policy that is based upon the transition and reward (but not the observation) functions of a Dec-POMDP. To lay the groundwork, we first discuss how this initial joint policy can be *computed* in a centralized, model-based fashion. We then discuss how agents can *learn* such policies in a distributed, model-free manner, and then demonstrate for three benchmark problems that Q-Alt initialized with this *informed policy* (QIP-Alt) produces better joint policies than Q-Conc and Q-Alt initialized with uniform stochastic policies (QSto-Alt) and Q-Alt initialized with randomly-chosen pure policies (QRand-Alt).

This work extends the short paper [8], and the informed policy that we propose has also been used for initialization in another successful Monte Carlo based approach called MCQ-Alt [2].

2. BACKGROUND

In this section we introduce the POMDP and the Dec-POMDP formalisms, both of which will be used in our work.

2.1 POMDPs

We can define a partially-observable Markov decision process (POMDP) as a tuple $\langle S, A, P, R, \Omega, O \rangle$, where:

- S is a finite set of (unobservable) environment states.
- A is a finite set of actions.

- $P(s'|s,a)$ gives the probability of transitioning to state $s' \in S$ when action $a \in A$ is executed in state $s \in S$.
- $R(s,a)$ gives the reward the agent receives upon executing action $a \in A$ in state $s \in S$.
- Ω is a finite set of observations.
- $O(\omega|s',a)$ gives the probability of observing $\omega \in \Omega$ if the current state is $s' \in S$ and the previous action was $a \in A$.

POMDP is an extension of the MDP formalism, where only certain features of the environment may be directly observed by an agent. That is, the agent may not directly know the state and must instead use observations received from the environment and the transition and observation probabilities to maintain a *belief* over the environment states and act rationally under this uncertainty.

An agent can interact with a POMDP for a finite or infinite number of steps, or *horizon*. At each step, the agent chooses an action and receives reward based upon the environment state and the action executed, and it receives an observation based upon the action executed and the resulting state. At a given step, the agent knows all of the actions it has executed and all of the observations it has received up to that point, and the goal is then to find a policy which maps action-observation *histories* to actions, such that the reward the agent expects to receive is maximized. More formally, if the t step history of an agent is $h_t = \langle a_1, \omega_1, \ldots, a_t, \omega_t \rangle$, then its policy must map this action-observation history to some action that the agent will execute in the next step, a_{t+1}.

For a finite horizon, such a policy can be represented as a tree where each level of the tree corresponds to a step and every node is labeled with an action to be performed. If T represents the horizon, then each node of depth $d < T$ has $|\Omega|$ children, i.e. one child for each observation. An agent follows a tree policy by executing the action that labels the root node, and then by following the subtree policy along the branch corresponding to the observation it receives.

Figure 1 gives an example policy as both an action-observation history to action mapping (left) and a policy tree (right) for the single agent tiger problem introduced in [7] with T=3. In the tiger problem, $\Omega = \{HearLeft(HL), HearRight(HR)\}$ and $A = \{Listen(L), OpenLeft(OL), OpenRight(OR)\}$. An agent following this policy will always listen twice. If it hears the tiger behind the same door twice, it will open the opposite door. If the agent receives mixed observations, it will not open a door. Note that a single policy may not be able to map many histories to actions because they do not partake in the policy, e.g., histories beginning with the agent opening a door are not a part of the policy in Figure 1.

$$
\begin{array}{rcl}
() & \rightarrow & \text{L} \\
(L, HL) & \rightarrow & \text{L} \\
(L, HL, L, HL) & \rightarrow & \text{OR} \\
(L, HL, L, HR) & \rightarrow & \text{L} \\
(L, HR) & \rightarrow & \text{L} \\
(L, HR, L, HL) & \rightarrow & \text{L} \\
(L, HR, L, HR) & \rightarrow & \text{OL}
\end{array}
$$

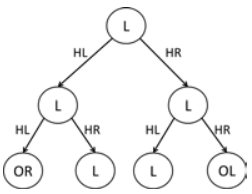


Figure 1: Two equivalent representations of an example policy for the single-agent tiger problem.

2.2 Decentralized POMDPs

The Decentralized POMDP (Dec-POMDP) formalism extends the POMDP formalism to accommodate multiple agents. We can define a Dec-POMDP as a tuple $\langle n, S, A, P, R, \Omega, O \rangle$, where:

- n is the number of agents playing the game.
- S is a finite set of (unobservable) environment states.
- $A = \times_i A_i$ is a set of joint actions, where A_i is the set of individual actions that agent i can perform.
- $P(s'|s,\vec{a})$ gives the probability of transitioning to state $s' \in S$ when joint action $\vec{a} \in A$ is taken in state $s \in S$.
- $R(s,\vec{a})$ gives the immediate reward the agents receive upon executing action $\vec{a} \in A$ in state $s \in S$.
- $\Omega = \times_i \Omega_i$ is the set of joint observations, where Ω_i is the finite set of individual observations that agent i can receive from the environment.
- $O(\vec{\omega}|s',\vec{a})$ gives the probability of the agents jointly observing $\vec{\omega} \in \Omega$ if the current state is $s' \in S$ and the previous joint action was $\vec{a} \in A$.

In Dec-POMDPs, it is generally assumed that agents cannot communicate their observations and actions to each other. These constraints are often present in real world scenarios, where communication may be expensive or unreliable. Consider, for instance, a scenario in which a team of robots must coordinate to search a disaster area for survivors. In such a task, robots may need to spread out to efficiently cover the area and also may need to travel deep underneath rubble, both of which could interfere with wireless communication.

Assuming agents cannot communicate observations and actions, each agent must choose actions based only upon his own actions and observations; however, since the transition, reward, and observation functions depend on *joint* actions, the quality of each agent's policy is dependent on the policies played by all agents or the *joint policy*. The goal of the Dec-POMDP problem, then, is to find the joint policy that maximizes the expected reward received by the agents. The problem of finding this optimal joint policy has been proven to be NEXP-complete [3].

3. Q-LEARNING FOR DEC-POMDPS

While many Dec-POMDP solution techniques have been devised, most of the prevalent Dec-POMDP solution techniques have been centralized and have required knowledge of the model. That is, a central solver determines which joint policy the agents should follow given P,R, and Ω, and then it distributes these policies to the agents. However, in real-world scenarios, the model may not be available, and it may not be feasible for the problem to be solved centrally. There is a need then, for a method by which agents can learn a Dec-POMDP policy in a distributed manner without prior knowledge of the model.

Reinforcement learning algorithms have previously been applied in infinite horizon POMDPs in both model-based [4, 9, 13] and model-free ways [10]. Model-based methods first learn a model of the environment, and then compute a policy based on that model, where model-free methods learn

a policy directly. In Dec-POMDPs, the actions and observations of the other agents are hidden, which makes model-based learning complex; therefore, we opt for the model-free approach. Recently, Zhang and Lesser [17] have applied reinforcement learning to a variant of the finite horizon Dec-POMDP problem, where agents are organized in a network, and agents' influences on one another are limited to cliques. While our goal is similar, we focus on less structured Dec-POMDPs that are inherently less-scalable.

Reinforcement learning algorithms for finite horizon often evaluate an action-quality value function Q, given by

$$Q(s,a,t) = R(s,a) + \gamma \max_{\pi} \sum_{s'} P(s'|s,a) V^{\pi}(s', t+1) \quad (1)$$

where $\gamma \in [0,1]$ is a discount factor, $R(s,a)$ is the reward the agent receives for executing a in state S, and $P(s'|s,a)$ is the probability of transitioning from state s to state s' if action a is executed. $V^{\pi}(s', t+1)$ gives the expected sum of rewards received when executing policy π starting at step $t+1$ given that the state is s'.

Since state transitions are dependent on joint actions in Dec-POMDPs, the equivalent Q function would be $Q(s, \vec{a}, t)$, i.e. based upon joint actions; however, since each agent can only know his own actions and observations, we use an *independent learning* [5] approach where each agent is responsible for learning individual action quality values. Since the states are not visible to the agents, we use a Q function based upon the policy representation of Dec-POMDPs, i.e. each agent maps his individual action-observation histories to actions [17].

Agents can learn these Q-Values concurrently; however, this can lead to oscillation, and our empirical results show that allowing the agents to alternate learning can produce better policies. To elaborate, in alternate learning, the agents take turns learning, and while an agent is learning, all other agents play static policies. Nair et. al have previously proposed an algorithm named JESP [11] in which agents alternately *compute* best responses to each other's policies using knowledge of the model; however, as we do not assume knowledge of the model, we instead *learn* the best-response policies.

For the sake of clarity, we will assume only two agents in notation. Given the policy of the other agent π_{-}, the quality of learner i's action at a given level t individual action-observation history h_t is then given by

$$Q^*(h_t, a|\pi_{-}) = R^*(h_t, a|\pi_{-}) + \sum_{\omega \in \Omega_i} H^*(h_t, a, h_{t+1}|\pi_{-}) \cdot \max_{a' \in A_i} Q^*(h^*_{t+1}, a') \quad (2)$$

where h_{t+1} is a level-$t+1$ action-observation history produced by the concatenation of h_t and (a, ω), i.e. $h_{t+1} = (h_t, a, \omega)$. The best response policy of the learner, π_l, to the other agents' policy is given by

$$\pi_l(h_t) = \arg\max_{a \in A_i} Q^*(h_t, a|\pi_{-}) \quad (3)$$

The function R^* and H^* represent the expected immediate reward and history transition functions for the learner, given by

$$R^*(h_t, a|\pi_{-}) = \sum_{s, h_t} P(s|h_t, h_{-}) P(h_{-}|h_t, \pi_{-}) R(s,a) \quad (4)$$

$$H^*(h_t, a, h_{t+1}|\pi_{-}) = \sum_{s, s', h_{-}} P(s|h_t, h_{-}) P(h_{-}|h^t, \pi_{-}) \cdot \sum_{\omega_{-} \in \Omega_{-}} P(\vec{\omega}|s', \vec{a}) \quad (5)$$

where h_{-} is the history of action-observations encountered by the other agent, and $\vec{\omega} = \langle \omega, \omega_{-} \rangle$ and $\vec{a} = \langle a, a_{-} \rangle$ are the joint observation and joint action, respectively.

Since the agents cannot communicate and do not have prior knowledge of the model, a learning agent will not have knowledge of π_{-}, h_{-}, and a_{-} and must estimate R^* and H^* (and therefore Q^*) from its own experience of executing actions and receiving observations and rewards. We denote the estimation of Q^* as Q. An agent can learn $Q(h_t, a)$ by applying the following update each time it executes a when the current history is h_t

$$Q(h_t, a) = Q(h_t, a) + \alpha \left[r + \gamma \max_{a'} Q(h_{t+1}, a') - Q(h_t, a) \right] \quad (6)$$

where ω and r are the observation and reward received, respectively, after executing a, and $h_{t+1} = (h_t, a, \omega)$.

In our work, we compare both alternating and concurrent Q-learning methods, both of which use the Q update method in equation 6. We refer to these as Q-Alt and Q-Conc, respectively. For both learning methods, each agent follows Algorithm 1 during each episode. In Q-Conc, the *learning* flag is always true for every agent, but in Q-Alt, the *learning* flag can only be true for one agent at a time.

Because the observation and reward functions in Dec-POMDPs depend on joint actions, step 4 of algorithm 1 represents a synchronization point because an agent cannot receive r or ω until all agents have executed an action. This does not, however, imply that agents must execute actions simultaneously.

Algorithm 1 ExecuteEpisode(*learning*)

```
1: h_1 ← ∅
2: for t = 1 to T do
3:    a ← SelectAction(h_t, learning)
4:    Execute a and receive r, ω.
5:    h_{t+1} ← (h_t, a, ω)
6:    if learning then
7:       Q(h_t, a) ← Q(h_t, a) +
8:             α [r + γ max_{a'} Q(h_{t+1}, a') − Q(h_t, a)]
9:    end if
10: end for
```

Algorithm 2 SelectAction(h_t, *learning*) for agent i

```
1: if learning then
2:    if explore then
3:       return explored_action
4:    else
5:       return arg max_{a∈A_i} Q(h_t, a)
6:    end if
7: else
8:    return π(h_t), the current static policy for this agent.
9: end if
```

Algorithm 2 describes the method by which agents select actions to execute. If an agent is currently learning it will

either explore (*explore* and *explored_action* are calculated by a chosen exploration method) or choose an action based upon current Q-Values; however, if an agent is not currently learning - which will only occur in alternate learning - it will choose an action based upon some policy π. Now, if the agent has already learned, π will be the policy it learned previously; however, if the agent has not had a turn learning, π must be some *initial* policy.

There are multiple efficient ways to arbitrarily initialize π. For example, we could let π be a pure policy (i.e. one where each possible action-observation history maps to exactly one action) chosen at random (as is done in the JESP algorithm), or alternatively, we could let π be a uniform stochastic policy, where the agent always chooses an action at random from a uniform distribution. We note, however, that such methods do not consider the behavior of the model. That is, for problems with identical $|A|$ and $|\Omega|$ the policies would initialized the same way, regardless of how P, R, and Ω are defined. In the following section, we will present a principled method for choosing an initial policy that respects both P and R (but not Ω).

The main motivation for selecting initial policies in a more informed manner is the hope of getting in the zone of attraction of the optimal Nash equilibrium. Our approach of alternating best response learning can be essentially looked upon as a form of hill climbing where each step is taken by a different agent. Since hill climbing can only be guaranteed to converge to a local optimum, we expect alternating best response learning to eventually lead to a *locally optimal* Nash equilibrium policy. In order to overcome locality of the optimum, hill climbing approaches often resort to multiple (re)starts, which is unrealistic in our case since each hill climbing step is an entire best response learning phase with a non-trivial cost. Instead, we devote some effort in selecting the initial location of the agents in the joint policy space, hoping that this sets us close enough to the *globally optimal* Nash equilibrium so that it becomes the attractor to the alternating hill climbing trajectory.

4. INFORMED INITIAL POLICIES

In this section, we present a centralized, model-based method to compute an initial policy which is informed by the model. The high-level idea is to map a Dec-POMDP problem into a constrained POMDP problem, solve that POMDP problem, and then map the resulting optimal POMDP policy back into the space of Dec-POMDP joint policies.

To ground the intuition on existing literature, consider the Q_{POMDP} heuristic used in [16], that finds an upper-bound for a Dec-POMDP by mapping the problem into a POMDP where the action set is the set of all joint actions and the observation set is the set of all joint observations.

The Q_{POMDP} relaxation can be formalized as follows. Given a Dec-POMDP instance $D = \langle n, S, A, P, R, \Omega, O\rangle$, create and find the optimal policy value for a POMDP instance $D' = \langle S, A, P, R, \Omega, O\rangle$.

Essentially, Q_{POMDP} assumes the agents are allowed to share their individual observations, which, in turn, eliminates miscoordination because the agents know exactly what their teammates have observed and executed and can therefore choose the best action for each contingency. This is not the case in Dec-POMDP, where each agent must choose the best action based only upon its own action-observation history. In a policy found by Q_{POMDP}, an agent may have different actions associated with the same action-observation history, disambiguated by others' action-observation histories; however, agents do not have access to the observations of other agents during Dec-POMDP execution. Thus, the resulting POMDP policy cannot be (and was not intended to be) mapped to a Dec-POMDP policy in a meaningful way.

In this backdrop, we note that if there were only one possible observation that each agent could receive at each step, this ambiguity would disappear because the agents would be able to know the joint observation history at all times.

Unfortunately, most (if not all) Dec-POMDPs of interest will have more than one observation; however, if the agent policies *ignore* observations, a similar effect can be achieved.

Our proposed method is simple. Create and solve a POMDP where the action set is the set of all joint actions and the observation set is $\{\delta\}$ (a dummy observation). That is, for a Dec-POMDP $D = \langle n, S, A, P, R, \Omega, O\rangle$, create the POMDP $D' = \langle S, A, P, R, \{\delta\}, O\rangle$. Since δ is the only possible observation, the optimal POMDP policy can be thought of as a chain of joint actions $\pi_{opt} = \vec{a}^1, \vec{a}^2, \ldots, \vec{a}^T$, where T is the horizon. The informed initial Dec-POMDP policy can then be constructed for each agent i by setting all level t action nodes to a_i^t (agent i's contribution to a^t). That is, agent i will always execute a_i^t at level t, regardless of what it has previously observed. Figure 2 depicts this process for $T = 2$.

5. LEARNING INFORMED INITIAL POLICIES

In the previous section, we presented a centralized, model-based method for calculating an informed initial policy; however, if we are to use such initial policies for model-free, distributed algorithms (such as Q-Alt), the method by which we calculate these policies should be model-free and distributed as well.

It is generally assumed in Dec-POMDPs that the agents cannot observe each other's actions, so without extra assumptions, the agents must learn independently. One simple approach would be concurrent Q-learning, where each agent i tries to learn

$$Q' : H^{A_i} \times A_i \mapsto \mathbb{R} \tag{7}$$

for all individual action-only histories $h_t^{A_i} \in H^{A_i}$ and individual actions $a \in A_i$. To accomplish this, agents use a modified version of algorithm 1 where the true observation ω is replaced with the dummy observation γ. Since the agents learn concurrently, the *learning* flag will always be true for each agent.

Independent concurrent learning can lead to oscillation because each agent is learning in a dynamic environment (all other agents are treated as part of the environment) that changes in response to the agent's behavior.

Suppose, though, that each agent i can announce the random seed he will use for exploration and his action set A_i before learning begins. This would allow each agent to model the exploration of every other agent. Since observations are ignored when learning the initial policy and since the agents all receive identical rewards, if agents can model each other agent's exploration, then each agent can learn the function

$$Q' : H^A \times A \mapsto \mathbb{R} \tag{8}$$

directly, where A is the set of joint actions and H^A repre-

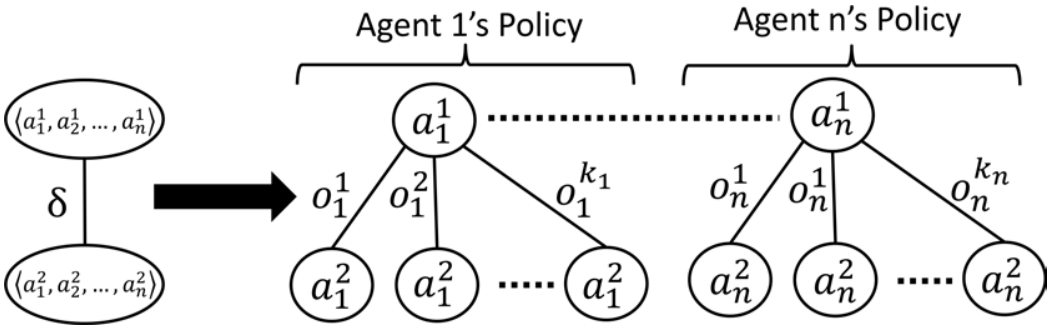


Figure 2: Conversion of the optimal policy of the constrained POMDP to a valid policy of the original Dec-POMDP for T=2.

sents the set of joint action-only histories. Note that such a technique can not be applied to learn $Q : H \times A \mapsto \mathbb{R}$, where H is the set of joint action-observation histories because observations are generated by nature, not the agents, and hence unsharable by announcing random seeds..

The agents can learn equation 8 by making a few modifications to algorithms 1 and 2. Specifically, all actions must be *joint* actions rather than individual actions, and the Q function should accept joint action-observation histories rather than individual histories. As before, ω should be replaced by the dummy observation γ, and the agents will learn concurrently (i.e. the *learning* flag will always be true).

It is important to note that because the set of joint actions A grows exponentially in the number of agents (i.e. $|A| = O(|A_{i^*}|^n)$, where A_{i^*} represents the largest individual action set), the potential memory requirement $(O((|A_{i^*}|^n)^T \cdot |A_{i^*}|))$ for the joint Q'-Value function also grows exponentially in the number of agents. However, for scenarios where this memory requirement is intractable, it is likely that the number of episodes required for convergence of the independent Q'-Value function is also intractably large.

6. INITIAL POLICIES FOR CONCURRENT LEARNERS

In allowing the agents to share random seeds for policy initialization, we are introducing extra information that Q-Conc is unable to exploit. For comparison purposes, we also would like to give Q-Conc access to this information as well. While our learned initial policy is readily usable for alternate learning where all agents except the currently learning agent follow static policies, applying our learned initial policy to concurrent learning where no agent follows a static policy is less straightforward.

In order to use our informed initial policy for concurrent learning, we use the Q'-Values learned as per Section 5 to initialize the agents' Q-Values as follows. When an agent i encounters a history $h_t = (a_1, o_1, a_2, o_2, \ldots, a_t, o_t)$ for the first time, if it learned independent Q'-Values (per equation 7), it will set $Q(h_t, a) = Q'(h_t^{A_i}, a) \forall a \in A_i$, where $h_t^{A_i} = (a_1, a_2, \ldots, a_t)$.

The process is less straightforward if agents learned joint Q'-Values (per equation 8). Recall from section 4 that an informed initial policy can be written as a single chain of joint actions. Suppose h_t^A/h_t is the joint action-only history in which all agents except for i are executing actions per the informed initial policy and agent i is executing $(a_1, a_2, \ldots, a_t)$. Also, suppose $\vec{a}_{t+1}/a$ is the action corresponding to step $t+1$ in the inital policy chain with agent i's action replaced with a. Then, agent i will initialize $Q(h_t, a) = Q'(h_t^A/h_t, \vec{a}_{t+1}/a) \forall a \in A_i$. Essentially, agent i is assuming that the other agents will continue still follow the initial policy, even though h_t or a may deviate from it.

7. EVALUATION

In this section, we evaluate the performance of the five different Q-learning settings: QIP-Alt, QIP-Conc, Q-Conc, QSto-Alt, and QRand-Alt for the BroadcastChannel [6], DecTiger [11], and MarsRover [1] benchmark problems. Note that we do not compare against state-of-the-art model-based Dec-POMDP solvers such as GMAA*-ICE [14] because such solvers rely on full knowledge of the model, which we do not assume is available.

7.1 Problem Domains

In the BroadcastChannel domain, two nodes broadcast messages over a channel that is limited to one message at a time. The nodes receive a reward for each message they successfully send; however, if they attempt to send a message at the same time, a collision occurs. Each node has a buffer that holds at most one message, and at every step this buffer becomes full with some probability.

In the Dec-Tiger domain, there are two doors. Behind one door is treasure, but behind the other door is a tiger. Opening the door concealing the treasure, yields a reward, and opening the door concealing the tiger, yields a penalty. The two agents do not know a priori which door the tiger is behind; however, they are allowed to listen for clues. Upon listening, each agent either hears the tiger behind the left door or the right door. The observations are noisy, however, and each agent has a fifteen percent chance of hearing the tiger behind the wrong door. This serves to make coordination more difficult, as an agent cannot be sure what the other has heard, and miscoordination is heavily penalized.

In the MarsRover domain, two rovers are tasked with conducting scientific experiments at four different test sites, arranged in a four-by-four grid. At each step the agents can choose to move north, south, east, or west, or they can sample or drill. Two of the sites are intended to be sampled, and thus drilling them incurs a large negative reward. Two of the sites must be drilled by both agents simultaneously. Each agent has full knowledge of his own location as well as whether or not an experiment has been performed at each site.

With only two states, two observations, and two actions, BroadcastChannel is the smallest of the three. It is also os-

tensibly the least complex, as it has been solved for a horizon of 900 [14]. DecTiger with two states, two observations, and three actions is larger and has only been solved for a horizon of six. MarsRover with 256 states, eight observations, and six actions is the largest of the three, and it has only been solved for a horizon of four [12].

7.2 Policy Computation and Evaluation

In order to compare the performance of the different settings, we must calculate the value of the policies they produce. One method for calculating a learned policy value is simulation-based, that simply allows the agents to interact with the environment for a sufficient number of episodes and averages the total reward received per episode; however, since the domains we use are known benchmark problems, we can exactly calculate the value of a policy using the model parameters. To evaluate the value of a joint policy $\vec{\pi}$, we find

$$V^{\vec{\pi}}(\vec{h}_0) = \sum_{s \in S} b_0(s) V^{\vec{\pi}}(\vec{h}_0, s) \quad (9)$$

where $b_0 \in \Delta(S)$ is the initial state distribution. $V^{\vec{\pi}}(\vec{h}_t, s)$ for a given joint history $\vec{h}_t$ and state s is given by

$$V^{\vec{\pi}}(\vec{h}_t, s) = R(s, \vec{\pi}(\vec{h}_t)) + \sum_{s' \in S} P(s'|s, \vec{\pi}(\vec{h}_t)) \cdot \sum_{\vec{\omega} \in \Omega} O(\vec{\omega}|s', \vec{\pi}(\vec{h}_t)) V^{\vec{\pi}}((\vec{h}_t, \vec{\pi}(\vec{h}_t), \vec{\omega}_t), s') \quad (10)$$

where $\vec{\pi}(\vec{h}_t) = \langle \pi_1(h_{t,1}), \ldots, \pi_n(h_{t,n}) \rangle$.

7.3 Experimental Results

For each domain, we chose eleven increasing values $k_1, \ldots, k_{11}$ and allowed each setting to learn for an $k = k_1, \ldots, k_{11}$ episodes. The k values were rounded to integers, and derived from experiments with another algorithm (reported in [2]).

k represents the *total* number of episodes agents are allowed to execute, including episodes required to learn initial policies and all alternations. Thus, in Q-Conc and QIP-Conc both agents learn for k episodes (with agents devoting k' of those episodes to learning the initial policy in QIP-Conc), in QSto-Alt and QRand-Alt each agent learns for $\frac{k}{n}$ episodes (where n is the number of agents), in QIP-Alt each agent learns for $k' + \frac{k-k'}{n}$ episodes. Note that since QIP-Alt and QIP-Conc allocate k' episodes for learning the initial policy, they are at a relative disadvantage if the initialization is not helpful because the other settings have k' extra episodes for learning.

We used $k' = 20000$ for DecTiger and $k' = \max(200000, \frac{k}{20})$ for both BroadcastChannel and MarsRover. For the alternating settings in Dectiger and BroadcastChannel, the agents were allowed one alternation each with $\frac{k-k'}{n}$ episodes per alternation ($k' = 0$ for QSto-Alt and QRand-alt), and in the MarsRover problem each agent was allowed we used epsilon-greedy exploration with $\epsilon = .05$. two alternations with $\frac{k-k'}{2n}$ episodes per alternation. In all runs, we used a learning rate $\alpha = .001$, and for our exploration method (lines 2-3 in algorithm 2), we used epsilon-greedy exploration with $\epsilon = .05$ [15]. All trials were run on an Intel Xeon 3.00 Ghz processor.

Tables 1, 2, and 3 give the relative error, $\frac{|v_{learned} - v_{opt}|}{v_{opt}}$ (where v_{opt} is the value of the known optimal policy), averaged over 50 runs for each setting, domain, horizon, and value of k. We have opted to report this data in tables because the range of values is wide some cases (e.g. [0, 16.494]), and such a coarse resolution makes it difficult to interpret relations between smaller values. If an error value reported in the table is bolded, this indicates that it is greater than the error for QIP-Alt for the same horizon and k. In addition to the tables, we report average relative error versus average runtime for QIP-Alt and Q-Conc for all the three domains in figures 3, 4, and 5. Note the the horizontal axes have been scaled logarithmically in these figures.

First note that for each domain and horizon, there exists a k beyond which QIP-Alt produces lower error than all other settings. In the case of BroadcastChannel, the advantage of QIP-Alt is less pronounced; however, since BroadcastChannel is the simplest of our domains, this is not entirely surprising. For MarsRover and DecTiger, the results are stronger, however. QIP-Alt is the only algorithm to have average relative error of zero (implying that all 50 policies found were optimal). In most cases, the nearest competitor in those two domains has error $> .2$ (and often it is much higher). As noted previously, T=4 is the highest horizon for which the MarsRover domain has been solved by a centralized algorithm with knowledge of the model. It is encouraging then that, being distributed and model-free, QIP-Alt is able to achieve near optimal solutions on average for that horizon.

In the case of QIP-Conc, it is unclear whether or not initializing the Q-Values as described in section 6 is beneficial. For Dectiger with $T = 3, k \geq 10^{5.02}$ and BroadcastChannel with $T = 3, k \geq 10^{5.73}$, it is clearly an improvement on Q-Conc, but for other domain-horizon combinations, it is not so clearly beneficial, especially in the MarsRover domain.

While these results suggest that the informed policy is beneficial if optimality is the only goal, they do not speak to the practicality of it. For instance, while QIP-Alt produces its lowest error for T=3 in MarsRover at $k = 10^{7.48}$, it takes on average 60000 seconds to produce that result, whereas $k = 10^{6.42}$ requires only 3000 seconds on average. In real-world scenarios, time may be more important than optimality. To this end, we provide a figures 3, 4, and 5, which show average relative error vs average runtime for QIP-Alt and QIP-Conc for all three domains. Since these are benchmark problems, we have no reason to prefer time over optimality (or vice-versa), so it is difficult to make a qualitative judgement about them; however, had QIP-Alt required days or months to produce error below that produced by Q-Conc, the informed policy would likely be of less practical use.

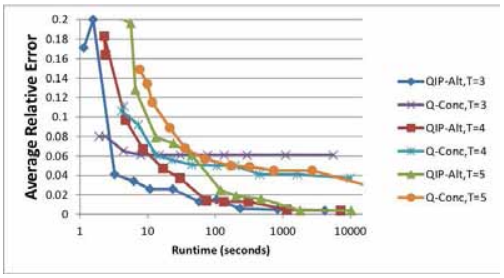


Figure 3: Average relative error vs runtime in seconds for QIP-Alt and QIP-Conc in the BroadcastChannel domain. Horizontal axis has been scaled logarithmically.

k	T=3					T=4					T=5				
	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt
$10^{5.34}$	0.171	**0.175**	0.080	0.085	0.112	0.164	**0.255**	0.111	**0.229**	**0.173**	0.201	**0.330**	0.149	**0.231**	**0.242**
$10^{5.46}$	0.200	0.094	0.080	0.089	0.128	0.183	0.141	0.106	**0.190**	0.166	0.196	0.186	0.134	**0.227**	**0.203**
$10^{5.73}$	0.041	**0.061**	**0.065**	**0.059**	**0.121**	0.097	0.078	0.092	**0.145**	**0.120**	0.128	0.117	0.115	**0.194**	**0.205**
$10^{6.00}$	0.034	**0.053**	**0.061**	**0.049**	**0.112**	0.067	**0.069**	0.061	**0.105**	**0.125**	0.079	**0.102**	**0.089**	**0.165**	**0.184**
$10^{6.27}$	0.026	**0.052**	**0.061**	**0.041**	**0.109**	0.047	**0.067**	**0.056**	**0.065**	**0.122**	0.073	**0.091**	0.068	**0.139**	**0.159**
$10^{6.57}$	0.026	**0.052**	**0.061**	**0.040**	**0.112**	0.037	**0.066**	**0.051**	**0.050**	**0.116**	0.061	**0.087**	0.057	**0.097**	**0.153**
$10^{6.97}$	0.013	**0.036**	**0.061**	**0.033**	**0.107**	0.014	**0.033**	**0.050**	**0.036**	**0.109**	0.025	**0.031**	**0.050**	**0.075**	**0.152**
$10^{7.24}$	0.015	**0.034**	**0.061**	**0.034**	**0.113**	0.013	**0.035**	**0.050**	**0.031**	**0.123**	0.019	**0.036**	**0.049**	**0.065**	**0.143**
$10^{7.60}$	0.006	**0.027**	**0.061**	**0.035**	**0.108**	0.013	**0.039**	**0.041**	**0.028**	**0.105**	0.016	**0.038**	**0.045**	**0.054**	**0.146**
$10^{8.16}$	0.005	**0.026**	**0.061**	**0.032**	**0.114**	0.005	**0.036**	**0.041**	**0.027**	**0.111**	0.004	**0.033**	**0.045**	**0.031**	**0.153**
$10^{8.84}$	0.004	**0.011**	**0.061**	**0.032**	**0.107**	0.004	**0.047**	**0.037**	**0.026**	**0.111**	0.004	**0.023**	**0.030**	**0.023**	**0.154**

Table 1: Average relative errors (compared to the optimal value) for the BroadcastChannel domain for all settings and horizons T=3,4,5. Bolded values are less than respective QIP-Alt values.

k	T=3					T=4					T=5				
	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt
$10^{4.40}$	0.402	**2.058**	**0.734**	**1.437**	**6.407**	1.014	**2.297**	0.726	**15.439**	**13.257**	1.807	**1.885**	1.135	**17.912**	**16.494**
$10^{4.63}$	0.263	**1.425**	**0.617**	**1.612**	**6.194**	0.537	**1.507**	**0.704**	**11.667**	**10.894**	1.221	**1.522**	1.122	**15.293**	**13.940**
$10^{4.81}$	0.084	**0.972**	**0.511**	**1.065**	**6.212**	0.395	**1.071**	**0.692**	**6.158**	**9.511**	1.177	**1.227**	1.089	**16.069**	**12.921**
$10^{5.02}$	0.032	**0.182**	**0.286**	**1.054**	**6.115**	0.197	**0.301**	**0.550**	**2.031**	**8.125**	1.208	0.802	1.034	**12.485**	**9.844**
$10^{5.37}$	0.000	**0.042**	**0.136**	**1.054**	**6.019**	0.000	**0.300**	**0.357**	**1.329**	**7.850**	1.156	0.751	0.941	**3.340**	**6.415**
$10^{5.56}$	0.000	**0.032**	**0.114**	**1.065**	**5.974**	0.000	**0.288**	**0.369**	**1.409**	**7.998**	1.153	0.705	0.816	**2.052**	**6.611**
$10^{5.74}$	0.000	**0.053**	**0.104**	**1.054**	**6.046**	0.005	**0.301**	**0.335**	**1.322**	**8.163**	1.159	0.691	0.754	**1.646**	**5.817**
$10^{6.13}$	0.000	**0.031**	**0.104**	**1.054**	**5.912**	0.005	**0.311**	**0.391**	**1.428**	**7.626**	0.078	**0.641**	**0.648**	**1.215**	**6.035**
$10^{6.42}$	0.011	**0.042**	**0.125**	**1.054**	**6.017**	0.000	**0.299**	**0.334**	**1.321**	**8.266**	0.005	**0.633**	**0.575**	**1.521**	**5.959**
$10^{6.82}$	0.000	**0.042**	**0.125**	**1.054**	**6.026**	0.005	**0.321**	**0.345**	**1.329**	**7.887**	0.030	**0.586**	**0.470**	**1.200**	**5.842**
$10^{7.48}$	0.000	**0.053**	**0.157**	**1.054**	**5.863**	0.000	**0.278**	**0.356**	**1.320**	**7.990**	0.029	**0.275**	**0.359**	**1.201**	**5.821**

Table 2: Average relative errors (compared to the optimal value) for the DecTiger domain for all settings and horizons T=3,4,5. Bolded values are less than respective QIP-Alt values.

k	T=2					T=3					T=4				
	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt
$10^{5.34}$	0.426	**0.862**	0.343	**0.529**	0.408	0.570	**0.682**	0.478	**0.592**	0.509	0.551	**0.672**	0.445	**0.561**	0.525
$10^{5.46}$	0.430	0.407	0.343	**0.476**	0.418	0.589	**0.666**	0.444	**0.601**	0.517	0.537	**0.664**	0.431	**0.586**	0.506
$10^{5.73}$	0.366	**0.381**	0.343	**0.434**	0.298	0.567	0.469	0.411	0.529	0.465	0.517	**0.525**	0.376	**0.537**	0.480
$10^{6.00}$	0.274	**0.381**	**0.343**	**0.415**	**0.291**	0.520	0.433	0.409	0.505	0.461	0.447	0.365	0.340	**0.499**	0.407
$10^{6.27}$	0.274	**0.381**	**0.343**	**0.382**	0.274	0.435	0.429	0.404	**0.461**	0.386	0.425	0.336	0.328	**0.432**	0.354
$10^{6.57}$	0.280	**0.381**	**0.343**	**0.380**	0.274	0.413	**0.427**	0.401	**0.465**	0.378	0.387	0.311	0.308	0.376	0.331
$10^{6.97}$	0.076	**0.381**	**0.343**	**0.377**	**0.288**	0.388	**0.426**	**0.399**	**0.473**	0.374	0.349	0.302	0.287	**0.394**	0.314
$10^{7.24}$	0.000	**0.366**	**0.343**	**0.376**	**0.288**	0.202	**0.427**	**0.398**	**0.451**	**0.369**	0.271	**0.301**	**0.282**	**0.383**	**0.327**
$10^{7.60}$	0.000	**0.381**	**0.343**	**0.375**	**0.301**	0.051	**0.423**	**0.398**	**0.444**	**0.383**	0.095	**0.299**	**0.277**	**0.366**	**0.297**
$10^{8.16}$	0.000	**0.380**	**0.343**	**0.374**	**0.281**	0.057	**0.422**	**0.396**	**0.444**	**0.358**	0.076	**0.302**	**0.272**	**0.363**	**0.292**
$10^{8.84}$	0.000	**0.381**	**0.343**	**0.374**	**0.263**	0.002	**0.421**	**0.393**	**0.436**	**0.342**	0.069	**0.297**	**0.296**	**0.301**	**0.270**

Table 3: Average relative errors (compared to the optimal value) for the MarsRover domain for all settings and horizons T=2,3,4. Bolded values are less than respective QIP-Alt values.

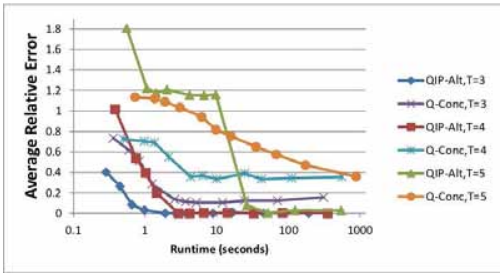


Figure 4: Average relative error vs runtime in seconds for QIP-Alt and QIP-Conc in the DecTiger domain. Horizontal axis has been scaled logarithmically.

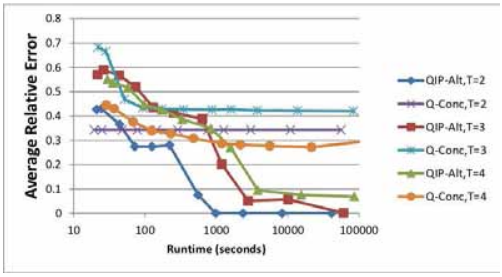


Figure 5: Average relative error vs runtime in seconds for QIP-Alt and QIP-Conc in the MarsRover domain. Horizontal axis has been scaled logarithmically.

8. CONCLUSION

We have presented a simple, principled technique to compute a valid Dec-POMDP policy for use as an initial policy in conjunction with reinforcement learning. Furthermore, we have discussed how such policies can be learned in a model-free manner, and we have shown for three benchmark problems that using such policies as initial policies (instead of stochastic policies or pure policies chosen at random) can improve the outcome of alternating Q-learning.

Acknowledgment: We thank the anonymous reviewers for helpful feedback. This work was supported in part by the US Army under grant #W911NF-11-1-0124.

9. REFERENCES

[1] C. Amato and S. Zilberstein. Achieving goals in decentralized POMDPs. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 593–600, Budapest, Hungary, 2009.

[2] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized pomdps. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, Toronto, Canada, July 2012. To appear.

[3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[4] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Articial Intelligence*, pages 183–188, San Jose, CA, 1992. AAAI Press.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[6] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artifical Intelligence*, pages 709–715, San Jose, CA, 2004.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artifical Intelligence*, 101:99–134, 1998.

[8] L. Kraemer and B. Banerjee. Informed initial policies for learning in dec-pomdps. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence Student Abstract and Poster Program*, Toronto, Canada, July 2012. To appear.

[9] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1995.

[10] N. Meuleau, L. Peshkin, K. Kim, and L. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proc. UAI*, pages 427–436, 1999.

[11] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, Acapulco, Mexico, 2003.

[12] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 577–584, Budapest, Hungary, 2009.

[13] G. Shani, R. Brafman, and S. Shimony. Model-based online learning of POMDPs. In *Proceedings of the European Conference on Machine Learning (ECML)*, volume Lecture Notes in Computer Science 3720, pages 353–364. Springer, 2005.

[14] M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2027–2032, Barcelona, Spain, 2011.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, Mar. 1998.

[16] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First*

Conference on Uncertainty in Artificial Intelligence, pages 576–583, Edinburgh, Scotland, 2005.

[17] C. Zhang and V. Lesser. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In *Proc. AAAI-11*, San Francisco, CA, 2011.

Fast Learning against Adaptive Adversarial Opponents

Mohamed Elidrisi, Nicholas Johnson, and Maria Gini
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
{elidrisi,njohnson,gini}@cs.umn.edu

ABSTRACT

The ability to learn and adapt when playing against an adaptive opponent requires the ability to predict the opponent's behavior. Capturing any changes in the opponent's behavior during a sequence of plays is critical to achieve positive outcomes in such an environment.

We identify two new requirements that we suggest are essential for agents that learn in adaptive environments. These requirements are dictated by the fact that repeated interactions in practice have to be limited and that the opponent can rapidly change strategy through the sequence of interactions. We believe that building intelligent agents that can survive in environments with such requirements will lead to wider deployment of learning agents.

We propose a novel algorithm that is able to learn and adapt rapidly to an opponent even when the number of interactions is limited and the opponent is adapting quickly by changing its strategy. The context we use for the experimental work is two player normal form games. We compare the performance of an agent using our algorithm against agents using existing multiagent learning algorithms.

1. INTRODUCTION

Learning in the presence of opponents is challenging not only because the opponent's behavior is unknown, but also because the opponent can change behavior and adapt to other players. We assume the opponent is adversarial in nature and that the learning agent does not know what strategy the opponent plays. Modeling the behavior of the opponent is a key to success. Moreover, capturing any changes in the opponent's behavior could help avoid issues of deception and exploitation by the opponent.

Multiagent learning has emerged as an active area of research in the past decade with algorithms that can function in adaptive environments as well as theoretical properties for new algorithms [15]. However, the deployment of such algorithms to real world application remains limited due to some unrealistic constraints. In this work we identify two constraints that have appeared in previous work [10, 17] that we believe could potentially increase the deployment of multiagent learning algorithms.

One of the major constraints initially set for agents that play against an opponent in a competitive environment is the assumption that the opponent is either stationary or will converge to a stationary policy [4]. New criteria that, to some degree, relax the stationarity assumption have been proposed in [15]. However, there are still some critical assumptions that we believe are obstacles for the use of learning agents in real world domains.

The first obstacle relates to the need for extremely long sequences of interactions between the agents, often in the order of hundreds of thousands, before the agent learns how to play against the opponent. The second relates to the fact that abrupt changes in the opponent's policy often require restarting the learning process. The second issue has been analyzed from different view points in terms of data bias and computing best-response in games [12].

We proposed an algorithm for fast learning (FAL) against an opponent that deals with those constraints [10]. In this paper we briefly describe FAL and introduce a more refined algorithm, called *FAL-Ensemble*, that is capable of detecting changes in the opponent's behavior faster than the original FAL. We show experimentally how the algorithm performs compared to other commonly used multiagent learning algorithms in scenarios that have the following properties:

1. **Limited repeated interactions.** We consider cases where agents have a limited number of repeated interactions with an adaptive opponent and do not have a long term history to build an opponent model. An intelligent agent deployed in the world is typically faced with a task and a limitation in its possible repeated interactions with its opponent. This is often true for the general case of any opponent but is even more relevant when the opponent is a human. Repeated interactions beyond a few hundred games are a luxury in the world and a rare commodity when an actual human is at the other end. Current literature typically shows empirical results in thousands of interactions even for simple normal form games. While convergence to equilibrium in the long term is very important theoretically, it doesn't address the issue of how to deal with a limited number of interactions.
2. **Rapidly adaptive opponents.** An abrupt policy change requires an agent to be able to learn fast so it can adapt to the changes in the opponent's play. There are numerous reasons for an opponent to change its policy such as the ability to deceive its opponents. The opponent could initially adopt a suboptimal strategy to confuse the learning agent and then switch to another strategy that could lead to more gains.

2. RELATED WORK

Multiagent learning has been an active area of research in the past decade, with the emergence of many criteria and theoretical foundations for developing new algorithms. Within the AI community, the problem has been addressed mainly from three different approaches, as elaborated in the extensive survey by Busoniu et al. [6].

The first approach is focused on adapting single agent reinforcement algorithms for the multiagent setting, as in Littman [13] and other multiagent reinforcement learning methods, but incorporating the adversarial nature of the environment. The second approach combines policy search methods with knowledge of the adversarial nature, as in the work of Bowling [4]. The third approach starts initially from a game theoretic perspective and attempts to produce an algorithmic framework as in [14, 16] and others.

We describe briefly a few algorithms from the different approaches. Minimax-Q [13] was one of the early algorithms with a clear framework for the multiagent learning problem. Minimax-Q is a modified Q-Learning algorithm that uses minimax instead of max in the action choosing step. The performance of Minimax-Q is demonstrated in a two-player grid-like game of soccer.

Godfather and Bully were introduced in [14] as examples of leader strategies that can influence the best response of follower agents in repeated games. Such strategies were shown to be advantageous in some normal form games. They represent a class of deterministic strategies that act with full knowledge of the game structure and the opponent's previous actions. Bully plays the minimax strategy that obtains the security value, which is the minimum value a player can obtain regardless of what the opponent does. Godfather is a generalization of Tit-For-Tat [14] that offers the opponent the opportunity to choose an action with a reward higher than its security value. If the opponent refuses the offer Godfather punishes it.

The "Win or Learn Fast"(WOLF) [4] principle can be applied in any setting where an algorithm is learning with some learning rate. The idea is that as long as the agent is winning the learning rate is increased by a variable α but when losing another rate $\beta > \alpha$ is used. Empirical results running a policy hill climbing (WOLF-PHC) algorithm show the algorithm will converge to the optimal policy when playing against a stationary opponent. Note that if the learning rate used is 1 then WOLF-PHC is equivalent to Q-Learning.

Bowling and Veloso proposed two criteria for multiagent learning, rationality and convergence. *Rationality* states that if the opponents converge to stationary policies then the learning algorithm will converge to a best response. *Convergence* states that the learner will converge to a stationary policy.

Powers and Shoham [16] extended those criteria to include a larger class of opponents. They parameterize the class of opponents against which they achieve optimal performance. This expands the set of allowable opponents from just opponents with stationary policies to adaptive opponents. However, to guarantee the algorithmic properties, the length of history the opponent can use is limited to only the last k moves. Without this assumption an agent will see the history only once and will be unable to learn. The class of adaptive opponents is restricted to a target class known *a priori* and to learn the agent needs to play a very long training sequence. In the results presented the training sequence is 200K rounds for small matrix games. Their algorithm guarantees the following: (1) *Targeted Optimality*: the agent's payoff approaches best response after a phase of exploration against the target class of opponents. (2) *Optimality*: the agent is Pareto efficient in self-play meaning that it cannot be dominated. (3) *Security* : the payoff against any other opponent approaches the security value.

Adapt When Everybody is Stationary, Otherwise Move to equilibrium (AWESOME) [9] is an algorithm for learning against stationary opponents by observing the opponent's actions. AWESOME precomputes a Nash equilibrium for the game and at each iteration of the game it checks whether the opponent is playing the precomputed Nash equilibrium or is playing a non-stationary strategy. It performs the check by defining an epoch and comparing the distribution of actions in the current epoch to the precomputed distribution of actions for the Nash equilibrium. If it determines that the opponent is not playing the Nash equilibrium it then compares the distribution of actions in the current epoch to the previous epoch to determine whether the opponent is playing some other stationary strategy. If it is then AWESOME plays its best response otherwise it restarts. After each restart AWESOME forgets what it has learned and begins by playing the Nash equilibrium and performing the strategy checks. AWESOME learns the best response against a stationary opponent and in self-play.

ReDValer [2] also works against stationary players and observes the opponent's mixed strategy. ReDValer has an additional feature that guarantees a constant regret. The algorithm NoRa [3] is a follow up on ReDVaLeR which modifies existing no-regret algorithms for the multiagent setting. The proposed no-regret learning algorithm satisfies a modified version of Targeted Optimality and Security. The algorithm provides guarantees against stationary opponents, opponents converging to a stationary policy, and produces an average expected payoff not much worse than what best response to the observed distribution from the opponent would produce. A limitation of no-regret strategies is that they do not capitalize on patterns in the opponent play [16] and do not account for an opponent whose strategy depends on the agent's moves.

Weighted Policy Learner (WPL) [1] is an algorithm that converges to a Nash equilibrium with limited knowledge (i.e. no knowledge of the underlying game or observation of the opponent's action). The algorithm works in two-player two-action games with limited knowledge where other multiagent learning algorithms fail.

Efficient Learning Equilibrium (ELE) [5] handles scenarios with imperfect information games. However, it requires that the learning algorithm itself be in an equilibrium. It computes the surplus the agent would have from playing the Nash equilibrium and shows that deviations from such learning algorithm are irrational in polynomial time.

Learn or Exploit in Adversary Induced Markov decision process (LoE-AIM) [7] is an algorithm that has target optimality against any opponent with bounded memory. The memory bound is important because without memory bound the Markov decision process induced by the adversary joint histories will have an infinite state space. The algorithm is distinctive because in theory it makes no assumption about a target class. However, in the implementation it assumes it knows if the opponent is stationary or not.

3. FAST ADAPTIVE LEARNER (FAL)

We proposed a novel algorithm for an agent which learns a strategy to use when playing repeated games against an adaptive opponent [10]. The key feature of our algorithm is that it is able to learn in a limited number of repeated interactions and is able to detect and adapt to potentially abrupt changes in the opponent's strategy.

The algorithm, at a high level, has two parts: (1) a *Predictive Model* which makes a prediction about the opponent's next action; (2) a *Reasoning Model* which chooses a suitable best response accordingly.

There is a large class of models and methods that can be used for both parts of the algorithm. However, in order to meet the requirements of limited repeated interactions and fast adaptive opponents we need to impose constraints on the models. These are the main requirements on each part:

1. the *Predictive model* makes a prediction about the opponent's next action with the following requirements:
   (a) is online in nature;
   (b) weighs the recent observation more than the past;
   (c) preserves the ordering of the sequence of repeated interactions;
   (d) is sensitive to abrupt changes in data streams.
2. The *Reasoning model* chooses a suitable best response with the following requirements:
   (a) reasons about the temporal accuracy of the predictive model above;
   (b) reasons if the opponent is cooperative or competitive;
   (c) attempts to teach the opponent to cooperate (if the opponent is teachable);
   (d) analyzes its own average reward and whether it is "losing" or "winning" and incorporates it in its decision making process.

We instantiate our *Fast Adaptive Learner (FAL)* algorithm with the models we describe next.

3.1 Predictive Model

For the Predictive Model, we use an algorithm called Entropy Learning Pruned Hypothesis Space (ELPH) [11]. ELPH was developed as an online predictive algorithm for sequences that met the requirements we specified above for the predictive model. ELPH has the ability to learn to predict from a sequence of observations rapidly and therefore can adapt to non-stationary opponents quickly.

At the core of ELPH is the hypothesis space, which is constantly updated with patterns and predictions using the recent history of events and predictions. Every time an observation matches one or more patterns, ELPH computes the entropy of the matching set and uses the entropy to prune the rules that have high entropy. The rule with the lowest entropy is then used to make the prediction. Pruning facilitates adaptation because it removes unsuccessful rules from the hypothesis space. Algorithm 1 provides a sketch of how ELPH works, more details are in [11].

3.2 Reasoning Model

For the Reasoning Model, we adapt a solution which is a compromise of two strategies: (1) a Godfather like strategy, called *Godfather-Future*, which takes as input the predicted future opponent action; (2) a model, called *Meta-Prediction*, which reasons about the success of the predictive model in predicting the opponent's next action.

Algorithm 1 ELPH: Entropy Learning Pruned Hypothesis Space

```
λ: is the threshold for hypothesis pruning.
{a_1, a_2, ..., a_k} is the history of k actions.
create all possible subsets of the history {a_1, a_2, ..., a_k}
compute the reliable entropy for each subset
prune subsets with entropy greater than λ
make a prediction based on the lowest entropy hypothesis.
```

The Godfather-Future strategy computes a targetable pair of actions, i.e. any pair of deterministic strategies with the property that it yields a reward for the players higher than their security value.

While the original Godfather plays its half of the targetable pair if the opponent played its half in the last interaction, Godfather-Future plays its half of the targetable pair if the opponent is predicted to play its half in the next interaction.

The Meta-Prediction computes a weighted average prediction success on a moving horizon. Let's assume that $P_i \in \{0, 1\}$ is the prediction of the predictive model in interaction i, $P_i = 1$ if it predicted successfully and 0 otherwise. The weighted W_t average prediction success at time t is

$$W_t = \frac{(t)P_t + (t-1)P_{t-1} + \ldots + (t-k)P_{t-k}}{t + (t-1) + \ldots + (t-k)}$$

where k is a constant.

Meta-Prediction compares the value of W_t to a parameter β. If $W_t > \beta$ then Meta-Prediction will consider the ELPH prediction reliable, otherwise it replaces the prediction with the opponent's last action. If β is set to 1 then the model will behave similarly to the original Godfather; as β gets closer to zero, the ELPH predictions are used more regardless of their accuracy. How to choose an appropriate value for β is discussed later in the experimental work.

Algorithm 2 is a high-level description of how FAL works.

Algorithm 2 FAL Algorithm

```
Teach_min: is the teaching period
β: is the threshold of confidence in the prediction
while i ≤ end of game do
   if W_t ≤ β then
      Predicted Next Action ← Last Action
   else
      Predicted Next Action ← ELPH Predicted Action
   end if
   if i ≤ Teach_min then
      Play half of Targetable Pair
   else
      Next Action ← Godfather-Future Predicted Next Action
      Compute Average Prediction Success
   end if
   Increment i
end while
```

4. ENSEMBLE OF FAST ADAPTIVE LEARNERS (FAL-ENSEMBLE)

The FAL-Ensemble algorithm (shown as Algorithm 3) functions in a way similar to the original FAL algorithm with a significant difference. It monitors the number of new hypotheses generated in its ELPH predictive model. If at time t the number of new hypotheses reaches a threshold of δ it creates a new predictive model that starts the observations from time t going forward. This process is repeated, every time the number of new hypotheses is above the threshold, a new model is added. When making a prediction FAL-Ensemble takes a majority vote among the ensemble of predictive models weighted by their past accuracy.

Algorithm 3 FAL-Ensemble Algorithm

```
δ: is the threshold for starting a new model.
M_1, ... M_j: is the set of ELPH models.
M_k = {ELPH(0)}
while i ≤ end of game do
  if i ≤ Teach_min then
    Next Action ← Play half of Targetable Pair.
  else
    Next Action ← Weighted Majority Vote from M_k
  end if
  Added Hypotheses ← Sum of added hypotheses in M_k
  if Added Hypotheses ≥ δ then
    M_k ← M_k ⋃ ELPH(i)
  end if
  Add New Observation to all models in M_k
  Increment i

end while
```

5. EXPERIMENTAL RESULTS

We compared experimentally the performance of different learning algorithms, using two-player repeated normal form games as the setting for our experiments. An example of a game matrix is in Table 1, where the row player's payoff is given first followed by the column player's payoff. At each interaction the players play a simultaneous move. After each interaction, each player receives a payoff which depends on the joint action chosen for the interaction.

Each of the normal form games we have chosen was played for 100 iterations. We repeated each of the 100 iterations 100 times to compute average outcomes and reduce the noise. We choose to limit the number of repeated games to 100 to support the core idea of having limited repeated interactions. We also tried the different algorithms against an adversary that at some point switches strategy. This enabled us to explore how the agents respond to situations where the opponent does not converge to a stationary policy.

We have chosen two set of algorithms for our experimental work. Bully and Godfather are representatives of deterministic methods. Q-Learning, WOLF-PHC, and AWESOME represent learning methods. Each one of the learning methods also represents one of the approaches of multiagent learning discussed earlier in section 2 and in [6]. All methods assume knowledge of the underlying game and observe the opponent's actions.

Q-Learning is a classical learning method, WOLF-PHC is one the early algorithms that combine policy search and opponent modeling, AWESOME learns best response. The specific algorithms and strategies we use are:

- Q-Learning is a general purpose semi-supervised learning algorithm that we adapted for the repeated game setting. Q-Learning traditionally assumes a stationary, fully observable environment. In this framework we break these assumption [14].
- WOLF-PHC operates on a different assumption from Q-Learning with the specific notion of deciding the best action depending on whether the agent is losing and winning. WOLF-PHC updates the belief on the current actions with a steady rate as long as it leads to wins. When the action leads to losses, WOLF-PHC changes that belief on the actions with a higher rate in order to find another suitable action. WOLF-PHC converges to the optimal policy under the assumption that the opponent is a learner that will converge to a stationary policy.
- FAL and FAL-Ensemble are our proposed algorithms.
- Godfather is a general class of Tit-For-Tat strategies with the basic idea of giving the opponent the chance to "cooperate" by playing a targetable pair that gains the opponent more than the security value. However if the opponent fails to play the targetable pair, Godfather will punish it by playing the minimax strategy.
- Bully is a deterministic policy that chooses the action the maximizes the player's payoff assuming the opponent will best respond. A more detailed description of Bully is in [14] .
- AWESOME is an adaptive algorithm that learns to play against opponents that (eventually) play a stationary strategy. It does this by observing its opponent's actions and either plays a precomputed Nash equilibrium strategy or a best response strategy.

The experimental work is divided into two groups of experiments. In the first group we test different agents, each one using one of the algorithms in the set listed above. The agents play in pairwise runs against each other in each game. The second group involves a subset of agents this time playing against a specific agent that switches strategy. The goal of this second group of experiments is to understand how fast different types of agents adapt to an opponent that switches its strategy during the game. For FAL, the β is set at 0.75 unless otherwise stated.

Experiment 1

The tables of results presented in this experiment represent the average reward at the end of 100 iterations for each game. Each entry in the table contains two values. The first represents the reward obtained by the row player and the second represents the reward obtained by the column player when playing against each other. We only show the results of FAL because FAL-Ensemble and FAL were identical due to the fact that in FAL-Ensemble the added hypotheses didn't reach the threshold to create an Ensemble.

Chicken

In the game of Chicken, shown in Table 1, in the most cooperative solution each player can get a payoff of 3.0. However, at each interaction the opponent has an incentive to exploit to receive a reward of 3.5. If both agents deviate they both get a lower outcome of 1.0.

3.0,3.0	1.5,3.5
3.5,1.5	1.0,1.0

Table 1: Chicken game matrix.

	Q1	WF	FAL	GF	Bully	AW
Q1	2.4,2.4	2.4,2.4	1.8,2.6	2.5,2.5	1.3,2.5	2.6,2.5
WF		2.4,2.4	1.8,2.6	2.3,2.3	1.3,2.5	2.7,2.4
FAL			3.0,3.0	3.0,3.0	1.0,1.0	2.9,2.9
GF				3.0,3.0	1.0,1.0	3.5,1.5
Bully					1.0,1.0	3.5,1.5
AW						2.5,2.5

Table 2: Average pairwise payoffs after 100 repeated games of Chicken.

Analysis. The reward obtained by Bully is 1.0, which is the lowest reward possible. This happens because Bully assumes the opponent will best respond and so it will never try to cooperate. It is evident that in this game cooperation would yield a higher payoff of 3.0 for both players. Agents that are willing to cooperate, like FAL and Godfather, or are willing to understand when there is potential of higher reward, like Q-learning and WOLF-PHC, end up choosing a set of actions that lead to higher reward, as shown in Table 2. However, an agent like AWESOME doesn't attempt to cooperate and plays the Nash equilibrium strategy to start and it continues to use it against Bully, Godfather, and in self-player. Against WOLF-PHC and Q-Learning, AWESOME switches to a best response strategy when these agents are exploring or cooperating (in the case of FAL). Due to this, AWESOME receives a lower reward then when playing against other agents.

It is important to notice the speed at which the agents were able to get the higher cooperative outcome of 3.0. Let's look at the row of FAL. FAL learns and signals to the opponent that it is willing to cooperate. Q-Learning, while better than Bully, is slow at converging to cooperation and gets only a payoff of 1.8 after 100 games.

Unfortunately, WOLF-PHC and Q-Learning are learners but are relatively slow in learning the cooperative nature of the opponent. The slowness in their learning is evident when compared to the speed of FAL in self-play or even the original Godfather, which is a stationary strategy. Playing against Bully is the most clear example of how the lack of fast learning implies higher reward for the opponent. Bully is a deterministic policy which does not cooperate, but Q-Learning and WOLF-PHC are unable to realize this fast enough and continue to explore in the hope of finding a better outcome. FAL with its fast prediction of the opponent's actions is able to realize this and to revert back to the security value.

Prisoner's Dilemma

In Prisoner's Dilemma, shown in Table 3, the dominant strategy is to defect and receive a reward of 1.0. Cooperating would lead to a higher outcome of 3.0 but with the added risk of getting 0 if the opponent decided to betray.

Analysis. The issue of speed of learning remains a challenge in this game but there are some interesting differences between the nature of the games Chicken and Prisoner's Dilemma that lead to WOLF-PHC and Q-Learning to learn

3.0,3.0	0.0,5.0
5.0,0.0	1.0,1.0

Table 3: Prisoner's Dilemma game matrix.

	Q1	WF	FAL	GF	Bully	AW
Q1	1.7,1.7	1.7,1.7	2.2,2.2	2.4,2.4	0.9,1.4	1.8,1.1
WF		1.9,1.9	2.2,2.2	2.4,2.4	0.9,1.4	2.0,1.1
FAL			3.0,3.0	3.0,3.0	1.0,1.0	1.9,2.0
GF				3.0,3.0	1.0,1.0	1.0,1.0
Bully					1.0,1.0	1.0,1.0
AW						1.0,1.0

Table 4: Average pairwise payoffs after 100 repeated games of Prisoner's Dilemma.

β	WOLF-PHC	FAL
1	2.4	2.4
0.75	2.2	2.2
0.55	2.1	2.1
0.35	2.0	2.0

Table 5: The effect of different β choices for FAL on WOLF-PHC vs FAL in Prisoner's Dilemma.

relatively faster than Bully. In Chicken the difference was between getting a reward of 1.0 versus 1.5 while in Prisoner Dilemma the difference is either 1 or 0 against Bully.

In a side sub-experiment reported in Table 5 we investigated the effect of using different values of β on the performance of our algorithm. This sub-experiment was conducted in Prisoner's Dilemma because it was the game with most significant changes in rewards.

We assume that the benefit occurs because FAL, by trusting its prediction, gives a longer grace period before it punishes the opponent which gives more time to the opponent to understand that FAL is willing to cooperate. However, this result depends on the exploration policy. The grace period might not be sufficient for WOLF-PHC to reach the cooperation stage.

Q-Learning against a simple strategy such as Bully fails to learn that Bully is stationary quickly enough and continues its attempt to explore. This leads to more than 10% loss in reward than the security value. This problem is not apparent in others agents that are adapting faster.

Note that it is evident that Q-Learning in its basic setting has no concept of winning and losing and it is simply learning parameters based on the collected reward. On the other hand, WOLF-PHC has a clear concept of winning and losing and that affects its parameter learning. WOLF-PHC is able to learn at a faster rate than Q-Learning and obtains an average reward of 1.9 in self-play while Q-Learning gets 1.7 in self-play.

AWESOME is able to receive the Nash equilibrium reward against Bully, Godfather, and in self-play and is able to receive a higher reward when playing against FAL, WOLF-PHC, and Q-Learning. It is able to receive a higher reward against these agents because it can take advantage of the initial exploration period of Q-Learning and WOLF-PHC to earn the highest possible reward of 5.0 until Q-Learning and WOLF-PHC eventually converge. Against FAL it is able to receive an even higher reward because FAL initially

3.0,3.0	2.0,0.0
0.0,2.0	1.0,1.0

Table 6: Deadlock game matrix.

	Q1	WF	FAL	GF	Bully	AW
Q1	2.7,2.7	2.7,2.7	2.8,2.8	3.0,3.0	3.0,3.0	2.1,2.1
WF		2.7,2.7	2.8,2.8	3.0,3.0	3.0,3.0	1.8,1.9
FAL			3.0,3.0	3.0,3.0	3.0,3.0	3.0,3.0
GF				3.0,3.0	3.0,3.0	3.0,3.0
Bully					3.0,3.0	3.0,3.0
AW						3.0,3.0

Table 7: Average pairwise payoffs after 100 repeated games of Deadlock.

cooperates and AWESOME is able to earn a reward of 5.0 many times until FAL learns it is doing this and switches to play the Nash equilibrium strategy.

Deadlock

In the game Deadlock, the choice of action is obvious, which is to choose action 1, i.e. cooperating. This leads to a reward of 3.0 for both sides.

Analysis. Deadlock is a simple game where the choices are clear. Most agents choose the cooperative action. The exception is Q-Learning and WOLF-PHC which get rewards slightly below 3.0 due to their exploration, especially when playing against other learning agents. The question remains what is the value of exploration in this game. This is a game where any exploration could be viewed as an irrational choice, because there is no incentive to get a higher reward from the one given. Godfather and Bully are deterministic strategies so they do not change their choice. FAL and AWESOME converge to a cooperative state and do not explore a suboptimal solution while in self-play. WOLF-PHC and Q-Learning continue to explore in self-play and against other learners.

Battle of the Sexes

In the game of Battle of the Sexes both players want to coordinate but they have conflicting interests. There are two pure Nash equilibria where both players choose the same action and one mixed Nash equilibria where both players play their preferred action more often than the opponents preferred action.

3.0,2.0	0.0,0.0
0.0,0.0	2.0,3.0

Table 8: Battle of the Sexes game matrix.

	Q1	WF	FAL	GF	Bully	AW
Q1	2.7,2.1	2.6,2.2	2.9,1.9	3.0,2.0	3.0,2.0	1.5,1.1
WF		2.7,2.0	2.9,2.0	2.9,2.1	2.9,2.0	1.5,1.2
FAL			2.0,3.0	2.0,3.0	2.0,3.0	2.0,3.0
GF				2.0,3.0	2.0,3.0	2.0,3.0
Bully					2.0,3.0	2.0,3.0
AW						2.0,3.0

Table 9: Average pairwise payoffs after 100 repeated games of Battle of the Sexes.

Analysis. In Battle of the Sexes, Godfather and Bully simply play one of the cooperative actions. FAL starts by playing one of the cooperative actions and learns what the opponent strategy is. However, it does not switch strategies because it will not gain any reward by doing so. In fact, if FAL were to switch strategies it would force both itself and its opponent to lose out on all potential reward. WOLF-PHC and Q-Learning receive slightly below the equilibrium reward due to the exploration period. AWESOME best responds to WOLF-PHC and Q-Learning during this time and is able to obtain more reward than it would if it were to not switch from the Nash equilibrium strategy. However, the average reward AWESOME receives is low because it is slow to recognize that its opponent has switched strategies and AWESOME receives a reward of 0.0 for a few iterations.

Collaboration

In the Collaboration game both players want to coordinate and choose the same action and don't have a preference for which action they both agree on. There are two pure Nash equilibria where both players choose the same action.

3.0,3.0	2.0,2.0
0.0,0.0	3.0,3.0

Table 10: Collaboration game matrix.

	Q1	WF	FAL	GF	Bully	AW
Q1	2.8,2.8	2.7,2.7	2.9,2.9	2.9,2.9	2.9,2.9	2.5,2.5
WF		2.7,2.7	2.9,2.9	2.9,2.9	3.0,3.0	2.4,2.4
FAL			3.0,3.0	3.0,3.0	3.0,3.0	3.0,3.0
GF				3.0,3.0	3.0,3.0	3.0,3.0
Bully					3.0,3.0	3.0,3.0
AW						3.0,3.0

Table 11: Average pairwise payoffs after 100 repeated games of Collaboration.

Analysis. In Collaboration, Godfather, Bully, and FAL play one of the cooperative actions and FAL learns that it should not change actions because it will receive less reward even though it knows what action its opponent will play. WOLF-PHC and Q-Learning again receive slightly below the equilibrium reward, similar to Battle of the Sexes, due to the exploration period. AWESOME best responds but the average reward is higher for this game because AWESOME does not receive a reward of 0.0 as often since it now also may receive a reward of 2.0.

Coordination

In the Coordination game both players want to coordinate and choose the same action and they both prefer action 1 over action 2. There are two pure Nash equilibria where both players choose the same action however action 1 dominates the other.

Analysis. In Coordination, the results are similar to Collaboration however the cases when Q-Learning plays against AWESOME and when WOLF-PHC plays against AWESOME are interesting. Both players receive far lower than the equilibrium reward. This is because Q-Learning and WOLF spend time exploring and as such sometimes play action 2. This will initially give both players a reward of 1.0. AWESOME eventually determines that its opponent is

3.0,3.0	0.0,0.0
1.0,1.0	2.0,2.0

Table 12: Coordination game matrix.

	Q1	WF	FAL	GF	Bully	AW
Q1	2.3,2.3	2.2,2.2	2.4,2.4	2.7,2.7	2.3,2.3	1.3,1.3
WF		2.2,2.2	2.6,2.6	2.7,2.7	2.4,2.4	1.3,1.3
FAL			3.0,3.0	3.0,3.0	3.0,3.0	3.0,3.0
GF				3.0,3.0	3.0,3.0	3.0,3.0
Bully					3.0,3.0	3.0,3.0
AW						3.0,3.0

Table 13: Average pairwise payoffs after 100 repeated games of Coordination.

playing a stationary strategy and plays its best response to this strategy which is to play action 2 instead of its initial Nash equilibrium action 1. This forces both players to receive a reward of 2 and the joint strategy of both players playing action 2 is a Nash equilibrium so both players never deviate from this strategy from then on.

5.1 Experiment 2: Switching Strategy

In Experiment 1 we have shown that FAL is able to learn faster, adapt and achieve better results than Q-Learning, WOLF-PHC, and Bully. FAL was also able to receive better results in self-play in comparison to AWESOME in self-play. However, the performance of Godfather, and FAL are almost identical in many scenarios. Moreover, we would like to further show the power of FAL-Ensemble over the original FAL. In order to show the importance of fast adaptive learning we present what happens against an opponent that changes its strategy after some period of time. Detecting the change and adapting to it is the real advantage that we are aiming at achieving in this work.

We introduce a new agent we call *Switch agent*. The agent starts by following the classical Godfather strategy until it reaches stage 40 of the game. After that, the agent follows a deterministic repeated sequence of actions $\{a_1, a_2, a_1, a_1, a_2, a_1\}$ indefinitely. This agent is intended to be deterministic and predictable with a bounded memory. The choice of making the agent switch to a deterministic policy was made to simplify the analysis. Despite its simplicity, learning to play against this opponent is challenging.

In this experiment, we show the results of the Switch agent playing against Godfather, WOLF-PHC, FAL, and FAL-Ensemble in the game of Chicken. The rest of the games show similar trends as Chicken so the analysis would be similar. WOLF-PHC and Godfather were chosen to represent the learner agents because of their strong performance in Experiment 1. Figure 1 shows the average reward over time for the 4 agents against the Switch agent.

Analysis of Rewards Figure 1 shows the graph for the Switch agent against the four agents Godfather, WOLF-PHC, FAL, and FAL-Ensemble. The figure reports the difference Delta in average reward between the agents and the Switch agent. Positive Delta rewards imply Switch is getting more reward and 0 is a tie. FAL and Godfather were able to detect that the opponent is cooperative and converged to a stable reward. WOLF-PHC attempted to learn the cooperative nature but started by fluctuating and did not actually converge to constant reward as FAL and Godfather.

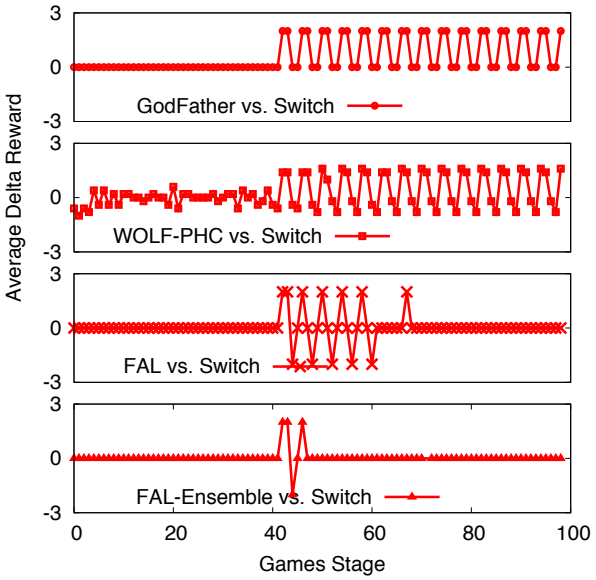


Figure 1: Average delta reward for the 4 Agents vs. Switch agent. Positive values imply the Switch agent is getting more reward, 0 are ties, and the positive values are the others.

At stage 40 of the game, the Switch agent switches to its deterministic strategy. At that stage, the behaviors of the four agents vary. FAL goes on a period of attempting to predict the opponent's action. In this period FAL's prediction accuracy fluctuates which leads into a phase where FAL will use a combination of its prediction or the opponent's previous action to decide its own next action. This continues until FAL succeeds in predicting the Switch agent action sequence which will occur at around stage 60. From that point forward, FAL has a 100% accurate prediction of Switch that will lead to stable Delta rewards. FAL-Ensemble starts a new Predictive model at time 40 and is able to capture the new behaviors without any biases in less than 8 interactions in comparison to 20 in FAL.

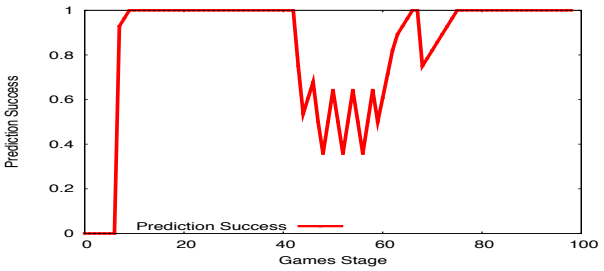


Figure 2: FAL success at predicting the agent's next action.

Figure 2 shows the success of FAL in predicting the action of the Switch agent. The graph shows how FAL after the first 7 interactions is able to determine the next action of its opponent with 100% accuracy. When it reaches stage 40, the prediction success dropped and continued to fluctuate around 40% and 70% but in around 20 interactions it

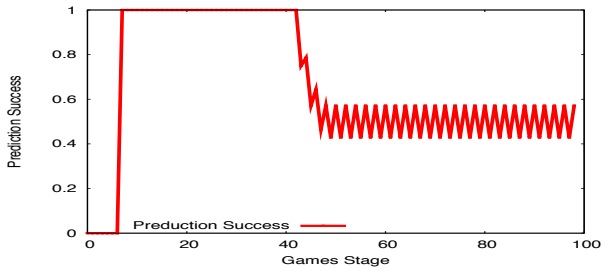


Figure 3: Godfather success at predicting the agent's next action (Godfather prediction is simply the opponent's last action).

went up to almost 100%. This is very significant as it shows that our predictive model is capable of adapting rapidly to changes in the opponents behavior.

Stage 40 and beyond shows a shortcoming of the Godfather strategy. Godfather continues to miscalculate the next action and ends up either getting 1.5 or 3.0, which is driven by the actions of the Switch agent. This issue rises because Godfather makes its next action based on the last action of the opponent while in order to fully capture the opponent a model needs to look at least three past actions as a sequence. Figure 3 shows by using the opponents last action as prediction for its next action Godfather was able to predict the first 40 stages. However, after stage 40 Godfather's prediction is almost random. This also explains our requirement of a predictive model that is able to find and exploit any information hidden in the ordered sequence of actions.

WOLF-PHC struggles in a fashion similar to Godfather after stage 40, by alternating between rewards of 1.5 and above 3.0. The issue with WOLF-PHC that even if WOLF-PHC is able to quickly learn the probability of each action, it will not be a clear sequence predicted as provided by FAL. This will cause WOLF-PHC to have sub-optimal best response to the Switch agent in the limited interactions as shown in the delta rewards. The comparison of these results is important as it shows the true power of FAL in comparison to both WOLF-PHC and Godfather. Moreover, it shows the power of FAL- Ensemble by eliminating bias and add a new predictive model to the Ensemble.

6. CONCLUSIONS AND FUTURE WORK

Our goal in this work was to motivate and introduce the need for new requirements on multiagent learning algorithms. We want to be able to build agents that learn even when interacting with an opponent for a limited number of repeated interactions as well as that have the capability to adapt to sudden and frequent changes in the opponent's strategy regardless of whether the opponent is truly adaptive or even stationary in the limit. We proposed a new algorithm, FAL, and its variation, FAL-Ensemble, which creates a new predictor whenever the opponent strategy switches and which makes predictions by a majority vote among its predictive models. We showed experimentally that FAL and FAL-Ensemble outperform other algorithms in this context. Future work will be directed at examining theoretical properties of FAL, extending it to work with n-player games, making predictions for more than the next opponent's action, and applying it to a larger class of games.

7. REFERENCES

[1] S. Abdallah and V. Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33(1):521–549, 2008.

[2] B. Banerjee and J. Peng. Performance bounded reinforcement learning in strategic interactions. In *Proc. AAAI Conference*, pages 2–7, 2004.

[3] B. Banerjee and J. Peng. Efficient no-regret multiagent learning. In *Proc. AAAI Conference*, volume 20, pages 41–46, 2005.

[4] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[5] R. Brafman and M. Tennenholtz. Optimal efficient learning equilibrium: Imperfect monitoring in symmetric games. In *Proc. AAAI Conference*, page 726. AAAI Press, 2005.

[6] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. on Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

[7] D. Chakraborty and P. Stone. Online multiagent learning against memory bounded adversaries. In *Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases*, pages 211–226. Springer, 2008.

[8] D. Chakraborty and P. Stone. Convergence, targeted optimality and safety in multiagent learning. In *Proc. Int'l Conf. on Machine Learning*, June 2010.

[9] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1):23–43, 2007.

[10] M. Elidrisi and M. Gini. When speed matters in learning against adversarial opponents (Extended Abstract). In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, page To Appear, 2012.

[11] S. Jensen, D. Boley, M. Gini, and P. Schrater. Non-stationary policy learning in 2-player zero sum games. In *Proc. AAAI Conference*, pages 789–794. AAAI Press, 2005.

[12] M. Johanson and M. Bowling. Data biased robust counter strategies. In *Twelfth Int'l Conf. on Artificial Intelligence and Statistics*, pages 264–271, 2009.

[13] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. Int'l Conf. on Machine Learning*, 1994.

[14] M. Littman and P. Stone. Leading best-response strategies in repeated games. In *Int'l Joint Conf. on Artificial Intelligence Workshop on Economic Agents, Models, and Mechanisms*, 2001.

[15] R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *Proc. Int'l Conf. on Artificial Intelligence*. Citeseer, 2005.

[16] R. Powers, Y. Shoham, and T. Vu. A general criterion and an algorithmic framework for learning in multi-agent systems. *Machine Learning*, 67(1):45–76, 2007.

[17] Z. Wang, A. Boularias, K. Mülling, and J. Peters. Balancing safety and exploitability in opponent modeling. In *Proc. AAAI Conference*, 2011.

Adaptive Agents in Dynamic Games : Negotiating in Diverse Societies

Eunkyung Kim, Yu-Han Chang, Rajiv Maheswaran, Yu Ning, Luyan Chi
Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{eunkyung, ychang, maheswar}@isi.edu, {yuning, lchi}@usc.edu

ABSTRACT

We address the problem of creating agents that can negotiate in an environment where adapting to the norms of the society is a significant factor to performing well. We focus on the Social Ultimatum Game, a multi-agent multi-round extension of the Ultimatum Game, a classical game-theoretic problem which has been studied for decades due to the variability of the behavior it elicits. We create ten societies of agents based on five classes of agent behavior to create a diverse test environment. We used Amazon Mechanical Turk to evaluate how human participants adapt to these various societies, serving as a baseline for several agent-based approaches. Current approaches primarily succeed in societies similar to their own and fail elsewhere. We construct a new type of adaptive agent, based on single-step marginal-value optimization, and show that it outperforms humans across these varying agent societies when playing the Social Ultimatum Game.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*

General Terms

Algorithms, Economics, Experimentation

Keywords

Multi-Agent Systems, Game Theory, Ultimatum Game, Mathematical Models of Human Behavior, Learning, Adaptation

1. INTRODUCTION

Creating software agents that can negotiate effectively is an important problem that has been studied by agent researchers in contexts such as the trading agent competition and the virtual agents community. In the former, a goal is typically to find optimal policies in settings with uncertain and incomplete information, and where policies are evaluated in societies of entirely artificial agents [25]. In the latter, a goal is to create agents that can interact with humans — in many cases, to train them in negotiation with individuals from particular cultures or different value settings [22].

In this paper, we examine a problem that combines the complexities of these goals. We want to create negotiating agents that can perform effectively in multiple environments, specifically in a multitude of societies where values and styles of negotiation might be significantly different. For example, certain societies may have a high degree of reciprocity. Others may have extremely high (or low) standards for what is considered a fair trade. Since performance can be highly dependent on the interaction environment, the design of such a negotiation agent is not a straightforward optimization problem. Performance depends on adapting to the norms of that society by learning quickly through interaction.

As context for this investigation, we use the Social Ultimatum Game [6], a multi-agent multi-round extension of the Ultimatum Game. The Ultimatum Game has a three-decade history in social sciences that shows how cultural and other factors significantly affect behavior/performance. It maps to real-world problems because each interaction splits surplus from a joint activity (a common negotiation issue) and thus abstracts many economic transactions. In the classical Ultimatum Game, an endowment can be shared by a proposer and responder, if the responder agrees to the proposer's split of the endowment. Otherwise, neither the proposer or responder receives any reward. Even in this one-shot interaction, it has been shown through many investigations in a wide-range of research communities that humans exhibit a wide range of behaviors that deviate from a "rational" payoff-maximizing strategy based on factors such as cultural background, occupation and emotional factors among others.

In order to perform well, it is important for the proposer to have an accurate model of the responder. The Social Ultimatum Game allows each player to take the role of the proposer, to pick the responder from the set of all players, to decide to accept or reject any offers it receives from the other players, and to repeat this process for multiple rounds. Thus, players have the chance to learn and adapt to the game population. Also, the game adds an alternate method for generating payoffs: enticing the other players to choose oneself as the responder.

To investigate adaptation in this context, we implemented several behavioral and bargaining approaches, drawn from the wide literature in this area: adaptive fairness, quantal response equilibrium, regret minimization, tit-for-tat, sigmoid acceptance learning, and randomized expected reward. We introduce a new approach called marginal value optimization. Where necessary, models were trained with data gathered from human players. We then created a collection of agent societies and conducted experiments where a single agent of each type was placed into societies of homogeneous agents of varying types. We also conducted experiments using Amazon Mechanical Turk where a single human played against these societies to establish a real-world baseline for performance that required adaptivity to diverse societies.

Our contributions in this paper are (1) a study of how various computational agents perform in diverse agent societies, (2) how these agents compare against human performance in these societies and (3) a simple approach based on single-step *marginal-value optimization* that outperforms humans.

2. RELATED WORK

The Ultimatum Game [8] has been studied extensively by many communities under various conditions in order to understand the diverse human behavior it elicits and its divergence from the "rational" Nash equilibrium strategy [4, 5, 7, 12, 13, 15, 17]. There are certain conditions under which people have gravitated towards the payoff-maximizing strategy including autism [12], or being familiar with economics [14] and others [3, 20, 21, 23]. However, there has been significant research showing that behavior varies greatly by culture and society to which one belongs [9, 10, 11, 18].

By adding the ability for the proposer to retrieve information about the recipient's past actions, others show that these adaptation rules, together with this added ability to retrieve *reputation* information, causes convergence to fair play [17]. Recently, researchers have been investigating genetic influences on play in the Ultimatum Game [24]. Most previous research investigates the one-shot game. In contrast, we investigate the multi-agent multi-round Social Ultimatum Game where players explore and build relationships with other players over time.

Building adaptive agents for economic contexts is an established and prominent research area in the multi-agent systems community, exemplified by the trading agent competition [25] and various similar competitions. For brevity, we only mention the background work that directly affects the agents we created for this investigation. Many of these agents are implemented within our Social Ultimatum Game framework, and are the subject of the comparative study conducted as part of this work.

A seminal work in using agent-based simulation to study human interaction was Axelrod's tournament for Prisoner's Dilemma that yielded the Tit-for-Tat strategy as a simple yet powerful strategy for success [2]. Recently, regret minimization has become a prominent method for modeling human behavior as well as for performing well under multiple scenarios [1].Quantal Response Equilibrium is a methodology for modeling human behavior that has been used in many settings including game-theoretic agents in security settings [16, 26]. A sigmoidal acceptance learning (SIGAL) approach was recently used to model social factors in a revelation negotiation game and outperformed humans and equilibrium agents [19]. An agent that used an adaptive model of fairness along with partial exploration was used to closely replicate human reciprocity over time in an Ultimatum Game context [6].

3. THE SOCIAL ULTIMATUM GAME

The classical Ultimatum Game is a two-player game where P_1 proposes a split of an endowment $e \in \mathbb{N}$ to P_2 where P_2 would receive $q \in \{0, \delta, 2\delta, \ldots, e-\delta, e\}$ for $\delta \in \mathbb{N}$. If P_2 accepts, P_2 receives q and P_1 receives $e-q$. If P_2 rejects, neither player receives anything.

The subgame-perfect Nash or Stackelberg equilibrium has P_1 offering $q = \delta$ (i.e., the minimum possible offer), and P_2 accepting, because a "rational" P_2 should accept any $q > 0$, and P_1 knows this. Yet, humans both make and reject offers that exceed δ.

The Social Ultimatum Game represents that people operate in societies of multiple agents and repeated interactions. Players, denoted $\{P_1, P_2, \ldots, P_N\}$, play $K \geq 2$ rounds, where $N \geq 3$. In each round k, every player P_m receives a new endowment e and chooses a recipient r_m^k and makes them an offer $q_{m,n}^k$ (where $n = r_m^k$). Each recipient P_n then considers the offers they received and makes a decision $d_{m,n}^k \in \{0, 1\}$ for each offer $q_{m,n}^k$ to accept (1) or reject (0) it. If the offer $q_{m,n}^k$ is accepted by P_n, then P_n receives $q_{m,n}^k$ and P_m receives $e - q_{m,n}^k$, where e is the endowment to be shared. If the offer $q_{m,n}^k$ is rejected by P_n, then both players receive nothing for that particular offer in round k. Thus, P_m's reward in round k, U_m^k, is the sum of the offers they receive and accept (if any are made to them) and their portion of the proposal they make, if their proposal was accepted. Thus, the utility to the m-th agent in the k-th round is

$$U_m^k = (e - q_{m,n}^k) d_{m,n}^k + \sum_{j=1\ldots N, j\neq m} q_{j,m}^k d_{j,m}^k \quad (1)$$

The total rewards for P_m over the game is the sum of per-round winnings, $U_m = \sum_{k=1}^{K} U_m^k$.

Figure 1 shows our Social Ultimatum Game interface. The given example is a 5-person 20-round game. Players are given avatars to preserve anonymity so that they cannot identify each other even when playing next to each other. The equilibrium analysis of the Social Ultimatum Game was described in [6].

4. EQUILIBRIUM ANALYSIS

We first address potential equilibrium strategies. First, let us characterize strategies by statistics that they produce in steady-state: the distribution of offers made by each player, where $p_m^g(n, q)$ denotes the likelihood that P_m will *give* an offer of q to P_n, and the distribution of offers accepted by each player, where $p_m^a(n, q)$ denotes the likelihood that P_m will *accept* an offer of q from P_n. Under these conditions,

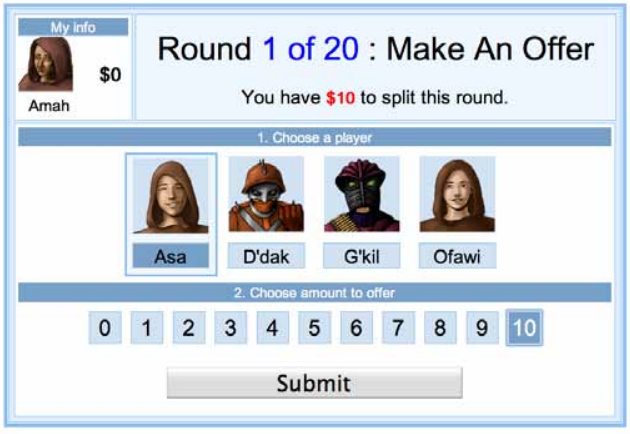


Figure 1: The Social Ultimatum Game web interface for a 5-person 20-round game. Players are given avatars to preserve anonymity so that they cannot identify each other even when playing next to each other.

the expected reward for P_m per round in steady-state is:

$$r_m = \sum_{n,q} q p_n^g(m,q) p_m^a(n,q) + \sum_{n,q} (e-q) p_m^g(n,q) p_n^a(m,q) \quad (2)$$

where $\sum_{n,q} p_m^g(n,q) = 1, \ \forall m$, as the total outgoing offers must total one offer per round, and the acceptance likelihoods are $p_m^a(n,q) \in [0,1], \ \forall m,n,q$. A player acting to maximize these rewards will modify their offer likelihoods $\{p_m^g(n,q)\}$ and acceptance likelihoods $\{p_m^a(n,q)\}$, given those of the other players. A player can always create the desired statistics by playing a stationary mixed strategy with the desired likelihoods. To optimize the offer likelihoods, P_m will set:

$$p_m^g(n,q) > 0, \forall n \in \mathcal{N}^g \subset \arg\max_n \max_q (e-q) p_n^a(m,q) \quad (3)$$

such that $\sum_{n,q} p_m^g(n,q) = 1$, and $p_m^g(n,q) = 0$, otherwise. Thus, in equilibrium, P_m will give offers to those agents whose acceptance likelihoods yield the highest expected payoff. Consequently, the offer likelihoods are a function of the acceptance likelihoods of the other players. A player's acceptance likelihoods are optimized with respect to the term $\sum_{n,q} q p_n^g(m,q) p_m^a(n,q)$ which is a function of the offer likelihoods. Thus, we can analyze steady state outcomes in terms of only the acceptance likelihoods, where the offer likelihoods are optimized, as discussed above, in some manner to meet the outgoing offer limitation of one per round.[1]

5. AUTONOMOUS AGENTS

To investigate various agent-based strategies as well as create diverse agent societies we considered a wide range of established approaches. We also present a new approach. We first summarize the approaches below and later provide more details about how they were adapted to the Social Ultimatum Game context.

- **Tit-for-Tat** : This is a fully reciprocal agent that chooses responders who previously made them offers, and offers an amount that reciprocates that previous offer. This is a standard approach based on its success in [2].
- **Regret Minimization** : This agent minimizes worst-case regret by hedging among a set of available actions. It hedges by increasing the weights associated with high payoff actions, and probabilistically chooses actions based on these weights, which are initialized using human data [1].
- **Expected Reward QRE** : This agent learns the expected rewards of various actions based on human play data and acts using a quantal response equilibrium strategy based on these rewards [16, 26].
- **SIGAL QRE** : This agent also uses a quantal response equilibrium strategy but the utility is based on the sigmoid acceptance learning approach which incorporates a model of social utility into the rewards [19].
- **Adaptive Fairness** : This agent is characterized by a fairness threshold which is dynamically updated based on an adaptability parameter and an exploration parameter. It accurately replicates human dynamic reciprocity behavior and is used as a stand-in for various human-like behaviors that are learned from data [6].
- **Marginal Value Optimization** : This agent chooses an action based on the marginal value of being seen as the preferred partner of each agent in the society. The value is a product of the expected value of the offer received from a particular agent and the marginal increase of the likelihood of receiving an offer. This new approach is a contribution of this paper.

[1]The results in this Section 4 were also described in [6]. We present them here to aid in explanation of the domain, and also to show the disadvantages of equilibrium-based strategies.

5.1 Tit-for-Tat

The Tit-for-Tat agent chooses the agent that made it the highest offer in the previous round as a responder and makes it an offer of the same value. If no one made it an offer in the previous round it chooses an agent at random and makes an offer of a preset value. As an agent society, we chose the preset value to be $2.

5.2 Regret Minimization (Hedging)

The Hedging agent treats the game as a multi-armed bandit problem and uses a novel extension of the EXP3 Hedging algorithm first described by Auer et al. [1] that we call SUG-Hedge. By playing according to the SUG-Hedge algorithm, the agent can guarantee that its *expected regret* will be minimized, with an upper bound on expected regret of

$$2\sqrt{e-1}\sqrt{KT\ln K},$$

where K is the number of possible actions and T is the number of time periods played. Here we use the usual definition of expected weak regret:

$$\max_{1\le j\le K} E_{a_1,\ldots,a_T}\left[\sum_{t=1}^{T} x_j(t)\right] - E_{a_1,\ldots,a_T}\left[\sum_{t=1}^{T} x_{a_t}(t)\right],$$

that is, the difference between the reward that *could have* been obtained had a different action been chosen over the T rounds of the game, and the reward that was actually obtained using the strategy determined by our algorithm. Due to space constraints, we omit the proof of the regret bound since it is not directly relevant to this paper.

The SUG-Hedge algorithm is fairly straightforward, and is shown below. We maintain weights for each of the possible actions $A_{i,j}$, with each action corresponding to a choice of player i to make an offer to, and a choice of the amount j to offer. Actions are chosen by sampling a distribution defined by these weights. When actions result in positive rewards, the weight for that action is multiplicatively updated. This update takes into account the fact that rewards can only be observed for the one action actually chosen; thus an expected reward is calculated and used for the update. By using an expected reward, we are able to adjust the weights appropriately, as if we had observed the reward resulting from each action at each time period.

We make a few further assumptions to increase the use of the information we gain from each action choice. This is easily seen by considering an example: Assume the SUG-Hedge algorithm makes an offer of $4 to player i, who accepts, and thus a reward of $6 is received. Then, we can assume that offers $\geq$ $4 to player i would also have been

Algorithm 1 SUG-Hedge

Parameters: $\gamma \in (0,1], K$ is total number of actions
Initialization: $\{w_{m,n}\}$

Repeat for round $t = 1, 2 \ldots$ until game ends
1. Get the probability of each action $A_{i,j}$

$$P_{i,j}^t = (1-\gamma)\frac{w_{i,j}}{\sum_{i,j} w_j} + \frac{\gamma}{K}$$

2. Choose an action A_{i_t,j_t} randomly according to the distribution $P_{1,1}^t, ..., P_{4,10}^t$
3. For $i = i_t, j \geqslant j_t$,
Revise reward of action $A_{i,j}$: $\widehat{X_{i,j}^t} = \frac{X_{i,j}^t}{\sum_{j' \leqslant j} P_{i,j'}^t}$
Update weight of action $A_{i,j}$: $w_{i,j}^{t+1} = w_{i,j}^t exp\left(\gamma \frac{\widehat{X_{i,j}^t}}{K}\right)$

accepted. SUG-Hedge thus updates not only $w_{i,4}$ but also $w_{i,j}$ where $j > 4$. This extension enables us to significantly tighten the bound described above. We also consider further extensions to SUG-Hedge that account for rewards due to player reciprocation, but the results are similar, and thus we omit them here. To initialize the weights $\{w_{m,n}\}$, we calculate the probabilities of different offer values based on the game data collected from human experiments (see Section 5.6), and initialize the weights to reflect the distribution of offer values exhibited in the data.

5.3 Expected Reward QRE

The quantal response equilibrium (QRE) model captures the fact that humans do not always choose the option that yields the highest expected utility. Instead, they follow a probabilistic, noisy decision process. Our QRE agent learns a static model of human action selection, using data from previous experiments where we collected data from games with five human players. This action selection model is then used at each round of the game. Thus, the agent does not adapt during game play.

The QRE model states that an action a is chosen with probability $p(a)$, where $p(a) = e^{\lambda U(a)}/\sum_{a \in A} e^{\lambda U(a)}$. We calculate the utilities $U(a)$ as the expected utility of playing an action a. To derive this quantity, we calculate the acceptance rate r_a of each offer amount a from our human game-play data. The expected utility for action a is thus $U(a) = r_a(10-a)$.

The parameter λ enables us to tune the amount of noise in the action choice. For example, $\lambda = 0$ results in uniform action choice, and $\lambda = \infty$ results in a best response action choice. Tuning λ is thus an important step in creating the QRE agent. We use maximum likelihood estimation (MLE) to fit λ using our human game-play data. The log likelihood of λ is

$$\log L(\lambda|a) = \lambda \sum_{a \in A} N_a U(a) - N \log\left(\sum_{a \in A} e^{\lambda U(a)}\right),$$

where N_a is the number of times action a was chosen in our data set. Since $\log L(\lambda|a)$ is a concave function, it has only one global maximum. Based on our human game-play dataset, the MLE of λ is 0.1020.

Feature	Value
BEN_p^k	7.6034
$P.BEN_p^k$	-7.3302
$P.BEN_d^k$	1.7826
Free Parameter	-0.5819

Table 1: SIGAL Parameters

5.4 SIGAL QRE

The Sigmoidal Acceptance Learning (SIGAL) agent introduced by Peled et al. [19] uses a social utility function instead of a selfish profit-maximizing utility function, and combines that with a sigmoidal acceptance function to determine action choice. The agent was shown to perform well relative to humans in two-agent negotiation tasks. Here we adapt the SIGAL agent to operate in the Social Ultimatum Game, which can be thought of as a multi-agent negotiation task. In fact, the Ultimatum Game was originally referred to as the Ultimatum Bargaining Game.

To adapt SIGAL for the Social Ultimatum Game, we first modified the set of features used to calculate the social utility. The features we used were:

- BEN_p^k: Proposer's benefit on the current round k,
- $P.BEN_p^k$: The benefit gained by the current proposer during the last interaction between the current proposer and decider, where the current proposer was also the proposer during that interaction,
- $P.BEN_d^k$: The benefit gained by the current decider during the last interaction between the current proposer and decider, where the current decider was the proposer during that interaction.

Thus we have the feature vector at round k:

$$x^k = (BEN_p^k, P.BEN_p^k, P.BEN_d^k)$$

Coefficients were learned using logistic regression after a standard normalization procedure

$$\hat{x} = \frac{x - std(X)}{mean(X)},$$

where X is the set of all feature vectors in our dataset. Table 1 shows the coefficient values learned in this way.

Since the original SIGAL agent was designed for a two-agent negotiation game, we also need to extend the agent to handle our multi-agent (five-person) game. We maintain the spirit of the agent by using a QRE model to select which of the four opponents to make an offer. Similar to the expected reward QRE agent, we estimate the λ parameter from data, using the expected social utility instead of the expected reward. In this case, the MLE of λ is 1.030.

5.5 Adaptive Fairness

The full description of the adaptive fairness approach is described in [6]. For clarity in understanding this work, we give a short description of the agent here. The desiderata for this agent incorporates the assumptions that people will: (1) start with some notion of a fair offer, (2) adapt these notions over time at various rates based upon their interactions, (3) have models of other agents, (4) choose the best option while occasionally exploring for better deals. Each player P_m is

characterized by three parameters: α_m^0 : P_m's initial acceptance threshold, β_m : P_m's reactivity and γ_m : P_m's exploration likelihood.

The value of $\alpha_m^0 \in [0, e]$ is P_m's initial notion of what constitutes a "fair" offer and is used to determine whether an offer to P_m is accepted or rejected. The value of $\beta_m \in [0, 1]$ determines how quickly the player will adapt to information during the game, where zero indicates a player who will not change anything from their initial beliefs and one indicates a player who will solely use the last data point. The value of $\gamma_m \in [0, 1]$ indicates how much a player will deviate from their "best" play in order to discover new opportunities where zero indicates a player who never deviates and one indicates a player who always does.

Each player P_m keeps a model of other players in order to determine which player to make an offer to, and how much that offer should be. The model is composed as follows: $a_{m,n}^k$: P_m's estimate of P_n's acceptance threshold; $\bar{a}_{m,n}^k$: Upper bound on $a_{m,n}^k$; and $\underline{a}_{m,n}^k$: Lower bound on $a_{m,n}^k$. Thus, P_m has a collection of models for all other players $\{[\underline{a}_{m,n}^k a_{m,n}^k \bar{a}_{m,n}^k]\}_n$ for each round k. The value $a_{m,n}$ is the P_m's estimate about the value of P_n's acceptance threshold, while $\underline{a}_{m,n}^k$ and $\bar{a}_{m,n}^k$ represent the interval of uncertainty over which the estimate could exist.

An important emergent property of this agent is that it can replicate the reciprocity dynamics of human players, without explicitly being coded to do so. This indicates that these agents, while not perfectly aligned, are a reasonable substitute for human-like agents and we can use them to create various societies.

5.6 Parameters learned from human data

As mentioned in the prior sections, we used a set of human experiments to learn the parameters of several of the described agents. These experiments were performed under two different settings: (1) University: Undergraduates and staff at a U.S. university, and (2) Conference: Attendees at an international conference with primarily computer science doctoral students and faculty.

We collected 40 game traces for the University setting, and 80 game traces for the Conference setting. Each game was a five-person, 20-round Social Ultimatum Game, with a $10 endowment given to each player in each round. Payouts were given relative to performance, with a conversion rate of US$0.025 for each Ultimatum Game dollar. From this data we estimated α, β and γ parameters for the adaptive fairness agents, using different subsets of humans for the different experiment settings described in the next section. These experiments include using the top 25% scorers in the Conference dataset, the top 25% scorers in the University dataset, two clusters of the human population based on offer value entropy (people who spread their offer values out the most and the least) and four humans drawn randomly from the populations. The learned parameters are shown in Table 2. The SIGAL-QRE, ER-QRE and Regret Minimization agents were also initialized with parameters learned from the same University and Conference datasets.

Agent	α	β	γ
Conference Top 25%	2.950	0.349	0.0675
University Top 25%	4.125	0.390	0.0562
Cluster1	3.075	0.287	0.0737
Cluster2	3.875	0.528	0.1475
Human #2	7	0.482	0.1
Human #16	4	0.330	0.1
Human #18	4	0.598	0.05
Human #39	2	0.631	0.15

Table 2: Parameters

5.7 Marginal Value Optimization

One of the fundamental gaps in the other approaches is that they do not leverage the possibility that the act of making an offer is also an opportunity to affect the mental model of other agents. An offer is not simply an opportunity for an agent to obtain a payoff but can also serve to increase the likelihood that another member of the society will choose the agent as a partner, or to possibly affect their notion of what is a fair offer if that society member is also adaptive.

We introduce the marginal value optimization (MVO) agent, which considers its offer as an opportunity to influence other agents. On each round it will make an offer value that it believes will make the target agent believe it is the most generous agent in the society.

Let us denote this value as the *top target value*. It chooses the target agent by calculating the marginal value of making that particular agent believe that it is the most generous agent in the society as follows. Let v_n^k be the marginal value of making the top target value to the n-th agent in the k-th round. Let $\hat{q}_n^k$ be the estimate of the offer value that the n-th agent would make to its target in k-th round. Let $p^*(q_{n,m}^k > 0)$ be the likelihood that the n-th agent will make the MVO agent an offer in the next round if the MVO agent offers it the top target value in the k-th round. Let $p(q_{n,m}^k > 0)$ be the likelihood that the n-th agent will make the MVO agent an offer in the next round if the MVO agent does not make it an offer in the k-th round. The marginal value of making the n-th agent a top target value offer in the k-th round is the estimate of the offer that the n-th agent will make back to the MVO agent multiplied by the marginal increase in the likelihood that the MVO agent will receive an offer:

$$v^k(n) = \hat{q}_n^k \cdot \Big(p^*(q_{n,m}^k > 0) - p(q_{n,m}^k > 0)\Big) \qquad (4)$$

This approach actually describes a space of possible approaches that are characterized by the functions that (1) determine the top target value, (2) estimate the value of the returning offer, (3) estimate the likelihood of an offer being reciprocated if the top target value offer is made and (4) the decay in that likelihood if an offer is not made.

Here, we make very simple assumptions for these functions as follows: (1) the top target value is the best offer received in the last two rounds, (2) the estimate of the value of the returned offer is the last known offer from that agent, (3) the likelihood of reciprocation of a top target value offer is 1, and (4) the decay is linear such that it reaches zero when the number of non-offer rounds is the number of players. All value estimates are set to $5 until we receive data during game play. All offers made to the MVO agent are accepted.

6. EXPERIMENTS

The experiments evaluate the adaptiveness of the various agents, when playing against a society of other players. In each experiment, we create a five-player game, where a given *test* player is evaluated when playing with four other *society* agents drawn from a single society type. Each game is 20

rounds, with a $10 endowment provided to each player in each round. The agent types for the test player were described in the previous section. The ten different societies are as follows:

- **AF-Conf-Top25** : 4 Conference Top 25% AF-agents. This represents the top scoring human strategies in the Conference dataset.
- **AF-Univ-Top25** : 4 University Top 25% AF-agents. This represents the top scoring human strategies in the University dataset.
- **AF-Cluster-1** : 4 Cluster 1 AF-agents. The Conference data was clustered into two sets: one which exhibited high offer value entropy, and one which exhibited low offer value entropy. Cluster 1 represented 40 humans who did not change their offer values very much over time.
- **AF-Cluster-2** : 4 Cluster 2 AF-agents. Cluster 2 represented 40 humans who spread out their offer values.
- **AF-Alpha-7**: : 4 AF-agents, all fit using Human #2. This society was chosen to stress the adaptiveness of the test agents by exposing them to a relatively unusual society. Here the human player would initially only accept offers of at least $7.
- **AF-4Types**: : AF-agents fit using Human #2, #16, #18, #39. This society is the only one which is composed of a mix of agents. We selected four random humans, and fit one AF agent for each of them. As shown in Table 2, the players varied in terms of initial offer acceptance threshold, and had generally reasonable rates of adaptiveness and exploration, with some variation exhibited across the four players.
- **SIGAL-QRE** : 4 SIGAL-QRE agents.
- **ER-QRE** : 4 ER-QRE agents.
- **Regret** : 4 Regret Minimization agents.
- **TFT-2** : 4 Tit-for-Tat agents with baseline $2 offers.

We evaluate six different types of test agents: (1) Human, (2) Regret, (3) SIGAL-QRE, (4) ER-QRE, (5) Tit-for-Tat, and (6) Marginal Value Optimization (MVO). We also evaluated other variations of these test agent types, as well as test agents based on Adaptive Fairness, but omit the results here for brevity and because they do not add additional insight to this discussion. Here we present the results of the sixty different pairings of one test agent with one society. For each pairing of one artificial agent and one society, we run 1000 games in simulation to obtain an estimate of the performance in each setting. For each pairing of one human and one society, we use Amazon's Mechanical Turk to obtain the results, obtaining 20 trials for each such pairing.

In total, we ran 200 games using the Mechanical Turk service. Each game was created as a Human Intelligence Task (HIT) inside Turk. In each game, the human players were paid according to their performance, with a conversion rate of US$0.01 for each $1 earned in the Social Ultimatum Game. Most players earned between $1.5 and $2 per game.

If it was the first time that a player accepted one of our HITs, the player would be asked to complete a compliance test before starting an actual game. This ensures that the experiment subjects understood our game rules and would provide useful data. The compliance test first described the game rules, provided the user with a chance to play a sample 3-round game, and ended with four short true/false questions to verify that the player understood the rules of the game. If the answers were wrong, the player had to restart the process by reviewing the game rules again. After successfully finishing the compliance test, we also gave the user a short survey to get background information including gender, occupation, age, education, and nationality.

Players could then play the 20-round games described above, paired against a society of four agents. They used the interface shown in Figure 1. The type of society was not revealed to the human player, and in fact, the human players were not told that they were actually playing against artificial agents. A random delay introduced between the rounds fostered the perception that they were playing against other humans, who might take varying amounts of time to decide on offer amounts and acceptances.

7. RESULTS

The experiment data is summarized in Figure 2. This table shows the mean and standard deviations of payoffs for the test players in the 10 different agent societies. We can draw a couple conclusions from this data.

Marginal value optimization (MVO) agent outperforms human players: The main result of the paper is that the marginal value optimization (MVO) agent outperforms human players in 9 out of 10 societies. In 7 out of 10 societies the differences in the mean payoff were very high; the MVO test agent outperformed the human test agent payoff by 16.6, 17.9, 24.5, 35.9, 41.6, 42.1, and 65.7. The only society where it does not outperform humans is the ER-QRE society, where humans obtained an average score that was $11.70 higher. The ER-QRE society is made up of agents which follow a static policy, i.e., what transpires during the execution of the game does not affect ER-QRE agent decisions at all - it is a static randomizer.

Most agents perform better within their own societies: The other agents (SIGAL-QRE, ER-QRE, Regret, TFT-2) can generally outperform humans and MVO agents when put within their own society. They also outperform humans in societies similar to their own. In particular, these agent types all have an "accept everything" rule, and perform well in societies composed of any of these five types of agents. However, they suffer significant degradation when wandering into societies composed of Adaptive Fairness type agents. This demonstrates the lack of adaptability of these agents, since they only succeed in societies similar to their own, and cannot adapt to societies that have a different social norm. In the societies composed of Adaptive Fairness agents, not all offers are accepted; only fair offers are accepted.

Unlike these agents, humans generally exhibit good adaptability to different societies, such as the societies composed of Adaptive Fairness agents. The results suggest that they are not as adaptive as one might initially hope to observe, however. For example, the AF-society where the fairness threshold is very high causes problems for humans, as does the society composed of regret minimizing agents. In the first case, humans cannot adjust to the high threshold and make many offers which are rejected; in the second case, humans reject some low offers and thus miss out on many

20 Round game		Society																			
		AF-Conf-Top25		AF-Univ-Top25		AF-Cluster-1		AF-Cluster-2		AF-Alpha-7		AF-4Types		SIGAL-QRE		ER-QRE		Regret		TFT-2	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
Test Player	Human	199.2	50.2	193.5	60.6	149.7	60.7	173.1	60.7	86.9	33.0	211.6	37.3	167.4	25.5	186.4	30.6	138.7	43.9	192.0	12.0
	MVO	203.6	43.9	229.3	80.7	215.4	41.5	215.1	63.3	111.4	54.0	221.3	48.5	184.1	14.8	174.7	13.5	180.2	16.6	209.9	5.7
	SIGAL-QRE	185.7	32.0	109.6	50.4	147.8	35.6	140.1	38.8	64.3	31.0	168.4	38.0	200.0	18.0	210.2	19.9	178.3	18.8	205.9	6.3
	ER-QRE	146.2	37.4	138.5	56.1	141.5	40.0	131.5	39.7	88.0	44.6	167.4	41.1	190.8	20.3	200.0	22.2	170.5	20.9	202.7	6.8
	Regret	172.3	37.4	98.4	44.4	143.9	34.2	140.9	39.6	62.7	28.0	162.1	38.5	224.1	21.1	231.3	22.9	200.0	20.8	199.0	8.0
	TFT-2	93.4	66.5	162.9	74.8	95.7	59.9	113.8	64.4	104.8	66.9	167.3	54.9	209.0	11.2	207.5	11.7	199.3	13.5	200.0	11.1

Figure 2: Mean and Standard Deviation of Payoffs for Test Players in Various Societies

future offers (and corresponding payoffs). Even given the limits of human adaptability, the MVO agent is the only approach that can almost uniformly outperform humans.

To investigate why the MVO agent is able to outperform humans, we separate the payoffs received by the humans and MVO agents into payoffs received from offers received and payoffs received by offers made. The results for the various societies are shown in Figure 3. We see that MVO's assumptions about generating payoffs from others by being the top target is validated. The MVO agent is able to generate more payoffs from offers made to it by others, when compared to human players in all societies.

Furthermore, MVO is also able to generate more payoffs from its own offers when compared to humans in 7 out of 10 societies. This is because the generous offer reduces the probability of rejection in several of the societies. However, it pays a price for this in societies where the probability of rejection is low (or zero). In two of the three cases, it is able to overcome this loss from improvement in the number and quality of offers made to it by others.

8. SUMMARY AND FUTURE WORK

We investigated the efficacy of several negotiation agents with respect to human performance in adapting to diverse societies. We use the Social Ultimatum Game for this investigation because of its rich and deep history as a domain for studying human behavioral diversity due to socio-cultural and other factors. Drawing from the literature, we generated a large collection of agent approaches including full reciprocity, regret minimization, quantal response equilibrium and sigmoid acceptance learning.

We performed human subject experimentation for data gathering in three stages culminating in a study with over 100 participants in Amazon Mechanical Turk. We introduced an approach based on marginal value optimization which leverages the strategic use of generosity to become the favorite partner of other agents in the society. This approach is able to outperform humans in a wide range of settings. This investigation shows a path to creating adaptive agents that perform well in a large set of societies. Applications of the work include developing strategies for automated trading in domains where norms are significant and creating agents for negotiating training in culturally-diverse settings.

There is significant room for improvement. We used simple estimation functions for many components of the MVO methodology. More accurate estimation may lead to greater performance. Investigating these parameters is a direction for future study. Also, we plan on testing the various adaptive agents in societies of multiple human agents. There are additional variations including game duration and endowment heterogeneity that are of interest. We also plan on creating social versions of other games such as Prisoner's Dilemma, Trust, Dictator, etc. to investigate if we can create agents that can perform well in multiple games and not just multiple societies.

9. REFERENCES

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, 1995.

[2] R. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211:1390–96, 1981.

[3] S. Blount. When social outcomes aren't fair: The effect of causal attributions on preferences. *Organizational Behavior and Human Decision Processes*, 63(2):131–144, 1995.

[4] T. Brenner and N. J. Vriend. On the behavior of proposers in ultimatum games. *Journal of Economic Behavior & Organization*, 61(4):617–631, 2006.

[5] C. R. P. J. Carnevale. Group choice in ultimatum bargaining. *Organizational Behavior and Human Decision Processes*, 72(2):256–279, 1997.

[6] Y.-H. Chang, T. Levinboim, and R. Maheswaran. The social ultimatum game. In *Decision Making with Imperfect Decision Makers*, 2011.

[7] R. H. Frank, T. Gilovich, and D. T. Regan. Does studying economics inhibit cooperation? *The Journal of Economic Perspectives*, 7(2):159–171, 1993.

[8] W. Güth, Schmittberger, and Schwarze. An experimental analysis of ultimatum bargaining. *Journal of Economic Behavior and Organization*, 3(4):367–388, 1982.

[9] J. Henrich. Does culture matter in economic behavior? ultimatum game bargaining among the machiguenga. *American Economic Review*, 90(4):973–979, 2000.

[10] J. Henrich, R. Boyd, S. Bowles, C. Camerer, E. Fehr, H. Gintis, R. McElreath, M. Alvard, A. Barr, J. Ensminger, N. S. Henrich, K. Hill, F. Gil-White, M. Gurven, F. W. Marlowe, J. Q. Patton, and

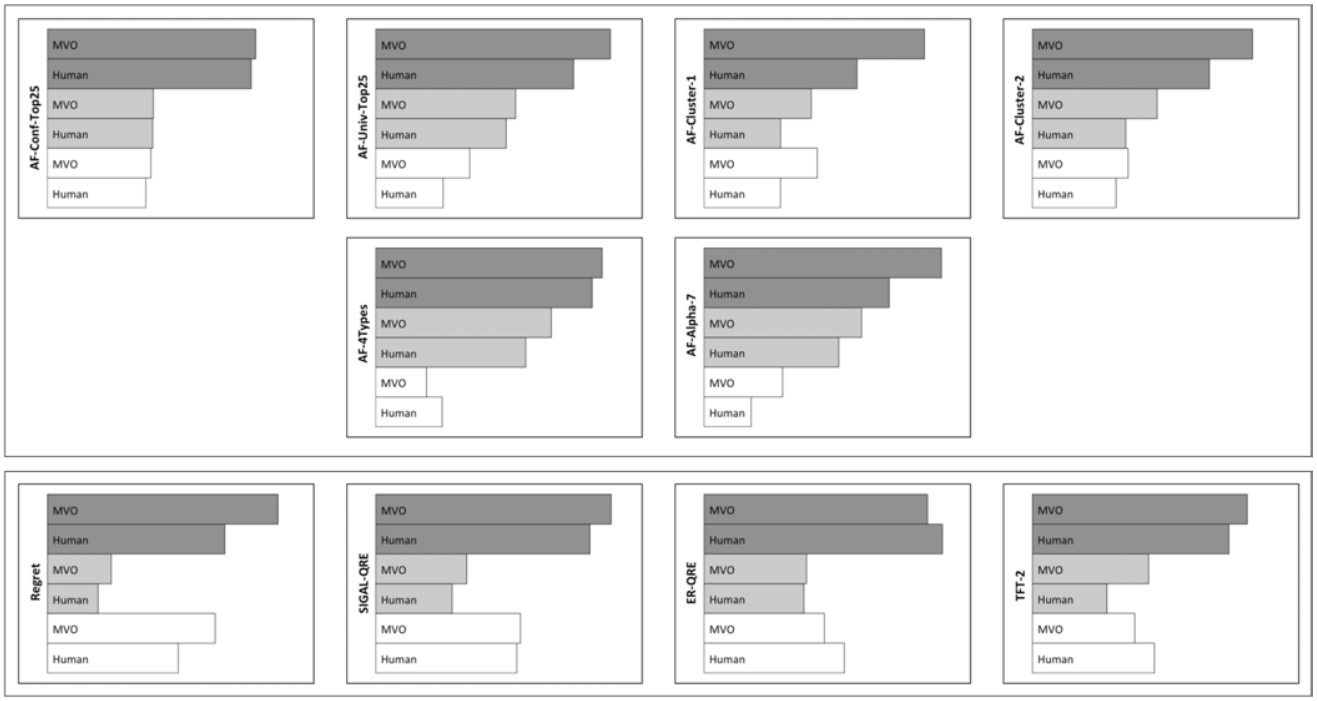


Figure 3: Total Payoffs (dark grey), Payoffs from Offers Received (light grey) and Payoffs from Offers Made (white) for Human and MVO Test Players in Various Societies

D. Tracer. "economic man" in cross-cultural perspective: Behavioral experiments in 15 small-scale societies. *Behavioral and Brain Sciences*, 28:795–815, 2005.

[11] J. Henrich, S. J. Heine, and A. Norenzayan. The weirdest people in the world? *Behavioral and Brain Sciences*, 33(2-3):61–83, 2010.

[12] E. Hill and D. Sally. Dilemmas and bargains: Theory of mind, cooperation and fairness. Working paper, University College, London, 2002.

[13] J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, Cambridge, 1998.

[14] D. Kahneman, J. L. Knetsch, and R. H. Thaler. Fairness and the assumptions of economics. *The Journal of Business*, 59(4):S285–S300, 1986.

[15] A. G. S. Mascha van't Wout, René S. Kahn and A. Aleman. Affective state and decision-making in the ultimatum game. *Experimental Brain Research*, 169(4):564–568, 2006.

[16] R. McKelvey and T. Palfrey. Quantal response equilibria for extensive form games. *Experimental Economics*, 1:9–41, 1998.

[17] M. A. Nowak, K. M. Page, and K. Sigmund. Fairness versus reason in the ultimatum game. *Science*, 289(5485):1773 – 1775, September 2000.

[18] H. Oosterbeek, R. Sloof, and G. van de Kuilen. Cultural differences in ultimatum game experiments evidence from a meta-analysis. *Experimental Economics*, 7(2):171–188, 2004.

[19] N. Peled, Y. Gal, and S. Kraus. A study of computational and human strategies in revelation games. In *Ninth International Conference on Autonomous Agents and Multi-Agent Systems*, 2011.

[20] J. K. Rilling, A. G. Sanfey, J. A. Aronson, L. E. Nystrom, and J. D. Cohen. The neural correlates of theory of mind within interpersonal interactions. *Neuroimage*, 22(4):1694–1703, 2004.

[21] A. G. Sanfey, J. K. Rilling, J. A. Aronson, L. E. Nystrom, and J. D. Cohen. The neural basis of economic decision-making in the ultimatum game. *Science*, 300(5626):1755–1758, 2003.

[22] D. Traum, S. C. Marsella, J. Gratch, J. Lee, and A. Hartholt. Multi-party, multi-issue, multi-strategy negotiation for multi-modal virtual agents. In *Proceedings of the 8th international conference on Intelligent Virtual Agents*, IVA '08, pages 117–130, Berlin, Heidelberg, 2008. Springer-Verlag.

[23] M. van't Wout, R. S. Kahn, A. G. Sanfey, and A. Aleman. ffective state and decision-making in the ultimatum game. *Experimental Brain Research*, 169(4), 2006.

[24] B. Wallace, D. Cesarini, P. Lichtenstein, and M. Johannesson. Heritability of ultimatum game responder behavior. *Proceedings of the National Academy of Sciences*, 2007.

[25] M. P. Wellman, A. Greenwald, and P. Stone. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press, 2007.

[26] R. Yang, C. Kiekintveld, F. Ordonez, M. Tambe, and R. John. Improving resource allocation strategy against human adversaries in security games. In *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Using Dynamic Rewards to Learn a Fully Holonomic Bipedal Walk

Patrick MacAlpine
Department of Computer Science
The University of Texas at Austin
Austin, TX 78701, USA
patmac@cs.utexas.edu

Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78701, USA
pstone@cs.utexas.edu

ABSTRACT

This paper presents the design and learning architecture for a fully holonomic omnidirectional walk used by the UT Austin Villa humanoid robot soccer agent acting in the RoboCup 3D simulation environment. By "fully holonomic" we mean the walk allows for movement in all directions with equal velocity. The walk is based on a double linear inverted pendulum model and was originally designed for the actual physical Nao robot. Parameters for the walk are optimized for maximum speed and stability while at the same time a novel approach of reweighting rewards for walking speeds in the cardinal directions of forwards, backwards, and sideways is utilized to promote equal walking velocities in all directions. A variant of this walk which uses the same walk engine, but is not fully holonomic as it employs three different sets of learned walk parameters biased toward maximizing forward walking speed, was the crucial component in the UT Austin Villa team winning the 2011 RoboCup 3D simulation competition. Detailed experiments reveal that adaptively changing the weights of rewards over time is an effective method for learning a fully holonomic walk. Additional data shows that a team of agents using this learned fully holonomic walk is able to beat other teams, including that of the 2011 RoboCup 3D simulation champion UT Austin Villa team, that utilize non-fully holonomic walks.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Parameter learning*; I.2.9 [**Artificial Intelligence**]: Robotics—*Kinematics and Dynamics*

General Terms

Algorithms, Design, Experimentation

Keywords

Bipedal walking, Robot soccer, Machine learning, CMA-ES

1. INTRODUCTION

In this paper, we investigate learning a fully holonomic humanoid walk in the robot soccer domain. By "fully holonomic" we mean a walk that allows for movement in all directions with equal velocity. This is in contrast to the "non-fully holonomic" learned walk used by the 2011 RoboCup[1] 3D simulation league champion team UT Austin Villa. The 2011 UT Austin Villa team's walk employs three different sets of learned walk parameters biased toward maximizing forward walking speed as the kinematics of the simulated robot model inherently allow for walking forwards faster than walking sideways. Although the learned walk of the 2011 UT Austin Villa soccer agent was the key component in the team winning the 3D simulation competition [9], its heavy emphasis on forward walking speed causes in to be significantly slower in other walking directions such as backward and sideways. This lack of speed when not moving forward slows the agent's reaction time and does not allow for quick changes of direction. In order to decrease this delay in changing directions, we would like to learn a set of walk parameters for the walk engine mentioned in Section 3 that allows for equal velocities in all walk directions.

The research reported in this paper is performed within a complex simulation domain, with realistic physics, state noise, multi-dimensional actions, and real-time control. In this test domain, teams of nine autonomous humanoid robots play soccer in a physically realistic environment. Though no simulation perfectly reflects the real world, the physical realism of the RoboCup domain enables pertinent research on realistic robotic tasks such as humanoid locomotion. An important advantage of working in simulation is that extensive experimentation and learning is possible without risk of mechanical wear and tear on any physical device.

As each robot is controlled through low-level commands to its joint motors, getting the robot to walk without falling over is a non-trivial challenge. In this paper, we describe a parameterized omnidirectional walk engine, whose parameters directly affect the speed and stability of the robot's walk, such as step height, length, and timing. Optimal parameters would allow the robot to stably walk as fast as possible in all situations. However, the set of possible walking directions is continuous, so it is infeasible to learn specific parameters for each direction. Therefore, the robot learns a general set of parameters with the goal for it to be able to move equally well in all directions.

The primary contribution of this paper is a methodology for adaptively changing the weights of rewards over time to encourage fast yet close to equal speeds for movement in all directions. There has been some related work in the area of bipedal locomotion such as using dynamic shaping rewards to integrate prior domain knowledge into the learning process for faster walking speeds [8]. Our work differs in that we are not incorporating domain knowledge into the learning process, but are instead trying to learn a walk with two conflicting objectives: fast walking speed and equal walking

[1] `http://www.robocup.org/`

velocities in all directions. These objectives clash with each other as an increase in speed in one direction often results in a decrease in speed in the perpendicular direction due to the robot's kinematics and joint structure.

The rest of the paper is structured as follows. Section 2 gives a domain description. Sections 3 and 4 describe our agent's omnidirectional walk engine and associated parameters for optimization respectively. Section 5 details the non-fully holonomic multiple parameter set walk optimization framework used by the 2011 champion UT Austin Villa agent. In Section 6 we discuss a fully holonomic walk optimization framework using dynamic rewards. Game performance results of agents with different learned walks are given in Section 7, and Section 8 summarizes.

2. DOMAIN DESCRIPTION

Robot soccer has served as an excellent platform for testing learning scenarios in which multiple skills, decisions, and controls have to be learned by a single agent, and agents themselves have to cooperate or compete. There is a rich literature based on this domain addressing a wide spectrum of topics from low-level concerns, such as perception and motor control [4, 10], to high-level decision-making problems [7, 11].

The RoboCup 3D simulation environment is based on SimSpark [3], a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine [2] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints used in the humanoid agents.

The robot agents in the simulation are homogeneous and are modeled after the Aldebaran Nao robot [1], which has a height of about 57 cm, and a mass of 4.5 kg. The agents interact with the simulator by sending torque commands and receiving perceptual information. Each robot has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. In order to monitor and control its hinge joints, an agent is equipped with joint perceptors and effectors. Joint perceptors provide the agent with noise-free angular measurements every simulation cycle (20 ms), while joint effectors allow the agent to specify the torque and direction in which to move a joint. Although there is no intentional noise in actuation, there is slight actuation noise that results from approximations in the physics engine and the need to constrain computations to be performed in real-time. Visual information about the environment is given to an agent every third simulation cycle (60 ms) through noisy measurements of the distance and angle to objects within a restricted vision cone (120°). Agents are also outfitted with noisy accelerometer and gyroscope perceptors, as well as force resistance perceptors on the sole of each foot. Additionally, agents can communicate with each other every other simulation cycle (40 ms) by sending messages limited to 20 bytes. Figure 1 shows a visualization of the Nao robot and the soccer field during a game.

3. WALK ENGINE

The UT Austin Villa 2011 team used an omnidirectional walk engine based on one that was originally designed for the real Nao robot [5]. The omnidirectional walk is crucial for allowing the robot to request continuous velocities in the forward, side, and turn directions, permitting it to approach continually changing destinations (often the ball) more smoothly and quickly than the team's previous year's set of unidirectional walks [12].

We began by re-implementing the walk for use on physical Nao robots before transferring it into simulation to compete in the RoboCup 3D simulation league. Many people in the past have used simulation environments for the purpose of prototyping real robot behaviors; but to the best of our knowledge, ours is the first work to use a real robot to prototype a behavior that was ultimately deployed in a simulator. Working first on the real robots lead to some important discoveries. For example, we found that decreasing step sizes when the robot is unstable increases its chances of catching its balance. Similarly, on the robots we discovered that the delay between commands and sensed changes is significant, and this realization helped us develop a more stable walk in simulation.

The walk engine, though based closely on that of Graf et al. [5], differs in some of the details. Specifically, unlike Graf et al., we use a sigmoid function for the forward component and use proportional control to adjust the desired step sizes. Our work also differs from Graf et al. in that we optimize parameters for a walk in simulation while they do not. For the sake of completeness and to fully specify the semantics of the learned parameters, we present the full technical details of the walk in this section. Readers most interested in the optimization procedure can safely skip to Section 4. The walk engine uses a simple set of sinusoidal functions to create the motions of the limbs with limited feedback control. The walk engine processes desired walk velocities chosen by the behavior, chooses destinations for the feet and torso, and then uses inverse kinematics to determine the joint positions required. Finally, PID controllers for each joint convert these positions into torque commands that are sent to the simulator.

The walk first selects a trajectory for the torso to follow, and then determines where the feet should be with respect to the torso location. We use x as the forwards dimension, y as the sideways dimension, z as the vertical dimension, and θ as rotating about the z axis. The trajectory is chosen using a double linear inverted pendulum, where the center of mass is swinging over the stance foot. In addition, as in Graf et al.'s work [5], we use the simplifying assumption that there is no double support phase, so that the velocities and positions of the center of mass must match when switching between the inverted pendulums formed by the respective

Figure 1: A screenshot of the Nao humanoid robot (left), and a view of the soccer field during a 9 versus 9 game (right).

stance feet.

Notation	Description
maxStep_i^*	Maximum step sizes allowed for x, y, and θ
y_{shift}^*	Side to side shift amount with no side velocity
z_{torso}^*	Height of the torso from the ground
z_{step}^*	Maximum height of the foot from the ground
f_g^*	Fraction of a phase that the swing foot spends on the ground before lifting
f_a	Fraction that the swing foot spends in the air
f_s^*	Fraction before the swing foot starts moving
f_m	Fraction that the swing foot spends moving
ϕ_{length}^*	Duration of a single step
δ^*	Factors of how fast the step sizes change
y_{sep}	Separation between the feet
x_{offset}^*	Constant offset between the torso and feet
x_{factor}^*	Factor of the step size applied to the forwards position of the torso
err_{norm}^*	Maximum COM error before the steps are slowed
err_{max}^*	Maximum COM error before all velocity reach 0

Table 1: Parameters of the walk engine with the optimized parameters starred.

We now describe the mathematical formulas that calculate the positions of the feet with respect to the torso. More than 40 parameters were used but only the most important ones are described in Table 1. Note that many, but not all of these parameters' values were optimized as described in Section 4.

To smooth changes in the velocities, we use a simple proportional controller to filter the requested velocities coming from the behavior module. Specifically, we calculate $\text{step}_{i,t+1} = \text{step}_{i,t} + \delta(\text{desired}_{i,t+1} - \text{step}_{i,t}) \forall i \in \{x, y, \theta\}$. In addition, the value is cropped within the maximum step sizes so that $-\text{maxStep}_i \leq \text{step}_{i,t+1} \leq \text{maxStep}_i$.

The phase is given by $\phi_{start} \leq \phi \leq \phi_{end}$, and $t = \frac{\phi - \phi_{start}}{\phi_{end} - \phi_{start}}$ is the current fraction through the phase. At each time step, ϕ is incremented by $\Delta\text{seconds}/\phi_{length}$, until $\phi \geq \phi_{end}$. At this point, the stance and swing feet change and ϕ is reset to ϕ_{start}. Initially, $\phi_{start} = -0.5$ and $\phi_{end} = 0.5$. However, the start and end times will change to match the previous pendulum, as given by the equations

$$\begin{aligned} k &= \sqrt{9806.65/z_{torso}} \\ \alpha &= 6 - \cosh(k - 0.5\phi) \\ \phi_{start} &= \begin{cases} \frac{\cosh^{-1}(\alpha)}{0.5k} & \text{if } \alpha \geq 1.0 \\ -0.5 & \text{otherwise} \end{cases} \\ \phi_{end} &= 0.5(\phi_{end} - \phi_{start}) \end{aligned}$$

The stance foot remains fixed on the ground, and the swing foot is smoothly lifted and placed down, based on a cosine function. The current distance of the feet from the torso is given by

$$\begin{aligned} z_{frac} &= \begin{cases} 0.5(1 - \cos(2\pi \frac{t - f_g}{f_a})) & \text{if } f_g \leq t \leq f_a \\ 0 & \text{otherwise} \end{cases} \\ z_{stance} &= z_{torso} \\ z_{swing} &= z_{torso} - z_{step} * z_{frac} \end{aligned}$$

It is desirable for the robot's center of mass to steadily shift side to side, allowing it to stably lift its feet. The side to side component when no side velocity is requested is given by

$$\begin{aligned} y_{stance} &= 0.5 y_{sep} + y_{shift}(-1.5 + 0.5\cosh(0.5k\phi)) \\ y_{swing} &= y_{sep} - y_{stance} \end{aligned}$$

If a side velocity is requested, y_{stance} is augmented by

$$\begin{aligned} y_{frac} &= \begin{cases} 0 & \text{if } t < f_s \\ 0.5(1 + \cos(\pi \frac{t - f_s}{f_m})) & \text{if } f_s \leq t < f_s + f_m \\ 1 & \text{otherwise} \end{cases} \\ \Delta y_{stance} &= \text{step}_y * y_{frac} \end{aligned}$$

These equations allow the y component of the feet to smoothly incorporate the desired sideways velocity while still shifting enough to remain dynamically stable over the stance foot.

Next, the forwards component is given by

$$\begin{aligned} s &= \text{sigmoid}(10(-0.5 + \frac{t - f_s}{f_m})) \\ x_{frac} &= \begin{cases} (-0.5 - t + f_s) & \text{if } t < f_s \\ (-0.5 + s) & \text{if } f_s \leq t < f_s + f_m \\ (0.5 - t + f_s + f_m) & \text{otherwise} \end{cases} \\ x_{stance} &= 0.5 - t + f_s \\ x_{swing} &= \text{step}_x * x_{frac} \end{aligned}$$

These functions are designed to keep the robot's center of mass moving forwards steadily, while the feet quickly, but smoothly approach their destinations. Furthermore, to keep the robot's center of mass centered between the feet, there is an additional offset to the forward component of both the stance and swing feet, given by

$$\Delta x = x_{offset} + -\text{step}_x x_{factor}$$

After these calculations, all of the x and y targets are corrected for the current position of the center of mass. Finally, the requested rotation is handled by opening and closing the groin joints of the robot, rotating the foot targets. The desired angle of the groin joint is calculated by

$$\text{groin} = \begin{cases} 0 & \text{if } t < f_s \\ \frac{1}{2}\text{step}_\theta(1 - \cos(\pi \frac{t - f_s}{f_m})) & \text{if } f_s \leq t < f_s + f_m \\ \text{step}_\theta & \text{otherwise} \end{cases}$$

After these targets are calculated for both the swing and stance feet with respect to the robot's torso, the inverse kinematics module calculates the joint angles necessary to place the feet at these targets. Further description of the inverse kinematic calculations is given in [5].

To improve the stability of the walk, we track the desired center of mass as calculated from the expected commands. Then, we compare this value to the sensed center of mass after handling the delay between sending commands and sensing center of mass changes of approximately 80ms. If this error is too large, it is expected that the robot is unstable, and action must be taken to prevent falling. As the robot is more stable when walking in place, we immediately reduce the step sizes by a factor of the error. In the extreme case, the robot will attempt to walk in place until it is stable. The exact calculations are given by

$$\begin{aligned} \text{err} &= \max_i(\text{abs}(\text{com}_{expected,i} - \text{com}_{sensed,i})) \\ \text{stepFactor} &= \max(0, \min(1, \frac{\text{err} - \text{err}_{norm}}{\text{err}_{max} - \text{err}_{norm}})) \\ \text{step}_i &= \text{stepFactor} * \text{step}_i \, \forall i \in \{x, y, \theta\} \end{aligned}$$

This solution is less than ideal, but performed effectively enough to stabilize the robot in many situations.

4. OPTIMIZATION OF WALK ENGINE PARAMETERS

As described in Section 3, the walk engine is parameterized using more than 40 parameters. We initialize these parameters based on our understanding of the system and by testing them on an actual Nao robot. We refer to the agent that uses this walk as the *Initial* agent.

The initial parameter values result in a very slow, but stable walk. Therefore, we optimize the parameters using the CMA-ES (Covariance Matrix Adaptation Evolution Strategy) algorithm [6], which has been successfully applied previously to a similar problem in [12]. CMA-ES is a policy search algorithm that successively generates and evaluates sets of candidates sampled from a multivariate Gaussian distribution. Once CMA-ES generates a group of candidates, each candidate is evaluated with respect to a *fitness* measure. When all the candidates in the group are evaluated, the mean of the multivariate Gaussian distribution is recalculated as a weighted average of the candidates with the highest fitnesses. The covariance matrix of the distribution is also updated to bias the generation of the next set of candidates toward directions of previously successful search steps. As CMA-ES is a parallel search algorithm, we were able to leverage the department's large cluster of high-end computers to automate and parallelize the learning. This allowed us to complete optimization runs requiring 210,000 evaluations in less than a day. This is roughly a 150 times speedup over not doing optimization runs in parallel which would have taken over 100 days to complete.

As optimizing 40 real-valued parameters can be impractical, a carefully chosen subset of 14 parameters was selected for optimization while fixing all other parameters. The chosen parameters are those that seemed likely to have the highest potential impact on the speed and stability of the robot. The 14 optimized parameters are starred in Table 1. Note that maxStep_i represents 3 parameters. Also, while f_g and f_s where chosen to be optimized, their complements f_a and f_m were just set to $(1 - f_g)$ and $(1 - f_m)$ respectively.

5. NON-FULLY HOLONOMIC WALK MULTIPLE SUBTASKS OPTIMIZATION

This section details how a non-fully holonomic multiple parameter set walk was optimized for use in the champion 2011 UT Austin Villa agent. This section serves to give both context and contrast to that of the fully holonomic walk optimization, described in Section 6, which utilizes the goToTarget optimization task presented in Section 5.1. Before describing the procedure for optimizing the walk parameters, we provide some brief context for how the agent's walk is typically used. These details are important for motivating the optimization procedure's fitness functions.

During gameplay the agent is usually either moving to a set target position on the field or dribbling the ball toward the opponent's goal and away from the opposing team's players. Given that an omnidirectional walk engine can move in any direction as well as turn at the same time, the agent has multiple ways in which it can move toward a target. We chose the approach of both moving and turning toward a target at the same time as this allows for both quick reactions (the agent is immediately moving in the desired direction) and speed (where the bipedal robot model is faster when walking forward as opposed to strafing sideways). We validated this design decision by playing our agent against a version of itself which does not turn to face the target it is moving toward, and found our agent that turns won by an average of .7 goals across 100 games. Additionally we played our agent against a version of itself that turns in place until its orientation is such that it is able to move toward its target at maximum forward velocity, and found our agent that immediately starts moving toward its target won by an average of .3 goals across 100 games. All agents we compared used walks optimized by the process described in this section.

Dribbling the ball is a little different in that the agent needs to align behind the ball, without first running into the ball, so that it can walk straight through the ball, moving it in the desired dribble direction. When the agent circles around the ball, it always turns to face the ball so that if an opponent approaches, it can quickly walk forward to move the ball and keep it out of reach of the opponent.

Similarly to a conclusion from [12], we have found that optimization works better when the agent's fitness measure is its performance on tasks that are *executed during a real game*. This stands in contrast to evaluating it on a general task such as the speed walking straight. Therefore, we break the agent's in-game behavior into a set of smaller tasks and sequentially optimize the parameters for each one of these tasks. Videos of the agent performing optimization tasks can be found online.[2]

5.1 Go to Target Parameter Set

In order to simulate common situations encountered in gameplay, the walk engine parameters are optimized for a goToTarget subtask.[3] This task consists of an obstacle course in which the agent tries to navigate to a variety of target positions on the field. Each target is active, one at a time for a fixed period of time, which varies from one target to the next, and the agent is rewarded based on its distance traveled toward the active target. If the agent reaches an active target, the agent receives an extra reward based on extrapolating the distance it could have traveled given the remaining time on the target. In addition to the target positions, the agent has stop targets, where it is penalized for any distance it travels. To promote stability, the agent is given a penalty if it falls over during the optimization run.

In the following equations specifying the agent's rewards for targets, $Fall$ is 5 if the robot fell and 0 otherwise, d_{target} is the distance traveled toward the target, and d_{moved} is the total distance moved. Let t_{total} be the full duration a target is active and t_{taken} be the time taken to reach the target or t_{total} if the target is not reached.

$$reward_{target} = d_{target}\frac{t_{total}}{t_{taken}} - Fall \quad (1)$$

$$reward_{stop} = -d_{moved} - Fall \quad (2)$$

The goToTarget optimization includes quick changes of

[2] www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2011/html/walk.html

[3] Note that we use three types of notation for each of goToTarget, *GoToTarget*, *goToTarget*, to distinguish between an optimization task, an agent created by this optimization task and a parameter set. Similarly for "sprint" and "initial".

- Long walks forward/backwards/left/right
- Walk in a curve
- Quick direction changes
- Stop and go forward/backwards/left/right
- Switch between moving left-to-right and right-to-left
- Quick changes of target to simulate a noisy target
- Weave back and forth at 45 degree angles
- Extreme changes of direction to check for stability
- Quick movements combined with stopping
- Quick alternating between walking left and right
- Spiral walk both clockwise and counter-clockwise

Figure 2: GoToTarget Optimization walk trajectories

target/direction for focusing on the reaction speed of the agent, as well as targets with longer durations to improve the straight line speed of the agent. The stop targets ensure that the agent is able to stop quickly, while remaining stable. The trajectories that the agent follows during the optimization are described in Figure 2. After running this optimization seeded with the initial walk engine parameter values we saw a significant improvement in performance. Using the parameter set optimized for going to a target, the *GoToTarget* agent was able to beat the *Initial* agent by an average of 8.82 goals with a standard error of .11 across 100 games.

5.2 Sprint Parameter Set

To further improve the forward speed of the agent, we optimized a parameter set for walking straight forwards for ten seconds starting from a complete stop. The robot was able to learn parameters for walking .78 m/s compared to .64 m/s using the *goToTarget* parameter set. Unfortunately, when the robot tried to switch between the forward walk and *goToTarget* parameter sets it was unstable and usually fell over. This instability is due to the parameter sets being learned in isolation, resulting in them being incompatible.

To overcome this incompatibility, we ran the goToTarget subtask optimization again, but this time we fixed the *goToTarget* parameter set and learned a new parameter set. We call these parameters the *sprint* parameter set, and the agent uses them when its orientation is within 15° of its target. The *sprint* parameter set was seeded with the values from the *goToTarget* parameter set. By learning the *sprint* parameter set in conjunction with the *goToTarget* parameter set, the new *Sprint* agent was stable switching between the two parameter sets, and its speed was increased to .71 m/s. Adding the *sprint* parameter set also improved the game performance of the agent slightly; over 100 games, the *Sprint* agent was able to beat the *GoToTarget* agent by an average of .09 goals with a standard error of .07.

5.3 Positioning Parameter Set

Although adding the *goToTarget* and *sprint* walk engine parameter sets improved the stability, speed, and game performance of the agent, the agent was still a little slow when positioning to dribble the ball. This slowness is explained by the fact that the goToTarget subtask optimization emphasizes quick turns and forward walking speed while positioning around the ball involves more side-stepping to circle the ball. To account for this discrepancy, the agent learned a third parameter set which we call the *positioning* parameter set. To learn this set, we created a driveBallToGoal2[4] optimization in which the agent is evaluated on how far it is able to dribble the ball over 15 seconds when starting from a variety of positions and orientations from the ball. The *positioning* parameter set is used when the agent is .8 meters from the ball and is seeded with the values from the *goToTarget* parameter set. Both the *goToTarget* and *sprint* parameter sets are fixed and the optimization naturally includes transitions between all three parameter sets, which constrained them to be compatible with each other. Adding the *positioning* parameter set further improved the agent's performance such that it, our *Final* agent, was able to beat the *Sprint* agent by an average of .15 goals with a standard error of .07 across 100 games. A summary of the progression in optimizing the three different walk parameter sets can be seen in Figure 3.

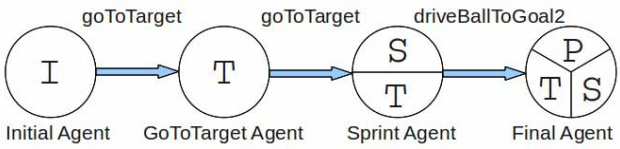


Figure 3: UT Austin Villa walk parameter optimization progression. Circles represent the set(s) of parameters used by each agent during the optimization progression while the arrows and associated labels above them indicate the optimization tasks used in learning. Parameter sets are the following: I = *initial*, T = *goToTarget*, S = *sprint*, P = *positioning*.

6. FULLY HOLONOMIC WALK OPTIMIZATION

One weakness of the non-fully holonomic walk learned in Section 5 is that the optimization process's heavy emphasis on forward walking speed results in significantly slower speeds in other walking directions such as backward and sideways. While this is somewhat mitigated by having an agent always turn to face the direction it is moving, and thereby quickly switch to walking at full speed in the forward direction, the turning process slows down the agent and does not allow for quick changes of direction.

In order to decrease this delay in changing directions we would like to learn a set of walk parameters for the walk engine mentioned in Section 3 that allows for equal velocities in all walk directions. With such a fully holonomic walk there will be no need, and resulting delay, for the robot to change its orientation as it changes direction. As the kinematics of the simulated robot model inherently allow for walking forwards faster than walking sideways, attempting to maximize walking speeds in all direction is likely to learn a walk engine parameter set biased toward a faster forward walk at the expense of slower speed in the sideways direction.

[4]The '2' at the end of the name driveBallToGoal2 is used to differentiate it from a driveBallToGoal optimization that was used in [12].

To account for the potential of the speed for any direction of the walk to dominate over the speed of other directions during the optimization process, we propose individually tallying the amount of reward given to walking in the three cardinal directions of forwards, backwards, and sideways, and then reweighting the rewards accumulated for these directions during the next iteration of CMA-ES so as to give more influence to directions in which the agent is walking slower. We implemented this idea by modifying the goToTarget optimization task mentioned in Section 5.1 to only count the positive reward from Equation 1 for three different walk targets that are part of the walk trajectories of the first item in Figure 2: long walks in the forward, backward, and sideways directions with each walk having a duration of 7 seconds. The positive rewards for these three walk targets are then all multiplied by weights whose values we adjust across iterations of CMA-ES, but whose sum is always normalized to 1, to calculate the overall positive portion of the fitness given to an agent (shown in Equation 4 for iteration i). During the first iteration of CMA-ES the values of these weights are all initialized to be 1/3 (Equation 3).

$$wgt_{i=1\{fw/bw/sw\}} = 1/3 \quad (3)$$

$$fit_{i\{positive\}} = rew_{i\{fw\}} * wgt_{i\{fw\}} + rew_{i\{bw\}} * wgt_{i\{bw\}} + rew_{i\{sw\}} * wgt_{i\{sw\}} \quad (4)$$

After every iteration of CMA-ES the three cardinal direction walk rewards (speeds) of the member of the population with the highest fitness are examined and used to calculate new reward weights for the next iteration of CMA-ES based on the following equations:

$$rew_{i\{fw/bw/sw\}} = max(rew_{i\{fw/bw/sw\}}, .1) \quad (5)$$

$$rew_{i\{max\}} = max(rew_{i\{fw,bw,sw\}})$$

$$wgt_{i+1\{fw/bw/sw\}} = rew_{i\{max\}}/rew_{i\{fw/bw/sw\}} \quad (6)$$

$$wgt_{i+1\{tot\}} = sum(wgt_{i+1\{fw,bw,sw\}})$$

$$wgt_{i+1\{fw/bw/sw\}} = wgt_{i+1\{fw/bw/sw\}}/wgt_{i+1\{tot\}} \quad (7)$$

Equation 5 first ensures that all direction rewards are positive which is necessary for the computation of reward weights. Reward weights are computed in Equation 6 and are equal to the factor that a directional reward needs to be multiplied by in order to be equal to that of the maximum directional reward. Finally Equation 7 normalizes all reward weights to sum to 1. We refer to the agent that uses this formulation for updating reward weights as the *DynamicRewards* agent.

Note that in addition to the positive portion of the fitness for an agent computed in Equation 4, the agent also still receives negative rewards for falling and movement when told to stop for all walk targets as described in Equations 1 and 2 in Section 5.1. This is done to ensure that the agent learns a walk that can quickly stop and is stable in the same way as the walk produced by the *goToTarget* parameter set in Section 5.1. As the agent no longer receives positive rewards for moving to all targets as in the original goToTarget optimization, and instead only receives a positive reward for moving to targets for a weighted total of 7 seconds, we need to reduce the value of the negative rewards so as to preserve the ratio between possible attainable positive and negative rewards present in the original goToTarget optimization. We found that not reducing the value of negative rewards causes the agent to learn walking parameters that keep it stationary. This is because the negative rewards incurred by moving when told to stop, and also potentially falling, dominate the possible rewards gained in just 7 seconds of measured positive movement toward walk targets. Our solution to this was to weight negative rewards by the ratio of current time moving to targets for which positive rewards are recorded to that of the same time as in the original goToTarget optimization (7/124.1).

An alternative to calculating new reward weights using the directional rewards of the member of the population with the highest fitness is to instead use a weighted average of the directional rewards of the top half of the population with the highest fitness. This weighted averaging is what CMA-ES does at the end of every iteration to update the mean of the multivariate Gaussian distribution it is sampling parameters from. This weighted averaging is computed by the following equations for which members of a population are first ranked and sorted in descending order of fitness ($i = 1$ for highest fitness member):

$$weight_i = log(popsize/2 + 1/2) - log(i)$$

$$weights_{sum} = \sum_{i=1}^{popsize/2} weight_i$$

$$weight_i = weight_i/weights_{sum}$$

$$rew_{avg\{fw/bw/sw\}} = \sum_{i=1}^{popsize/2} rew_{i\{fw/bw/sw\}} * weight_i$$

We call the agent that uses weighted averages of distance rewards to update reward weights the *DynamicAvgRewards* agent.

In addition to the two agents that change the weight of rewards over time (the *DynamicRewards* and *DynamicAvgRewards* agents), for comparison purposes we also ran optimizations on the modified goToTarget task for a couple of agents that do not modify rewards. The first of these is the *StaticRewards* agent which is optimized in the same way as the *DynamicRewards* agent except that it holds each of the directional reward weights constant at 1/3. Unlike the *Final* agent used by UT Austin Villa in the 2011 competition, which controls its orientation differently for different tasks as described in Section 5, the *DynamicRewards*, *DynamicAvgRewards*, and *StaticRewards* agents all directly move in the direction of their targets without purposely modifying their orientations in any way. The second agent which does not modify directional reward weights is the *FaceForward* agent. This agent always turns to face whatever target it is moving to and is similar to the the *GoToTarget* agent except that it was optimized using the modified version of the goToTarget task. All optimizations were done using CMA-ES with a population size of 150 across 200 generations.

Figure 5 shows how the directional weights for the *DynamicRewards*, *DynamicAvgRewards*, and *StaticRewards* agents vary across iterations of CMA-ES. Note that the weights used by the *StaticRewards* agent are fixed and that the weights shown in this figure are only computed and displayed for comparison purposes to show what the weights chosen would be if the agent was dynamically adjusting its weights. The *DynamicRewards* and *DynamicAvgRewards* agents' weights are very close together with a slightly higher weight for the forward direction (meaning that the forward direction is producing a slightly lower reward than the other directions). The fluctuation in weights is a little smoother

Table 2: Directional walking speeds of learned walks for different agents described in Section 6 as well as the *Final* agent used by UT Austin Villa in the 2011 RoboCup competition. All speeds are in m/s and were measured by recording the distance the agent traveled in ten seconds when starting from a complete stop.

Agent	Forward	Backward	Sideways
DynamicRewards	.42	.53	.48
DynamicAvgRewards	.45	.53	.51
StaticRewards	.58	.52	.37
FaceForward	.74	.35	.03
Final	.71	.40	.21

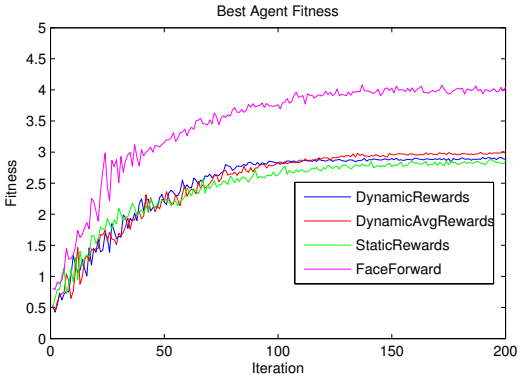


Figure 4: Best agent fitness across iterations of CMA-ES.

for the *DynamicAvgRewards* agent than that of the *DynamicRewards* agent as it is using averaging. The *StaticRewards* agents shows a considerable difference in weights, however, with a much higher weight (and thus lower reward) coming from the sideways direction. This is an indicator of the optimization finding it easier to optimize for speed in the forward direction than that of the sideways direction.

In Figure 4 the best agent fitnesses across iterations of CMA-ES are shown for the agents described in this section. Here we see that all agents' fitnesses start to plateau around iteration 100. The static and dynamic rewards agents all have similar fitnesses while the *FaceForward* agent has a higher fitness. The probable cause for this is because the lengthy long walks of 7 seconds in each direction allow the *FaceForward* agent plenty of time to turn and walk in the forward direction for which it is being optimized for.

Directional walking speeds of the different agents described in this section, as well as the *Final* agent used by UT Austin Villa in the 2011 RoboCup competition, and described in Section 5, are shown in Table 2. Here we see fairly close values across all directions for both the dynamic rewards agents. This gives proof that either method of adjusting the weights of rewards across iterations is a viable solution for learning a walk that is close to fully holonomic. The *StaticRewards* agent has a forward walk speed over 50% faster than it's side walk speed which shows evidence of the forward walk speed being easier to increase at the expense of the sideways walk speed. The *FaceForward* agent has the highest speed for the forward walk which is not surprising as this is the direction it is walking in for most of the time during optimization. Despite never walking backwards during optimization the agent still has a backward speed close to half its forward speed. This suggests a correlation in movement between walking forward and backward as they both lie in the sagittal plane. Walking sideways (movement in the opposite coronal plane) on the other hand never occurs during optimization which results in a speed in this direction of nearly 0. The *Final* agent from the 2011 competition has very good forward speed similar to the *FaceForward* agent, which it was generally optimized for, but also has much better sideways speed than the *FaceForward* agent due to using multiple learned parameter sets including the *positioning* parameter set for which sideways movement was included as part of the optimization.

7. AGENT GAME PERFORMANCE RESULTS

In Table 3 we present game results of the agents with different walk parameter sets described in Section 6 as well as the *Final* agent used in the 2011 competition and described in Section 5. Both the *StaticRewards* and *FaceForward* agents do very poorly against the other agents. Empirical evidence shows that the *StaticRewards* agent is too slow when trying to move sideways, which happens roughly half the time, allowing other agents to easily get around or steal the ball from the agent. The *FaceForward* agent, on the other hand, gets mired down in constantly turning and trying to adjust its position when around the ball as it is unable to move sideways.

The *DynamicRewards* and *DynamicAvgRewards* agents perform very similarly which isn't surprising considering how close their walk speeds are in Table 2. What is surprising is that they are both able to beat the champion *Final* agent from the 2011 competition which uses three learned walk parameter sets instead of just one. Despite the *Final* agent having a significantly faster top walking speed than both of the dynamic reward agents, the dynamic reward agents are much faster positioning around the ball due to the *Final* agents slow sideways speed and need for turning to adjust its orientation while circling the ball. While the average goal difference that the *DynamicRewards* agent beat the *Final* agent by across 100 games was only .20 goals, this still translated to 29 goals for with 9 against and a record of 23 wins with only 7 losses and 70 ties.

8. SUMMARY AND DISCUSSION

We have presented the design and learning architecture for a fully holonomic omnidirectional walk used by the UT Austin Villa humanoid robot soccer agent acting in the RoboCup 3D simulation environment. The key to our optimization method is using a novel approach of reweighting rewards for walking speeds in the cardinal directions of forwards, backwards, and sideways to promote equal walking velocities in all directions. A team of agents using this learned fully holonomic walk, which consists of just a single learned parameter set, is able to beat the UT Austin Villa 2011 RoboCup 3D simulation champion team that uses a

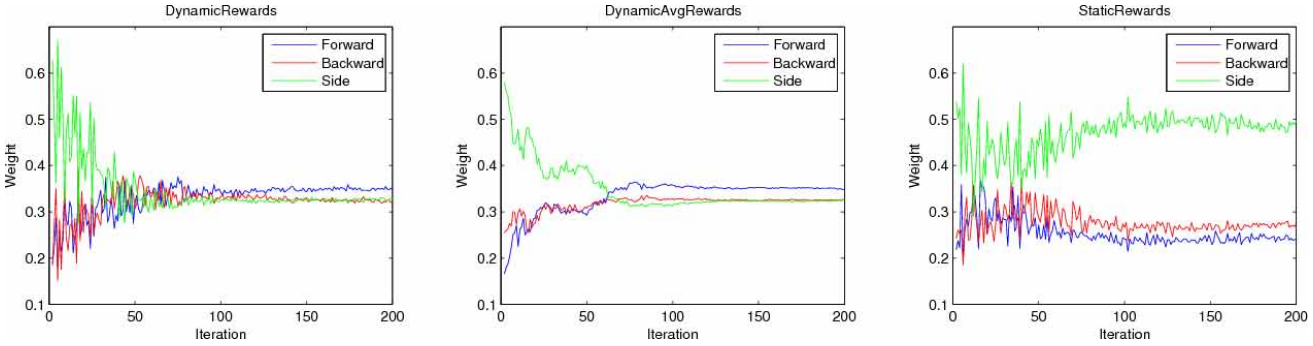


Figure 5: Directional reward weights during the course of optimization for the the *DynamicRewards* agent (left), *DynamicAvgRewards* agent (center), and *StaticRewards* agent (right). Note that the weights used by the *StaticRewards* agents are fixed and that the weights shown in this figure are only computed and displayed for comparison purposes to show what the weights chosen would be if the agent was dynamically adjusting its weights. Higher weights correlate to lower relative rewards.

Table 3: Game results of agents with different walk parameter sets described in Section 6 as well as the *Final* agent used in the 2011 competition and described in Section 5. Entries show the average goal difference (row − column) from 100 ten minute games. Values in parentheses are the standard error.

	Final	FaceForward	StaticRewards	DynamicAvgRewards
DynamicRewards	0.20(.08)	3.27(.09)	3.18(.11)	-0.06(.07)
DynamicAvgRewards	0.10(.07)	3.49(.11)	2.88(.11)	
StaticRewards	-2.77(.13)	0.22(.06)		
FaceForward	-2.99(.12)			

non-fully holonomic walk employing multiple walk parameter sets. This is a significant accomplishment as the 2011 UT Austin Villa team won all 24 games it played during the RoboCup competition while scoring 136 goals and conceding none.

Our ongoing research agenda includes applying what we have learned in simulation to the actual Nao robots which we use to compete in the Standard Platform league of RoboCup. Additionally, we would like to learn multiple parameter sets for the fully holonomic walk, specialized for walking in each of the different cardinal directions, in a similar fashion to how the *sprint* parameter set was learned in Section 5.2.

More information about the UT Austin Villa team, as well as video highlights from the 2011 competition, can be found online at the team's website.[5]

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. Thanks especially to UT Austin Villa 2011 team members Daniel Urieli, Samuel Barrett, Shivaram Kalyanakrishnan, Michael Quinlan, Francisco Barrera, Nick Collins, Adrian Lopez-Mobilia, Art Richards, Nicolae Ştiurcă, and Victor Vu. Also thanks to Yinon Bentor and Suyog Dutt Jain for contributions to early versions of the optimization framework employed by the team. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030). Patrick MacAlpine is supported by a NDSEG fellowship.

9. REFERENCES

[1] Aldebaran Humanoid Robot Nao. `http://www.aldebaran-robotics.com/eng/`.

[2] Open Dynamics Engine. `http://www.ode.org/`.

[3] SimSpark. `http://simspark.sourceforge.net/`.

[4] S. Behnke, M. Schreiber, J. Stückler, R. Renner, and H. Strasdat. See, walk, and kick: Humanoid robots start to play soccer. In *Proc. of the 6th IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids 2006)*, pages 497–503. IEEE, 2006.

[5] C. Graf, A. Härtl, T. Röfer, and T. Laue. A robust closed-loop gait for the standard platform league humanoid. In *Proc. of the 4th Workshop on Humanoid Soccer Robots in conjunction with the 2009 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 30 – 37, 2009.

[6] N. Hansen. *The CMA Evolution Strategy: A Tutorial*, January 2009. `http://www.lri.fr/~hansen/cmatutorial.pdf`.

[7] S. Kalyanakrishnan and P. Stone. Learning complementary multiagent behaviors: A case study. In *RoboCup 2009: Robot Soccer World Cup XIII*, pages 153–165. Springer, 2010.

[8] A. Laud and G. Dejong. Reinforcement learning and shaping: Encouraging intended behaviors. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 355–362. Morgan Kaufmann, 2002.

[9] P. MacAlpine, D. Urieli, S. Barrett, S. Kalyanakrishnan, F. Barrera, A. Lopez-Mobilia, N. Ştiurcă, V. Vu, and P. Stone. UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, June 2012.

[10] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

[11] P. Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, USA, December 1998.

[12] D. Urieli, P. MacAlpine, S. Kalyanakrishnan, Y. Bentor, and P. Stone. On optimizing interdependent skills: A case study in simulated 3D humanoid robot soccer. In *Proc. of the Tenth Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 769–776, May 2011.

[5] `www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/`

An Empirical Analysis of RL's Drift From Its Behaviorism Roots

Matthew Adams
North Carolina State University
mbadams3@ncsu.edu

Robert Loftin
North Carolina State University
rtloftin@ncsu.edu

Matthew E. Taylor
Lafayette College
taylorm@lafayette.edu

Michael Littman
Rutgers University
mlittman@cs.rutgers.edu

David Roberts
North Carolina State University
robertsd@ncsu.edu

ABSTRACT

We present an empirical survey of reinforcement learning techniques and relate these techniques to concepts from behaviorism, a field of psychology concerned with the learning process. Specifically, we examine two standard RL algorithms, model-free SARSA, and model-based R-MAX, when used with various shaping techniques. We consider multiple techniques for incorporating shaping into these algorithms, including the use of options and potential-based shaping. Findings indicate any improvement in sample complexity that results from shaping is limited at best. We suggest that this is either due to reinforcement learning not modeling behaviorism well, or behaviorism not modeling animal learning well. We further suggest that a paradigm shift in reinforcement learning techniques is required before the kind of learning performance that techniques from behaviorism indicate are possible can be realized.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Theory, Performance

Keywords

Markov decision processes, reinforcement learning, behaviorism

1. INTRODUCTION

Human-canine collaboration is unique and astonishingly successful. It is an existence proof that human beings can communicate complex tasks to non-human autonomous agents. With only minimal communication tools, humans and dogs can work together to play fetch, hunt, search for buried avalanche survivors, track missing persons, sniff for explosives/narcotics/contraband, or guide the blind. Published literature on training going back to Skinner's work on behaviorism [8] provides insight into how dogs learn, and how human trainers teach dogs complex tasks quickly and accurately.

At a very high level, machine reinforcement learning agents act in a way similar to animals undergoing training. They take actions from different states, and their behavior over time is affected by the rewards that they are given. In this paper, we report on a series of experiments designed to illustrate how the performance of two common machine reinforcement learning paradigms compares to the experiences human trainers have when training dogs. Specifically, we examine two popular reinforcement learning techniques: the model-free SARSA algorithm [9] and the model-based R-MAX algorithm [11]. We performed a survey of popular dog training methods methods and identified computational analogues to those approaches. We ran experiments to evaluate whether reinforcement learning would benefit from "shaping" inputs commonly used in behaviorism-inspired dog training techniques. We were looking for significant performance increases where learning would occur in orders of magnitude less time or for problems to be learned that weren't learnable without these techniques.

Although a few of our experiments resulted in faster learning rates for the agent, the benefits that were seen were nowhere near the improvements in learning rates seen in animals when corresponding training techniques are used. At best, applying shaping techniques to reinforcement learning agents resulted in about half as many atomic actions taken in order to learn an optimal policy. This pales in comparison to the effects shaping has on animals.

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

2. CANINE LEARNING

Shaping by successive approximation [13] is a technique used in animal training by which desired behaviors are taught to learners by means of selectively rewarding actions closer and closer to the desired behavior. At first, in order to obtain a reward, the animal only needs to perform an action that is vaguely similar to the desired one. Once the animal learns how to get the reward, the criteria is tightened and the animal has to perform actions that are more similar to the desired behavior in order to obtain a reward. Eventually, the criteria converges and the goal behavior is learned.

Clicker training is a popular paradigm for shaping by successive approximation in animal training [7]. First, the positive stimulus of a treat is paired with the click sound from a noise-making device. This process of charging the clicker makes it a "conditioned secondary reinforcer" using classical conditioning [6]. Because the click is associated with a treat, the click noise indicates a promise of a future reward. In order to train the dog to perform a certain behavior, a click is given whenever the dog's actions get closer to the target behavior. After every click, a primary reinforcer (*e.g.*, a treat) is provided and the episode is over. The criteria for clicking is tightened as the dog more closely approximates the target behavior.

Shaping by successive approximation in animal training allows animals to learn much more rapidly than if the same rewards were given only when the exact target behavior was performed. Without

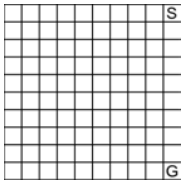


Figure 1: The 10x10 open grid with goal locations.

any cue as to the sort of behavior the animal is supposed to be doing, the animal can only behave arbitrarily until it finally stumbles into performing the behavior. The addition of the shaping rewards expedite the process by guiding the animal into the sort of behavior it is supposed to perform.

There are many ways that a trainer can use shaping concepts. The specifics of when to use what approach are part of the "art" of training. That being said, there is overwhelming evidence that these approaches enable efficient learning of complex tasks that wouldn't be realistic to learn without shaping (*cf.*, [3, 7, 8]). We believe that in order to get machine reinforcement learning to perform in some of the awe-inspiring ways we see human-canine teams perform, the algorithms must respond to the behavior shaping process in an appropriate way. The experiments we describe in subsequent sections of this paper illustrate that, unfortunately, RL algorithms do not respond favorably to shaping.

3. EXPERIMENTAL DESIGN

Reinforcement learning problems are related to Markov decision processes (MDP). A MDP is defined by a tuple $\{S, A, T, R\}$, with S the set of possible states, and A the set of actions which the agent can take. The transition function $T : S \times A \times S \mapsto [1 : 0]$ represents the probability of transitioning to state s' after performing action a in state s, and the reward function $R : S \times A \mapsto \mathbb{R}$, determines the reward received for performing action a in state s.

The goal of the learner is to find a policy $\pi : S \mapsto A$, that maximizes the expected discounted total reward that the learner receives. Model-based algorithms such as R-MAX learn explicit models of T and R, whereas model-free methods such as SARSA learn a function $Q : S \times A \mapsto \mathbb{R}$, which represents the expected discounted total reward for taking action a in state s. If the state and action spaces are discrete and of sufficiently small size, the transition and reward models, or the Q function, as well as the policy π, can all be represented as tables. For large or continuous state spaces, some form of function approximation, such as a neural network or tile-coding, is often used to represent these functions. For the purposes of this paper, we focus on tabular representations only. Our goal is not to scale these algorithms to large problems, but to see how their performance on smaller problems changes when different behavior shaping techniques are used in the learning process.

In order to determine the ability of current machine learning algorithms to operate in paradigms similar to and display performance characteristics similar to human-dog teams, we conducted a series of simulation experiments on an open grid world. At each timestep, the agent can move one step in any of the four cardinal directions, and the result of each action is deterministic. The goal state is in one corner of the grid, and reaching the goal state results in a reward of 1 and the end of the episode. All other states have a reward of zero. The discount factor was 0.8.

The initial state for each episode is in an adjacent corner to the goal state (see Figure 1). Defining the problem this way, rather than having the initial state and goal state on opposite corners, creates a region of the state space that will not be reached by applying an optimal policy, and therefore is not likely to be fruitful to explore.

One of the major advantages of using the simple grid space is the incremental adjustments that we can make to the size of the state space. The same dynamics of the space exist when the region is small (*e.g.*, 10x10) or large (*e.g.*, 100x100), so good comparisons can be made between methods on varying problem sizes.

Other reasons to use the grid space include the fact that the optimal policy is known and is easy to visualize. Because we know what the optimal policy should be, we can compare it to the policy that the agents actually learn and investigate how long it takes the agents to approximate the optimal policy.

The simplicity of the grid space allows us to see how the agent reacts to changes in the problem in accordance with what humans might do when training animals. Many reinforcement learning methodologies and improvements that have been investigated in the past are only applicable to learning domains with very particular properties. The basic grid space can accommodate many modifications that allow us to simulate the different sorts of training modifications that are done in animal training.

By using this simple environment, we were better able to compare the various algorithms' performance under different conditions. We sought to determine: ***Does the performance of learning algorithms improve in ways consistent with canine learners when canine training techniques are used during machine training?*** To that end, we tested both a model-based learning algorithm (R-MAX) and a model free learning algorithm (SARSA) in various conditions modeled after common canine training techniques. For each of these algorithms, we were looking for a *significant* improvement in performance when common variants of canine training techniques were applied to the machine training process. By "significant," we mean criteria commonly used in evaluating canine behavior: speed of learning and accuracy of learned behaviors. Due to the simplicity of our experimental domain, we focused entirely on learning speed in the survey.

These two learning algorithms were chosen because they are characteristic of the two major approaches to reinforcement learning problems. SARSA is a good representative of model free methods, whereas R-MAX is a good representative of model-based methods. Applying shaping techniques to both methods gives us a good survey of how reinforcement learning methods react in general to canine training techniques.

The SARSA Algorithm: The SARSA algorithm is an on-policy temporal difference reinforcement learning method. The algorithm builds a value table using the SARSA update rule, which updates values using the action that will be taken at the next step, as opposed to the greedy action. We used a learning rate $\alpha = 0.1$ and optimistic Q-value initialization combined with a pure exploit policy. In part, we did this because it is a good model of the behavior we observe in canines. They receive an "intrinsic" reward from the environment, but a more valuable reward from the trainer. Once a dog understands what behaviors cause rewards, it will relentlessly exploit the reward until something better comes along [4].

The R-MAX Algorithm: The R-MAX algorithm [11] estimates a model of both the MDP transition probabilities and reward function. These models are initialized to assume that all actions in all states deterministically transition to the fictitious state G_0, and receive a reward of R_{max}. The algorithm keeps a record of what transitions and rewards have been observed for each state and action. Initially, and whenever a new state-action pair becomes known, the algorithm computes an optimal policy under the current model. In our implementation this is done using value iteration with a convergence threshold of 0.1. The problem that the R-MAX agent learns

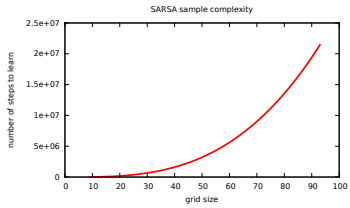


Figure 2: The number of steps taken to learn using SARSA as a function of the grid size.

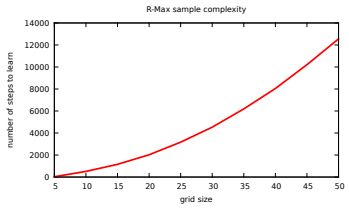


Figure 3: The growth rate of the number of actions required for R-MAX to learn the optimal policy, as a function of the width and height of the square grid.

on is exactly the same as for the SARSA agent, with rewards of 1 for the goal state and 0 for all other states. As the agent acts greedily, the parameter R_{max} is set optimistically to 1.0 to encourage exploration, and its value affects the degree to which the learner exploits its current knowledge. As the considered domains are fully deterministic, a state-action pair is considered known when it has been observed once.

Algorithm Performance: To be able to make solid empirical comparisons of the effects of various shaping techniques, it is critical to have good metrics to compare the learning rates of the agents. However, in order to compute metrics about how long it takes the agent to learn a problem, it is necessary to make a decision about when the agent has sufficiently learned a good policy for the problem. One common solution is to wait until the stored value of Q-estimates converges, and no changes are made during an episode that are greater than a certain threshold. Because this method can result in the agent learning far after a decent policy has been established, we decided to use a method that was more strictly performance based. Because we know the optimal policy for the grid space, we can determine how many steps the agent needs to take to get to the goal in the ideal case. In order to declare that the agent has learned the problem sufficiently, it needs to get to the goal in four out of the five most recent episodes within 105% of the steps the optimal policy would result in. This so called 80% criteria is a common criteria used by dog trainers [2].

Unfortunately, passing this competence criteria does not guarantee that the agent has a stored policy that leads exactly to the goal. If the agent updates its policy during the last episode before it is declared finished, the update can change the policy to something invalid. In order to make sure this does not happen, we need to explicitly verify the policy before the trial is declared finished. To do this, we simply walk through the policy from the start state without making any policy updates. If the goal state is not reached, then we let the agent continue learning until it passes both the competence metric and the policy is deemed valid.

We opted to focus on measuring performance of the agent using the cumulative number of steps required across all episodes in order for the algorithm to learn a policy according to the 80% criterion described above. There are other metrics we could have examined. For example, we could have measured the number of episodes needed to learn the policy, the quality of the resulting policy, or the CPU time. However, as all of these measures are related in some way and vary highly depending on the particular learning algorithm, there was little insight we could gain beyond what was available using the steps measure.

4. BASELINE PERFORMANCE

To establish a baseline for performance, we ran both algorithms on the open grid. For SARSA, we scaled the grid size from 10x10 to 100x100. As shown in Figure 2, the number of steps taken to learn a policy in accordance with the competence criteria grew rapidly with respect to the size of the state space.

In order for a shaping method to show adequate improvement over the baseline SARSA methodology, the results need to show either a reduction of the growth rate of steps in accordance with the state space size, or cut the learning time by a significant constant factor. Some shaping methods might trade higher or lower numbers of episodes for shorter or longer episodes, so comparing the number of atomic actions taken to learn an adequate policy is a good comparative metric for the shaping methodologies.

For R-MAX, we scaled the grid from 5x5 to 50x50, with the start and goal states in the same positions as in the SARSA example. The results are presented in Figure 3. The number of action steps required to learn the policy grows roughly linearly in the number of states (quadratically in the grid dimension). Because in this case the algorithm only needs a single example to learn the model for a given state-action pair, and since R-MAX can compute an optimal policy as soon as it has a complete model, it makes sense that the sample complexity would scale in this way. As a model-based method, R-MAX performs much better in terms of sample complexity than model-free SARSA. However, it should be noted that R-MAX incurs a significant computational cost, as it must repeatedly plan a new policy. The complexity of computing this policy also scales with the size of the state space, but is dependent on the details of how that policy is computed. While this makes direct comparison with SARSA difficult, here we are concerned with how sample complexity can be improved for an algorithm relative to itself using different training paradigms.

5. POTENTIAL-BASED METHODS

One common approach to shaping behaviors in canines is to use a reward as a lure to instruct the desired behavior. For example, in this "lure-reward" paradigm [3], if you were trying to teach a dog to "heel" (stand at your side, facing forward, shoulders aligned with your knees) you might do so with a treat as a lure. You would begin by placing the treat in front of the dog's nose, and moving it a few inches toward your side. When the dog moves forward to grab it, you give it to her. This is repeated, moving the lure closer to your side until she is in position. This process of providing intermediate rewards for each step towards the ultimate goal, is akin to reward shaping in machine learning [5].

5.1 Potential Bands

Reward schemes based on potentials are a traditional method in reinforcement learning to introduce the idea of shaping behavior. Each state in the state space is associated with a potential value, and the reward given for any action is simply the difference in potentials of the resultant state and the original state. By changing the potential values of states, the characteristics of the agent's learned

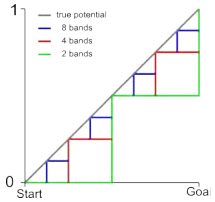


Figure 4: A depiction of various potential bands. As the number of bands grows, the resulting potential function is an increasingly accurate approximation of the true potential.

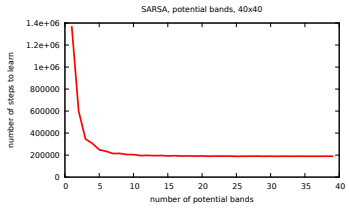


Figure 5: The growth rate of the number of actions required for SARSA to learn the optimal policy on a 40x40 grid, as a function of the number of distinct *potential bands*.

policy can be changed.

A simple way to use potentials for shaping by successive approximation is to group states according to how far away they are from the desired goal state, and then to assign potentials based on how far the group is from the goal. In this way, the groups of states are different approximations of the desired behavior, for which closer and closer approximations are preferred.

We set the potential for the goal state to 1, and the potential for the state farthest from the goal state to zero. We then equally space bands of potentials between those two states. The potential value for each state in that band is:

$$\Phi(state) = 1 - \frac{dist(band, goal)}{dist(start, goal)}, \quad (1)$$

where $dist(x, y)$ is the distance between either a state or band x and another state y. The potential values for the bands thus increase regularly as the states get closer to the goal.

Figure 4 contains an illustration of how "bands" are used to estimate the true potential function. As seen in Figure 5, as one might expect, the more bands of potentials that were used, the faster that the agent was able to learn how to get to the goal state—more potential bands equate to a more accurate approximation of the true potential of each action. Using more bands entailed narrow bands and decreased potential differences between the bands. As the number of bands increases, the average distance that the agent has to move before it gets a reward drops. With all of this extra information, the agent learns much faster. With as few as 10 bands on a 40x40 grid, with a width of each band of 4, the agent using SARSA learns at nearly its maximum pace.

In the case with the maximum number of potential bands, the agent receives a positive reward whenever it moves closer to the goal, and a negative reward any time it moves away. As one would expect, it learns extremely rapidly in this scenario. In the degenerate case with just one band, the problem is identical to the basic case, and the agent learns at the basic rate.

However, these potential bands do not correspond very well with

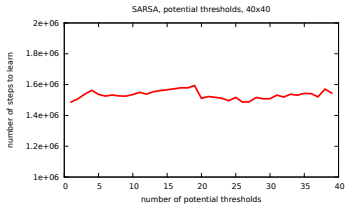


Figure 6: The growth rate of the number of actions required for SARSA to learn the optimal policy on a 40x40 grid, as a function of the number of *potential thresholds* used.

how dogs are generally trained. Dogs generally don't get feedback at every point that it gets closer or farther from the desired behavior. Even in the lure-reward paradigm, the rewards are spaced out over time. As our goal is to investigate how the agents react when canine training techniques are used, it makes sense to try a different method for using potentials to better simulate those techniques.

5.2 Potential Thresholds

In training methods that try to shape by successive approximation, the animal is rewarded when its behavior approximates the desired behavior within a certain "distance". When the animal demonstrates proficiency on behavior within a given approximation, the threshold is made tighter so that the animal has to get even closer to the desired behavior. For example, when teaching a dog to "watch me," they are first asked to look for just a few seconds. Once they are proficient, the duration of maintaining eye contact is expected to increase (a new threshold).

A more accurate way to shape agent behavior by successive approximation using potential shaping is to use a threshold function for the potential value. For simplicity and consistency, the values of our function are zero and one.

$$\Phi(state) = \begin{cases} 0 & : dist(state, goal) > threshold \\ 1 & : dist(state, goal) \leq threshold \end{cases} \quad (2)$$

Once the agent crosses the boundary between the potentials and receives a reward, the episode terminates. Once the current threshold has been learned sufficiently, the criteria is changed and a lower threshold (closer approximation to the desired behavior) is used.

In order to train the agent to reach the goal state, we need to decide when it is appropriate to up the criteria and move the threshold closer to the goal. We use the same criteria inspired by dog training to decide whether or not the agent has sufficiently learned the problem as a whole to decide whether or not the agent has learned the subproblem [2]. In order to move on to the next potential threshold, the agent has to cross the threshold boundary within 105% of the minimum steps four out of the five most recent trials (the 80% rule). Again, because criteria changes are determined during learning (and not in a separate evaluation process) a policy verification step is necessary to make sure a Q-table update during the last trial did not make the policy invalid.

As shown in Figure 6, the performance for the agent using potential thresholds did not change much. The number of steps taken to learn the policy in accordance with the competence criteria was fairly consistent with the base SARSA algorithm (illustrated in Figure 7) regardless of the number of thresholds used. Thus, despite starting with easier tasks and slowly increasing complexity (*i.e.*, by using more thresholds), RL algorithms perform essentially the same. This is because, regardless of the number of potential thresholds, the world must be explored to build the Q-table.

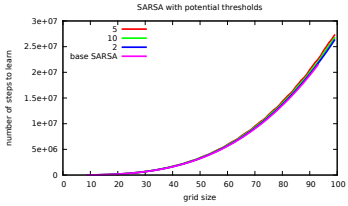


Figure 7: The growth rate of the number of actions required for SARSA to learn the optimal policy, with 2, 5, and 10 *potential thresholds*, as a function of the square grid size.

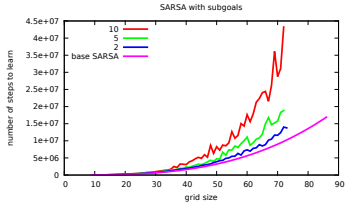


Figure 8: The growth rate in the number of actions required for SARSA to learn the optimal policy for varying numbers of subgoals, as a function of grid size.

6. SUCCESSIVE APPROXIMATIONS

A paradigm that is gaining significant momentum in the dog training world is clicker training [7]. Through the use of a conditioned secondary reinforcer [6] humans are able to more effectively shape the behavior of canines by requiring them to perform increasingly more accurate approximations of the desired behavior. Shaping, conceived of this way, is a process by which a common start state is used (*e.g.*, the dog is in a standing position) and the goal state is moved closer and closer to the ultimate goal (*e.g.*, the dog's behind is closer and closer to the floor for a sit behavior). In this section, we report on a variety of experiments that use this shaping by successive approximations technique.

A straightforward method of performing successive approximation is to introduce the idea of subgoal states. Rather than giving a reward for the agent reaching the ultimate goal state, we give a reward once the agent has reached some state that lies along the optimal trajectory between the start state and the goal state.

For a given optimal trajectory between a start state and a goal state, we choose a number of subgoals spaced evenly between the start state and the goal state.[1] We begin by running episodes starting from the same start state, but where a reward is given and the episode is terminated when the agent reaches the first subgoal.

We determine when the agent has sufficiently learned how to get to the subgoal by the same competence criteria as used in the previous experiments. When the agent has learned how to get to a particular subgoal, the subgoal that is next closest to the goal state becomes the current subgoal and is the only state that is rewarded.

Once the agent has demonstrated that it knows how to get to the final subgoal, the true goal state becomes the state that is rewarded. As in the original problem, the agent must get to the goal state within 105% of the minimum distance to the goal four out of five of the most recent trials and the policy must be verified to lead to the goal in order for the agent to be done. Our primary metric of concern was the total number of steps taken. The total number of episodes used to learn is an interesting measure, but use of the subgoal architecture is likely to result in a shorter average episode length because the start and goal states are closer.

It is worth noting that any time the criteria changes and a new subgoal is selected, the old Q-values become inaccurate because the reward is non-stationary. The Q estimate for *all* actions of states close to the subgoal will be fairly close to the reward value, as they are only a few states away from getting a reward. When the Markov decision process is changed so that the subgoal is farther away, all of those values will need to decrease, rendering the work done to give preference to the action along the optimal trajectory futile.

Even worse, the policy for states that are between the subgoal and the goal state are likely to point back to the previous subgoal even when the subgoal state is moved closer to the goal. Time is wasted when the episode does not terminate when the agent reaches the previous subgoal, and the policy for states closer to the goal pushes the agent backwards. That portion of the policy has to be unlearned before the agent can move on to reaching the subgoals closer to the goal. The data we present support these observations.

6.1 Simple Subgoal Decompositions

In our testing we found that SARSA with optimistic initialization and a fixed exploit policy was frequently unable to learn an optimal policy in the face of non-stationary rewards. Because certain Q-values that should reflect a reward for later subgoals in the decomposition may be learned to have low Q-values during training on earlier subgoals in the decomposition, the learner may have a hard time converging on more accurate higher values when the subgoal criteria changes. In those cases, learning may get stuck. Therefore, to better compare the results of using subgoals to the basic agent, we used an epsilon-greedy action selection policy for the agent that used the simple subgoal decomposition. The agent takes a random, exploratory action 5% of the time, which fixes the issue by allowing the agent to discover the fact that the optimal trajectory has a higher value than the one that is stored.

Figure 8 shows that the performance of agents using the subgoal architecture was consistently worse than agents learning on the base problem. The more subgoals that are included in the architecture of the problem, the longer that the agent takes to complete the problem. The growth rate of the number of steps that the agent takes versus the number of subgoals used is not directly proportional, which points to the fact that there is some benefit gained to having the knowledge of how to get to the previous subgoal over starting from scratch on each subgoal state. However, this benefit is dwarfed by the amount of overhead taken up by the subgoal architecture, either by proving that the agent knows a policy to get to the subgoal well enough, or by relearning parts of the policy that differ once the subgoal is moved.

As seen in Figure 9, similar problems can be seen when using R-MAX with subgoals. To incorporate subgoals into R-MAX, we attempted to consider all subgoals simultaneously. The transition and reward models for the current subgoal are updated to reflect the fact that the episode terminates there, and are set such that any action causes the agent to remain in the subgoal with 0 reward. When any subgoal is reached and it is determined that the current policy is optimal for that subgoal, then the subgoal is removed. When a subgoal is removed, the model is reset to show all actions from that state leading to state G_0 with reward R_{max}. When a transition or reward does not match the current model, the value for

[1]Although omitted from this paper, we found that the spacing of subgoals had no effect on performance. This is actually contrary to what we observe in dog training, where subtask complexity can increase as learning persists. We believe that curriculum design [12] is actually crucial to algorithm performance in the long run.

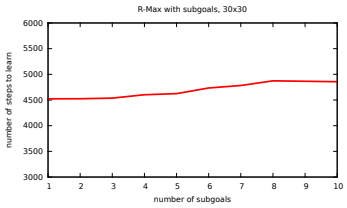


Figure 9: The growth rate of the sample complexity for R-MAX using subgoals, as a function of the number of subgoals

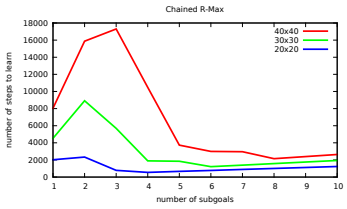


Figure 10: The growth rate of the sample complexity of R-MAX, as a function of the number of subtasks.

that transition is replaced, and the optimal policy is recomputed. This allows the agent to adjust its model to reflect the removal of subgoals. Once the reward for an earlier subgoal was reduced, the new greedy policy would be exploratory until the new (sub)goal was learned. However, because there were still many unexplored states when the subgoal was moved, the exploration policy would not always be efficient at finding the new goal. As a result, it took longer to find each new subgoal, as well as the final goal, than it would have taken without the subgoals.

For the R-MAX algorithm, subgoals lead to unnecessary exploration, and so we attempt to address this issue by decomposing the full task into subtasks which should require less exploration to solve. In this case, we consider moving from one subgoal to the next as distinct subtasks that are learned separately by R-MAX, and the final policy then consists of a chain of policies from the start state, through each subgoal, to the final goal. Each subtask is processed in order going towards the goal, but as is the case for the Q-tables for options, each subtask is learned independently, without sharing transition or reward models. Learning on a subtask completes when an optimal policy for that task has been found.

Figure 10 shows that with a sufficient number of subtasks, that is, with subtasks that are sufficiently small, a lower sample complexity can be achieved. For a smaller number of subtasks, however, the number of steps required actually increases. There is a specific number of subtasks for each grid size at which the sample complexity peaks and then begins to decrease. With a sufficient number of subtasks the total sample complexity of solving all of the subtasks is actually less than that of solving the full task. One likely explanation for this is that once the number of subgoals becomes large enough, and consequently the distance from one subgoal to the next drops below a certain threshold, the optimal policy at the earlier subgoal is to move towards the next subgoal, instead of trying to reach unexplored states, which it assumes yield reward R_{max}, but now incur a greater penalty to reach than the subgoal. Chained R-MAX does show a substantial improvement in performance, in terms of sample complexity, over the base R-MAX. There is, however, a point after which sample complexity begins to increase again as a function of the number of subgoals,

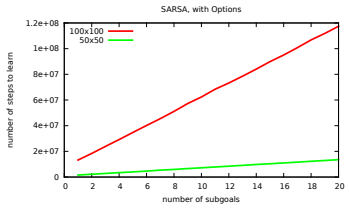


Figure 11: The growth rate of the sample complexity using options, as a function of the number of options.

suggesting that there is still a limit to how much of an improvement can be gained through this method. Further, these gains can only be realized when each subgoal is considered an independent subtask—something that does not model real-world learning.

6.2 Subgoals with Options

In order to avoid the issue of forcing the agent performing SARSA to do extra work by unlearning the Q-estimates near the previous subgoal, we can leverage the options framework [10] to try to improve shaping by successive approximation. We still start with giving a reward to the agent when it reaches the first subgoal. However, when the agent has demonstrated that it knows how to get to the subgoal according to the 80% competence criteria, instead of using the same table of Q-estimates for the following subgoal, we start with a fresh table of Q-estimates and package the previous Q-table as an option available at the start state (including recursively-defined options). The initiation set for the option is just the original start state. The option exits with probability 1.0 upon reaching the subgoal for which the option was built, and probability 0.0 otherwise. The Q-estimate for the option is initialized in the same optimistic manner as for the other actions. The agent is not forced to take the option, but is very likely to learn to do so as the option will follow a near-optimal policy towards the next subgoal.

Because the policy for each subgoal is required to pass policy verification before being declared complete, the option is guaranteed to terminate. It is possible, and very likely, that the policy that is wrapped into an option contains a previous option as part of the policy. As the problem progresses, there could be a recursive stack of options potentially as large as the total number of subgoals. Each of the options on the stack ultimately controls the behavior from its corresponding previous subgoal to its own subgoal state.

As seen in Figure 11, using options in this way increases the sample complexity over the basic algorithm, that is, the case with only 1 option. The problem caused by inconsistency between the optimal policies to reach different subgoals is avoided by using the options architecture. However, there is an additional sample overhead caused by the fact that the agent must learn, with every new subgoal, that it needs to take the provided option over the atomic actions. It seems that this overhead is still much greater than the benefits to be had by shaping the behavior of the agent with subgoals that are easier to reach.

7. LEARNING FROM EASY MISSIONS

Shaping by successive approximation, as described above, is a common tool for teaching dogs simple behaviors (like sit, stay, or down). However, it is also very common when teaching dogs more complex behaviors to use a slightly different approach. Strictly speaking, this other approach is still shaping by successive approximation. However, rather than keeping a fixed start state and varying the goal, a fixed goal is used and the start state is moved increas-

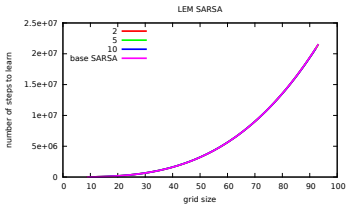


Figure 12: The growth rate of the LEM agent as the grid size scales as compared to the basic agent.

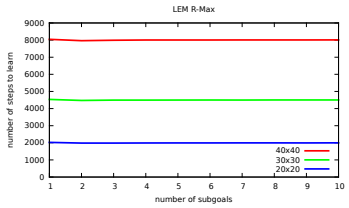


Figure 13: The sample complexity of the LEM R-MAX agent as a function of the number of start states.

ingly "farther" away. This technique is very commonly used when training working dogs (*e.g.*, hunting dogs [2]).

One of the major problems that slows down shaping by successive approximation using simple subgoals is the fact that the Q-estimates that are learned for each subgoal are inconsistent with the $Q^\star$ values for the optimal policy. This results from moving the goal state to perform shaping. Instead of using a complex method such as applying the options framework in order to get around this issue arising from moving subgoals, we can instead keep the goal location fixed and vary the start state (like in hunting dog training). This process is known as "learning from easy missions" in RL [1].

A set of start states are evenly spaced between the ultimate start state and the goal state. The agent starts by performing episodes starting from the start state closest to the goal. Once the agent reaches the 80% competence criteria as before and the policy is verified to lead to the goal, the start state is moved back to the next position. This process is repeated until the agent demonstrates that it knows how to get from the final start state to the goal state.

The major advantage that fixing the goal state has over the basic subgoal method is that the portions of the Q-table that are learned initially according to the early start states do not need to change when the start state is moved away from the goal state. Because the same state is always rewarded, intermediary states' values are unchanged and there is nothing unlearned when subgoals change.

The agent performed strictly better than the agent using the basic subgoals architecture (see Figure 8). However, as seen in Figure 12, like the agent that used potential thresholds, any performance gains over the baseline SARSA were limited. This result held regardless of the size of the state space or the number of start states.

Compared to the agent not using subgoals at all, there is a fair amount of sample overhead that comes with using multiple start states. A speed gain is accomplished when the agent spends more time closer to the goal and thus less time wandering aimlessly far from the goal. The most productive learning can occur just outside the region for which the agent has a good policy to get to the goal, and thus reasonable estimates for the value of the actions. Because the agent is using dynamic programming, the agent is able to expand that region outward to new states when it is on the frontier just outside of the region it knows well. With the start state moving progressively backward, the agent is able to spend a larger percentage of time on the productive frontier learning rapidly. The EVFA algorithm [14] for approximating solutions to MDP's can be said to do something similar to this. It maintains a set of states for which it has an accurate value estimate. The algorithm uses those estimates to expand the set by learning values for nearby states.

R-MAX is well suited to the LEM approach, as neither the transition function nor the reward function change when the start state is moved. In this case, the start state is moved back when an optimal policy from the current start has been found. The same transition and reward models are used for each start state, and the known/unknown parameters are not changed. As seen in Figure 13 LEM does not degrade the performance of R-MAX, it does not improve it either. As the start state is moved back, the algorithm may need to explore states that it did not need to when the start state was closer to the goal, and so the total number of actions taken is the same as if the full task were solved at once.

8. EPISODIC INTERRUPTION

One of the challenging things when training dogs is to deal with the fact that they frequently will lose interest in training. Perhaps they get full and a food reward becomes less enticing. Sometimes, especially when they are puppies, they lose focus focus and wander off. Whatever the cause, it is often necessary as a dog trainer to interrupt an errant dog and start the trial over again. Generally, this is accompanied by a verbal correction so the dog knows its actions were inconsistent with the task. By using this technique, dog trainers are able to focus their training sessions on reinforcing successful trials, thereby reducing the amount of time dogs spend performing irrelevant actions.

One observation about the performance of agents in our state space is that they spend a large portion of their time wandering in the large region away from both the start state and the goal state. The agent using SARSA and the LEM approach on a 50x50 grid space spent, on average, only 7.5% of its steps in states that are within two steps of the optimal trajectory between the start and goal state. The policy near the goal state is well-defined, and once the agent nears the goal state, the episode ends within a short period of time. However, the policy far from the goal state is nearly random, and the agent can spend a lot of time in the distant region. Additionally, little utility is gained from spending time in that region. Temporal difference learning relies on the fact that the stored value for the next state the agent will enter is a reasonable estimate for the future value of the previous state. However, the stored values in this unexplored region are mostly arbitrary, so not much useful learning can actually happen.

An additional observation is that the most rapid learning takes place on the frontier between the region near the goal state for which it has a near-optimal policy, and the region far from the goal state for which it has an arbitrary policy. If the agent is closer to the goal state, it does not improve much on the already near-optimal policy. If the agent is too far from the goal state, it wanders aimlessly without a good way to improve its policy. However, when it is on that frontier, it can expand what it knows about the region near the goal state into the area farther from the goal state.

In order to improve the performance of the algorithm, we can force it to spend more time on the productive frontier than in other regions. One way to do this is to end the episode early if the agent takes more than a certain number of steps. The number of steps is set to some amount greater than the minimum number of steps between the current start state and the goal. If the agent enters

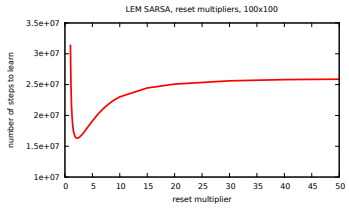


Figure 14: How the parameter determining how long to wait before resetting the trial affects how long the agent takes to learn on a 100x100 grid.

the area near the goal for which the policy is already near-optimal, then the episode will end anyway. However, if the agent enters a region far from the goal state and starts to wander, than it will get cut off before it wastes too much time wandering aimlessly. In the Learning from Easy Missions framework, the current start state is always near, but not in, the area that the agent already has a good policy for. Thus the potential advantage of giving more attempts to learn between the current start state and solidified region is large.

In order to explore the effect of resetting the episode when the agent takes too long, we ran a series of trials on grids of different sizes with different parameters for when the episode was cut off. The episode was cut off when the number of steps taken reached some multiple of the minimum steps necessary to reach the goal from the current start state along the optimal trajectory. This multiple was varied from 1.0 to 10.0.

As in shown in figure Figure 14, the performance of the agent is highly contingent upon when the episode is cut off. If the parameter is set too high and the episode is allowed to go on too long, the performance converges upward to the performance of the agent without cutting episodes off, as one might expect. However, if the parameter is set too low and the agent is cut off to early, performance rapidly deteriorates because the agent has trouble reaching the goal state before the episode terminates.

The optimal parameter for resetting the episode seems to be to reset the episode when the agent takes a number of steps around twice the minimum distance necessary to reach the goal. In this case, the agent takes about two-thirds as many steps as it would without cutting off any episodes early. In addition, on the 50x50 grid space, it spends 17.1% of its steps on average in states within 2 steps of the optimal trajectory versus 7.5% for the agent that does not use episodic interruption. Although this improvement is notable, it is not indicative of a strong change in the agent's ability to solve the problem when subgoals are introduced.

9. CONCLUSION

In this paper, we have surveyed a number of techniques for shaping behavior. We have drawn connections between these topics in the canine training literature and the machine learning literature. We have conducted an empirical analysis of these techniques on a simple, open grid environment where complete policy and performance comparisons can be made. Overwhelmingly, we found that the "behavior" of both model-based and model-free reinforcement learning algorithms did not match our expectations based on our experience training dogs and on relevant dog training literature.

Based on these data, we conclude that shaping techniques are unlikely to yield significant improvements in the performance of standard reinforcement learning algorithms in simple grid (and similar) domains. As these shaping concepts enable dog trainers to train dogs to incredibly high performance, it would be expected that any model of animal learning should see similar gains when such techniques are applied. The fact that similar improvements are not found suggests that standard reinforcement learning algorithms are insufficient as models of animal learning. It may be the case that, for certain domains and state representations, shaping techniques may prove more effective, or it may be that these learning algorithms are fundamentally unsuitable for this form of training. Regardless of which may be true, reinforcement learning—being more a dynamic programming technique than model of learning—may need a paradigm shift if it is to benefit from the training regimens that human trainers use with animal learners.

Future analyses will extend this work to stochastic domains, as well as to domains with continuous or factored state spaces, using function approximation. Additional work will examine different environments and learning tasks to determine what properties of tasks, if any, afford shaping. Future work will also contribute the development of algorithms which are more suitable to shaping.

10. REFERENCES

[1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 223:279–303, 1996.

[2] J. Barry, M. Emmen, and S. Smith. *Positive Gun Dogs: Clicker Training for Sporting Breeds*. Sunshine Books, 2007.

[3] I. Dunbar. *Before and After Getting Your Puppy: The Positive Approach to Raising a Happy, Healthy, and Well-Behaved Dog*. New World Library, 2004.

[4] S. R. Lindsay. *Handbook of Applied Dog Behavior and Training, Procedures and Protocols*, volume 3. Wiley-Blackwell, 2008.

[5] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[6] I. P. Pavlov. *Conditional Reflexes*. Oxford Univ. Press, 1927.

[7] K. Pryor. *Don't Shoot the Dog!: The New Art of Teaching and Training*. Bantam, 1999.

[8] B. F. Skinner. *Science and Human Behavior*. Macmillan, 1953.

[9] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.

[10] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[11] P. Tadepalli and D. Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 100(1-2):177–224, April 1998.

[12] M. E. Taylor. Assisting transfer-enabled machine learning algorithms: Leveraging human knowledge for curriculum design. In *Proceedings of the AAAI 2009 Spring Symposium on Agents that Learn from Human Teachers*, 2009.

[13] E. L. Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review Monograph*, Supplement 2:1–109, 1901.

[14] P. Zang, A. J. Irani, P. Zhou, C. L. Isbell, and A. L. Thomaz. Using training regimens to teach expanding function approximators. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS10)*, pages 341–348, 2010.

Fast Multiagent Learning from Actions Not Taken for Heterogeneous Agents

Carrie Rebhuhn
Oregon State University
rebhuhnc@onid.orst.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

In a cooperative multiagent system, information discovered by one agent may be useful to others. Defining a reward structure that can tap into the information discovered by other agents can promote faster learning through group exploration of the state space. Previous work has shown that collective rewards of this nature, referred to as Actions Not Taken (ANT) rewards, outperform classic reward structures on the Bar Problem in both learning speed and robustness to system changes. However the applicability of ANT rewards is severely limited by the fact that they only consider learning in a system in which agents have identical properties and action spaces. Because most real-world systems do not involve identical agents, we propose a new ANT reward which better accommodates heterogeneous agents using a similarity weighting. In addition, we show the applicability of this approach by extending to more complex domains: (i) a version of the El Farol Bar Problem with agents that have heterogeneous actions and, (ii) a network routing domain. We show that our formulation of ANT for heterogeneous agents shows up to 55% better performance with 30% faster convergence in the network routing domain.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms

Keywords

Multiagent Reinforcement Learning, Congestion Games, Network Routing

1. INTRODUCTION

Reinforcement learning in multiagent systems has broad applications to many domains, including RoboCup Soccer [4], data routing [5, 8], and air traffic control [6]. Learning-based algorithms are in many cases more adaptable to system stochasticity than static algorithms, but careful reward design is imperative to keeping convergence time low and solution quality high. Reward design in multiagent systems frequently must take into account a system-level goal, which does not necessarily maximize the utility of an individual agent. In routing domains, agents which attempt to increase their personal benefit may cause congestion and decrease the performance of the system. In such situations agents implementing personally-optimal policies may actually minimize system performance, a phenomenon known as the Tragedy of the Commons [3].

Difference rewards [8] have been shown to effectively overcome the Tragedy of the Commons while maintaining better convergence speed than standard system-level rewards. However in more complicated systems difference rewards may require resimulation or function approximation in order to capture the full effects of the system. As these methods of reward calculation can be costly, it is crucial for agents to learn quickly from sampled information.

To maximize the usefulness of each reward given within the system, it is possible to use the actions taken by one agent to determine the effect of another agent taking the same action. This method, called 'Actions Not Taken' (ANT) [7], which updates the information of all agents based on the actions of its teammates, results in faster exploration of the state space as compared to local, system, and difference rewards. ANT rewards been shown in the El Farol Bar Problem [2] to converge faster and to recover quickly from system changes. ANT rewards offer a promising method of speeding up learning within a multiagent system, but they hinge on one main premise: the actions of one agent taking an action must have the same effect as if another agent were to take the action in its place.

When agents are homogeneous, as seen in the Bar Problem, it can be assumed that the impact of any agent taking another action would be the same. In most real-world domains, however, agents have different properties and abilities to impact the system. To give an example from RoboCup Soccer, an agent playing the goalkeeper would have different actions available to it than an agent playing as an offensive team member. Even if an offensive player took an action and had a positive effect on the system, the goalkeeper taking the same action could not necessarily expect to have the same positive effect. To extend the applicability of ANT rewards to more complex domains, we introduce a new ANT-based reward which takes into account agent differences to more accurately translate actions not taken. We demonstrate the effectiveness of these ANT rewards on a heterogeneous-agent formulation of the Bar Problem as well as a network routing domain.

The remainder of this paper is structured as follows: Section 2 provides background on reward shaping techniques and difference rewards, as well as shows the previous formulations of ANT rewards and an overview of the network routing problem. Section 3 introduces our testing domains

and their respective objective functions. Section 4 outlines the learning mechanism and reward formulations for both of our testing domains. Section 6 presents the results of the simulations in the bar problem and network routing domain and Section 7 provides a discussion of the general results and potential areas of future work.

2. RELATED WORK

To motivate our reward structure we present alternative methods of reward modification for engineering better system performance, generally referred to as 'reward shaping'. We also present a brief overview of the network routing domain.

2.1 Reward Shaping

In many cases it is too complex to model interactions between agents and mathematically compute the total effect which an agent had on the system. It may be nearly impossible to model these interactions in nondeterministic multiagent settings as in the routing domain, where actions taken by agents have 'downstream' consequences which will affect congestion at a later time period. This is difficult to quantify in a forward fashion. Difference rewards [8] provide a compact encapsulation of the agent's true impact on the system by examining the effects of removing an agent from the system and comparing the result to the system with the agent.

These rewards balance two parameters, factoredness and learnability [1], which define the behavior of a learning system. System rewards are perfectly factored but tend to have too much noise in the signal for an agent to identify its impact on the reward signal. Local reward signals are perfectly learnable in that an agent's actions directly affect the reward, but these are subject to the Tragedy of the Commons effect as described in Section 1. Difference Rewards have proved to outperform global and local rewards in many congestion domains [1, 8, 7, 9]. The difference reward formulation in its general form is:

$$D_i(z) = G(z) - G(z - z_i + c) \quad (1)$$

where z is the system state, z_i is the part of the system state that agent i impacted, and c is some counterfactual which did not depend on the actions of agent i.

2.2 Actions Not Taken Rewards

This work builds directly off of the Actions Not Taken (ANT) concept provided in [7], which introduces the idea that an agent may be able to learn from simulating taking different actions in order to reduce the exploration of the state space. A number of different setups for ANT have previously been explored including exploration of all alternative actions, ANT with early stopping (stopped sampling actions not taken and gave difference reward with zero counterfactual after some training period), ANT with teams (only explored alternative actions taken by team members), and ANT with weighted teams (increased the perceived reliability of the estimate based on how many other agents took the action). In previous work, ANT with weighted teams dominated the other ANT formulations, and was given by the reward:

$$D_{WT}^{i\to b} = G(z - z_i^a + z_i^b) - \mu_{|T^i_{day_{i\to b}}|} \cdot G(z - z^b_{T_i} - z_i^a) \quad (2)$$

for an agent i that takes an action a and compares to an action b and where $G(z - z_i^a + z_i^b)$ refers to the system evaluation of a state in which agent i has taken action b instead of a, $\mu_{|T^i_{day_{i\to b}}|}$ is the average number of team members taking action b, and $G(z - z^b_{T_i} - z_i^a)$ refers to the system evaluation of a state in which agent i and its teammates taking action b are removed. In preliminary testing of $D_{WT}^{i\to b}$, it was discovered that this reward performed better in the case where $\mu = 1.0$, which equated to the D_{Team} reward in [7]. For this reason our reference to 'previous ANT rewards' is to the D_{Team} reward rather than D_{WT}.

2.3 The Network Routing Domain

Network routing is a common real-world problem, covering air traffic control, toolpath scheduling, and Internet routing. Typical commercial solutions in these applications have not included learning, but it nevertheless provides an interesting testing domain. Shortest-path algorithms are popular for network routing, however it has been shown in previous work that in a routing domain that objects being sent on a longer path may be beneficial to the system as a whole as it may alleviate downstream congestion [10].

This result is similar to the bar problem, in which some agents must take suboptimal actions in order to optimize the entire system's performance [9]. Learning algorithms and reward structures such as the difference reward have been applied to the data routing domain, specifically with mobile ad-hoc networks (MANETs)[8], but ANT rewards have never been investigated in this domain. As these represent a system with complex dynamics and real world applications, a reward's success in this domain serves to demonstrate it can be applied effectively to stateful domains with heterogeneous agents.

In this work, we do not provide non-learning popular algorithmic solutions as a comparison metric, as our goal is to use the network routing domain to demonstrate the applicability of ANT rewards to complex systems rather than to challenge present algorithms.

3. EXPERIMENTAL DOMAINS

We explore two experimental domains in order to test our reward formulation. The first is a bar problem featuring a modification such that agents have somewhat different actions available to them. This provides a toy domain on which to demonstrate the success of our method on a system with heterogeneous agents. We then present a network routing domain, which closer models a real-world problem with heterogeneous agents.

3.1 The Bar Problem

The El Farol Bar Problem is a frequently-used abstraction of congestion problems. The problem is set up such that there is an optimal capacity c which provides the most enjoyment for all who attend the bar on a particular night. The local enjoyment of an agent attending the bar is a function of its attendance that night:

$$L_i = e^{\frac{-x_i(z)}{c}} \quad (3)$$

where $x_i(z)$ is the attendance on the night that agent i went to the bar. The total enjoyment of the system is given by a summation of these rewards:

$$G(z) = \sum_{k=0}^{K} x_k(z) e^{-x_k(z)/c} \quad (4)$$

where k is the index of the day within the week, and $x_k(z)$ is the number of people who attend on that night.

Our modification of the bar problem features restricting the action space of agents with a certain probability P_{action} of any action being in its action space. As agent action spaces were generated semi-randomly, this meant that the new bar problem optimal solution was not calculable, and presented a more difficult learning problem for the agents.

3.2 Network Routing

Our network consisted of N nodes connected by L links. Packets had to be routed such that they arrived at their destination within a certain set amount of time. Nodes were modeled as learning agents. Links were generated such that each node linked to another node in a ring, and then links were generated randomly until L links had been created. The network was therefore sparsely connected but guaranteed that any packet could be successfully routed from any starting point to its destination (at worst a packet's optimal path would be travelling $N/2$ hops).

Links were set to have homogeneous delay (synchronous packet delivery) and packets were set to have homogeneous starting attributes. Packets had unit size, and all began with the same time-to-life (TTL) value which defined how many hops a packet would exist in the system before it timed out and was removed. This was decremented after each hop. Packets were also assigned with a random starting node and destination node, which were not identical. Congestion levels in the system were kept constant by random packet generation upon destruction of a packet.

Nodes had queues in which they stored packets waiting to be routed, and had a capacity C. If this capacity was exceeded they dropped the excess packets from their queue. Node states were defined by the attributes of the packet a node was routing, as well as the number of packets in the queue of the neighboring nodes at that timestep (this would change before the packet arrived, but nonetheless gave some indication of congestion at that point).

The system utility was defined as the number of packets delivered over the number of packets handled by all of the nodes in the system per turn, yielding:

$$G(z) = \frac{Delivered}{Handled} \quad (5)$$

where $Delivered$ and $Handled$ represent the total number of packets delivered to their destination and the total number of packets routed in the system. The local reward at a node was given similarly, except on a per-agent basis rather than globally.

4. AGENT LEARNING AND REWARDS

Agents learned in the bar problem using Q-learning with a zero-initialized Q-table which updated according to the stateless Q-update equation:

$$Q_{new}(a) \leftarrow Q_{old}(a) + \alpha(R(a) - Q_{old}(a)) \quad (6)$$

where $Q_{new}(a)$ is the updated Q-value for taking action a, $Q_{old}(a)$ is the previous Q-value for taking action a, α is the learning rate which is bounded by 0 and 1 (0 producing no learning and 1 causing the agent only to consider the most recent reward), and $R(a)$ being the reward received for action a.

In the network routing problem, agents used stateful Q-learning with a zero-initialized Q-table. This updated according to the equation

$$\begin{aligned} Q_{new}(s,a) \quad \leftarrow \quad & Q_{old}(a) + \alpha(R(s_{t+1}) \\ & +\gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q_{old}(s,a)) \end{aligned}$$

where γ refers to the learning discount factor. Actions in both domains were selected using ϵ-greedy action selection.

4.1 ANT in the Bar Problem

We postulate that a weight may exist which expresses the similarity of one action to another action. In this work, we modify the original ANT rewards by incorporating this weighting ($w_{i,j}$) to previous ANT team reward formulations seen in [7]:

$$D_{het}^{i \to b} = G(z - z_i^a + w_{i,j} \cdot z_i^b) - G(z - w_{i,j} \cdot z_{T_i}^b - z_i^a) \quad (7)$$

where $D_{het}^{i \to b}$ is the reward given to each agent simulating reward agent i would have gotten if it had taken action b instead of its own move (action a), z represents the system state, z_i^a represents the part of the state affected by agent i taking action a, and z_i^b represents the part of the state that would have been affected by agent i taking action b.

In many cases, the intersection of the action space between agents may be small, but this does not mean that there is no correlation between the actions of these agents. The analytical solution to maximize the system utility of the bar problem favors an even spread at the capacity of the bars over all nights but one, with one night hosting the 'excess' agents who get negligible enjoyment for their night but allow the maximum enjoyment on the other nights. In this problem some agents in a team may share a single night on which they are both capable of attending. It therefore may be optimal for the 'excess' agents to attend this night preferentially, as it will hold the most agent excess.

However the other nights that they attend have approximately equal value as far as the system reward is concerned. If the 'excess' night is night 1, and there are two agents with action spaces {1, 2, 3} and {1, 4, 5, 6} then for purposes of biasing the results toward faster convergence using ANT rewards the rewards from the first agent's actions {2, 3} may translate to equal the second agent's actions {4, 5, 6} (all of the non-excess actions within the set having roughly equal value). We experiment with this possibility by using a weight to define an action's similarity, which is related to the amount of overlap between the actions of agents:

$$w_{i,j} = \frac{n_{i \cap j}}{n_j} \quad (8)$$

where $w_{i,j}$ is the weighting factor to translate an action found in the set of agent j's actions to an action in the set of actions available to agent i, $n_{i \cap j}$ is the number of actions which are shared by i and j, and n_j is the number of actions available to agent j.

This factor is meant only to translate actions which are seen in agent i's action space but not that of agent j. In the

Domain	Bar Problem	Network Routing
Agents	460	20
Actions/Agent	7, 3.5	5
Capacity	4	15
Teams	10	1
Training Weeks (ANT)	100	100
Statistical Runs	30	30
Packets	NA	100
ϵ	0.1	0.1
α	0.3	0.1
γ	NA	0.9

Table 1: Specifications of the Bar Problem and the Network Routing domains. ϵ, α, and γ values refer to exploration rate, learning rate, and learning discount factor respectively

case that agent i and j share the action in question $w_{i,j} = 1$. In the case where agent i and j share no actions, it is assumed that they are completely independent of one another and the action is not updated. Congestion and learning specifications of the bar problem in this implementation are given in Table 5.

5. ANT IN THE NETWORK ROUTING DOMAIN

The difference reward, which was used for ANT calculation, was determined from this definition of the global reward by combining Equation 5 with Equation 1, giving:

$$D_i(z) = \frac{Delivered}{Handled} - \frac{Delivered - Delivered_i}{Handled - Handled_i} \quad (9)$$

where $Delivered_i$ and $Handled_i$ refer to the total number of packets that node i has individually delivered or handled.

Specifications for the testing values for both the Bar Problem and the network routing domain used can be found in Table 5

6. RESULTS

Team rewards converged to higher values than weighted team, in this experimentation, which differs from the results found in the original work on ANT rewards [7].

6.1 Bar Problem Results

The heterogeneous formulation of the bar problem proved to be more statistically variable than anticipated; the distribution of agent action spaces heavily impacted the optimal solutions available. However, our reward compares favorably against previous ANT rewards in both the original bar problem and the action-restricted bar problem. We show that our heterogeneous formulation performs just as well as D_{Team} in the original bar problem (which is intuitive, as in this case they are mathematically equivalent), while when the action spaces are reduced by approximately half the heterogeneous formulation outperforms D_{team}. The total reward is also decreased, but this is intuitive due to the fact that the optimal solution for the original bar problem becomes impossible due to the limited choices of the agents.

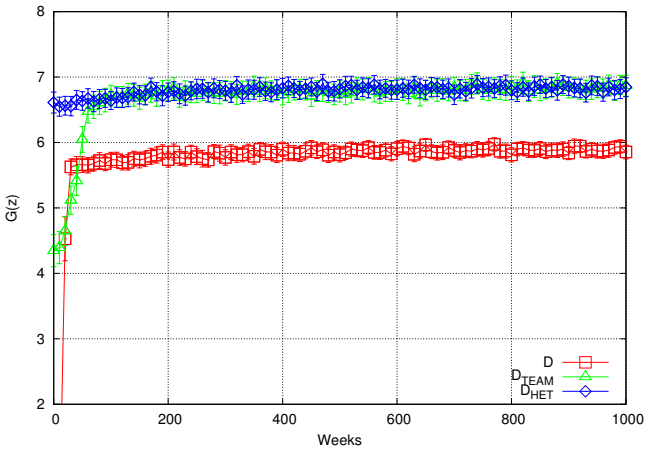


Figure 1: Bar Problem results for difference, team difference, and heterogeneous difference rewards for the original bar problem.

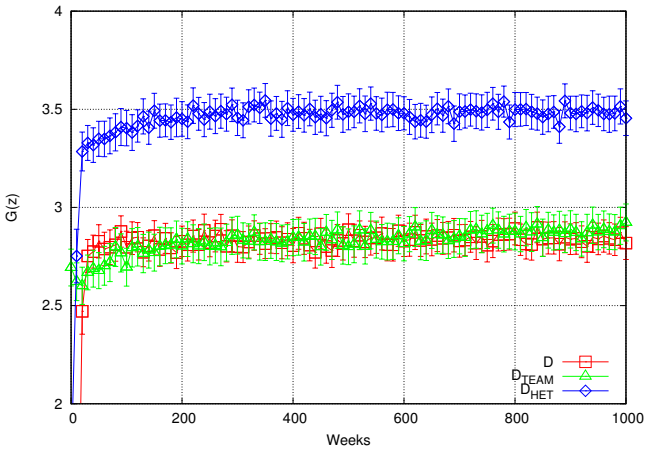


Figure 2: Bar Problem results for difference, team difference, and heterogeneous difference rewards for the bar problem with approximately half the actions available to each agent.

6.2 Network Routing Results

The statistical variability of the performance of the system may be explained by the fact that it is not possible to deliver all packets at once; that is, a large number of packets may be just one hop from their destination at one timestep, and at another timestep there may be no packets which can be delivered in the next hop, through no fault of the agents.

ANT rewards were shown to converge much faster than difference rewards in the network routing domain. As shown in Figure 3 ANT rewards reached their peak in 250 hops with a value of 0.3 whereas difference rewards were still unconverged at a value of 0.18. Moreover, after this peak, the ANT rewards had converged whereas difference rewards remained unconverged after 1000 hops. In Figure 3 it can be seen that the reward trends slightly downward after the peak, but this is attributed more to the variability of the sporadic packet receiving schedule, and it is assumed that

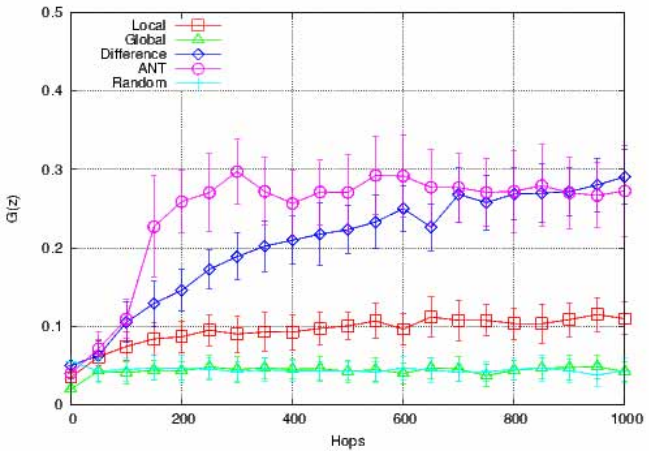


Figure 3: Network Routing results for local, global, difference, ANT, and random action.

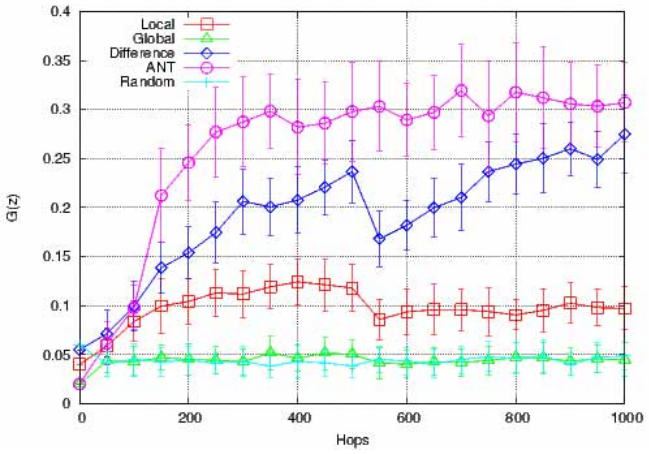


Figure 4: Network Routing results for local, global, difference, ANT, and random action. At 500 timesteps 80/100 links are randomly re-created.

the reward has converged.

One major benefit demonstrated by the results from the link failure testing in Figure 4 is that ANT rewards are more robust than difference reward. After 500 weeks 80 links out of a total of 100 links in the system were destroyed and randomly re-created. In examining the ANT reward it appears that there is very little effect from this link destruction, whereas a clear dip in performance is seen in the graph of the difference reward. This may be attributed to ANT's much faster learning speed than the difference reward, as it can easily recover from perturbations in the domain.

7. DISCUSSION AND FUTURE WORK

ANT rewards show promise in both handling failure in the system (such as in Figure 4) and in learning with much less sampling of the system. In domains in which it is costly to sample, it may be quite valuable to be able to extrapolate information from rewards in this manner.

One main downside of our formulation of our translation weighting is that it represents an action which the agent cannot physically take; if an agent shares 30% of its action space with another agent, it will translate the other agent's action as being 30% there for one night–something which is not possible by the action selection in the bar problem. This means that an agent is comparing itself to imaginary actions. However previous work on the difference rewards has indicated that this is a permissible method of reward calculation [9].

The bar problem and the network routing domain are different in the way they introduce congestion in the system. In the bar problem, an agent may only take a single action, which causes congestion by its presence in the system. In the network routing problem, an agent introduces congestion into the system by sending a number of packets (up to its queue size) to another node. The ANT rewards in the network routing domain were important in that they allowed us to analyze the effects of not only single actions, but chains of actions leading to temporally-dependent congestion in the system. By enforcing a cap on the number of packets in the queue the agents had to learn to route around bottlenecks or risk packet loss and its subsequent performance penalties.

As shown in Figure 3, our modified ANT rewards delivered much faster learning speeds than difference rewards. However their converged performance is not as clearly superior in the network routing domain as in the bar problem. This indicates that a simple weighting may not be enough to capture the effects of replacing another agent. In the bar problem there was little sense of connectedness and chains of related actions, whereas this is one of the aspects which makes the network routing domain very complex, and inhibits calculation of an optimal solution. Further adjustment of training time could also offer faster convergence speed.

A weighting function to translate better between nodes may require a higher level of domain-specific knowledge, whereas our definition of connectivity is simple enough to have broader applications in other domains. Highly complex networks may require a more sophisticated translation function such as a neural network to fully characterize the effects of states and actions of one agent on another. Alternately a statistical sampling may be done to develop better static weights during a preliminary run.

Translation of learning of one node to another node using our weighting is a simple first step, but it provides a foundation for exploiting the similarity between agents to increase learning speed. Significant improvements were seen using our simple formulation, and leveraging this idea of sharing information between heterogeneous actions with a more sophisticated method of translation could amplify these improvements even further. Our results offer motivation for exploration into further application of ANT rewards in conjunction with methods of assessing the relative impacts of heterogeneous agents.

8. REFERENCES

[1] A. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[2] W. B. Arthur. Inductive reasoning, bounded rationality, and the bar problem. *American Economic Review (Papers and Proceedings)*, 84:406–411, 1994.

[3] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, December 1968.

[4] M. Hausknecht and P. Stone. Learning powerful kicks on the aibo ers-7: The quest for a striker. In J. R. del Solar, E. Chown, and P. G. Plöger, editors, *RoboCup-2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Artificial Intelligence*, pages 254–65. Springer Verlag, Berlin, 2011.

[5] S. Kumar and R. Miikkulainen. Confidence based dual reinforcement q-routing: An adaptive on-line routing algorithm. In *16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 758–763, Stockholm, Sweden, 1999. San Francisco, CA: Kaufmann.

[6] K. Tumer and A. K. Agogino. A multiagent approach to managing air traffic flow. *Journal of Autonomous Agents and Multi-Agent Systems*, 2010.

[7] K. Tumer and N. Khani. Learning from actions not taken in multiagent systems. *Advances in Complex Systems (ACS)*, 12(04):455–473, 2009.

[8] D. Wolpert and K. Tumer. Collective intelligence, data routing and braess' paradox. *Journal of Artificial Intelligence Research*, 16(2002):359–387, 2002.

[9] D. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Modeling complexity in economic and social systems*, page 355, 2002.

[10] M. Zhang, R. Batta, and R. Nagi. Modeling of workflow congestion and optimization of flow routing in a manufacturing/warehouse facility. *Management Science*, 55(2):267–280, february 2009.

Multiagent Learning of choices via simpler MDPs and Reward Shaping

Atil Iscen
Oregon State University
iscena@onid.orst.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

In many multiagent learning problems, the complexity can be decreased by exploiting the repetitive nature of the problem. For this reason, the literature of Reinforcement Learning (RL) contains many different hierarchical RL methods that decompose a given task into subtasks that need to be accomplished for a defined goal. On the other hand, some single agent and multiagent problems require to achieve only one of many similar goals. These problems force the agent to choose one subtask instead of accomplishing all subtasks. To exploit this type of problems with choices of repetitive structure, we introduce a method that uses High Level Evaluation of Low Level MDPs (HELM) where a low level MDP is used for each of the subtask choices of the original MDP. Inspired by hierarchical schema, this simpler but parameterized MDP is used both to learn to achieve a subtask and to evaluate different subtask options for the agent. To be able to use the schema in a multiagent setting, we extend the method using a subtask based version of difference rewards, a reward shaping method proven to work for cooperative multiagent systems. The algorithm is tested using the multi-rover problem where rovers cooperatively learn to observe POIs in the environment. The results show that agents which use the combination of HELM and subtask based difference rewards result in significant improvement both on learning speed, and converged policies.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Cooperative Learning, Hierarchical Learning, Reward Shaping

1. INTRODUCTION

Multiagent learning is a growing research area, and the difficulties associated with learning in a dynamic environment become more pronounced as the number of agents increases. In addition to the dynamism, the problem of credit assignment arises between the agents. Because of these problems, autonomous agents that learn to cooperate are studied from many different aspects to improve the collaboration between agents using implicit or explicit coordination methods.

Looking at the nature of the multiagent problems, there can be a repetitive pattern where the agents learn to share resources. In these problems, although the main goal of the agents is to learn to cooperate, they have to deal with the environment using primitive actions. The repetitive routine of the primitive actions used by agents can be easily collected into a hierarchical structure to separate general learning into learning to collaborate and learning the task. For this reason, hierarchical reinforcement learning methods are attractive for these problems. However, the application of hierarchical learning schemas face the challenge of having two layers concurrently learning using one reward.

In this study, we propose a learning method for multiagent problems where agents face choices between similar types of tasks while cooperating with each other. Inspired from hierarchical schemas, High Level Evaluation of Low Level MDPs (HELM) is composed of two layers, but the learning is only happening at the lower level. Higher level decisions are made by evaluating the Q values of the low level learning. The advantages of such a schema are simplifying the problem to learn, giving ability to use a task oriented state representation, and reuse of the knowledge by exploiting the repetitive patterns in the problem.

We first define HELM for single agent problems and then extend it to multiagent problems by coupling it with the reward shaping method "difference rewards" to solve the credit assignment problem. The method is applied on the Continuous Rover Problem where the agents learn to observe Points of Interests (POIs) distributed to the environment. The results show that when coupled with difference rewards, HELM provides a better learning experience for the agents both in terms of learning speed and converged policy. In many different tests, the agents using HELM and the difference rewards outperformed the agents using HELM with other reward schemas and the agents using flat learning with difference rewards.

The rest of the paper is organized as follows: Section 2 gives the required background on multiagent learning and hierarchical learning schemas. Section 3 gives the formal definition of HELM on single agent problems and explains the logic behind it. Section 4 explains the problems with using HELM in multiagent learning problems and gives the solution by integrating difference rewards as a reward shaping method. Section 5 presents the rover problem and explains the application HELM. Section 6 presents and discusses the experimental results, and the paper ends with

Section 7 where the conclusions and future work directions are stated.

2. BACKGROUND

Reinforcement Learning and MDPs

Reinforcement Learning (RL) is a learning method, where an agent learns from its experiences with an environment [18]. In RL, the agent is expected to learn the optimal behavior using the rewards given by the environment as feedback. At a given timestep, the agent gets the state information from the environment, then decides on an action and according to the state and the action of the agent, the environment returns the new state and reward.

The task is defined in terms of Markov Decision Processes (MDPs) specificied by the tuple (S, A, T, R). The agent observes a state $s \in S$ that is a vector of state variables. It decides on an action $a \in A$. Reward function $R : S \times A \rightarrow \mathbb{R}$ maps state and action to reward value. Transition function $T : S \times A \rightarrow S$ maps the state and action to a new state. Given these definitions, the policy of the agent $\pi : S \rightarrow A$ defines which action to take on a given state.

The Q value of a state action pair, denoted by $Q_\pi(s, a)$, is the estimation of the sum of all the rewards that one agent would get by taking action a at state s and following policy π. Through the learning process, the agent improves the estimation of Q values and the policy to increase performance for the given problem. In action value learning algorithms such as q-learning and sarsa, the agents learn Q values assigned to each state action pairs. In the sarsa algorithm, this update is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r_t + \gamma Q(s_{t+1}, at + 1) - Q(s_t, a_t)\}$$

where α is learning rate and γ is discount factor. Using these updates, the Q values converge to discounted values of the best action for the successor state, and the policy becomes selecting the action that gives the best Q value for every state.

In the definitions given previously, the state s is a discrete entity, but in many real world problems, the state variables are continuous which requires discretization. Even when the states are discrete, the number of states can be impossible to handle. Moreover, generalization between similar states can speed up learning significantly. Because of these reasons, RL algorithms are usually coupled with different types of function approximations such as linear function approximations, tile coding or neural networks. [5]. In this paper, the agents use tile coding where the state space is divided into equal sized intervals having separate coefficients to estimate the Q values [18].

Hierarchical Reinforcement Learning

For different problems, the structure can allow different approaches to provide better learning experience for the agent. For problems where the same sequence of actions are repeated, one possible approach is using Hierarchical Reinforcement Learning (HRL) [3] . The general idea in HRL is having an abstraction for repeated action sequences. One approach is defining temporally extended actions that takes more than one timesteps and that can be initiated in particular states [14]. Using this setting, the problem can be defined in terms of semi Markov Decision Processes (SMDPs).

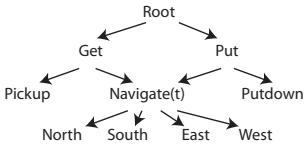


Figure 1: Hierarchical Structure of Taxi Domain used by Dietterich to explain MAXQ method. The taxi has to pick up and drop the passengers. The subtasks have to be completed in order to achieve the goal.

Another approach is using Hierarchically Abstract Machines where an upper level controller uses SMDPs and lower level ones are finite MDPs. In this setting, the upper level controller gives a decision when the lower level controller reaches boundary conditions. Another HRL method is MAX-Q Decomposition by Dietterich [7]. The task is treated as an SMDP and decomposed into subtasks. For each subtask, the state has additional information specifying the parameters and stack of calling subtasks. Subtasks have pseudo-reward functions, and according to execution time and reward accumulated, the Q values propagate to the upper levels of the hierarchy. The literature also contains HRL for cooperative multiagent problems, but again the problem is divided into different subtasks that needs to be accomplished in an order [10].

From the HRL perspective, all these methods introduce a multiple layered structure to learning, but the subtasks are abstractions that need to be accomplished in order to solve the root task. Figure 1 shows the taxi domain example used to explain MAXQ learning. The problem is decomposed into subtasks that are needed to be accomplished in order. The method introduced in this paper similarly decomposes into higher and lower level structures, but the overall approach is significantly different. First, the original task is treated from a different perspective. We specifically work on cases where a task contains multiple choices of similar goal, on the other hand HRL methods handles the execution of successive subtasks. If we explain it using taxi domain example, we are considering the selection between multiple passengers. Second, the higher level decision in our approach does not contain learning. Section 3 gives a detailed explanation about the method.

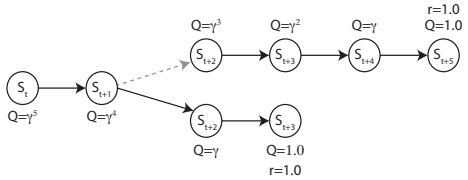
In addition to the HRL methods, the literature contains other task decomposition methods. Mostly developed for multiagent systems, these methods are based on the idea of decomposing the task into smaller tasks for different agents and coordinating the agents to get a more effective assignment schema [15, 16]. Compared to these methods, multiagent usage of HELM serves a similar purpose, but the approach is completely different. HELM is developed first as a single agent algorithm to help an agent learn a task with multiple goals to pick from. The approach is then extended to multiagent problems and coordination via integration of reward shaping (Section 4).

Multiagent Learning and Reward Shaping

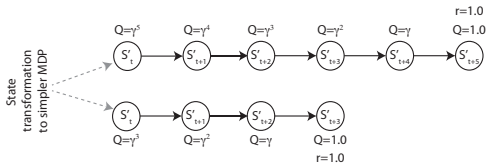
Reinforcement Learning methods are first developed and proven for single agent systems. On the other hand, many problems require or are naturally defined by multiagent systems [20, 17]. A multiagent problem can be defined as an environment containing multiple agents interacting directly

or indirectly through the environment. The task can be competitive, cooperative, or both by having opponents and teammates together. From the learning perspective, a multiagent problem is much harder than a single agent problem. Since each agent's actions effect all the agents in the environment, the environment is highly dynamic. Another dimension of the problem is learning to cooperate or beat other agents. In addition to the environment, agents have to deal with other agents' behaviors that are changing over time by learning.

There are many applications of single agent learning methods to multiagent problems, but the results are not always satisfying. Because of that, the literature contains many different multiagent learning methods [13]. Cooperative problems are subset of multiagent problems where the agents learn to coordinate. The coordination might be explicit such as using communication or coordination graphs, or it can be implicit such as using reward shaping, or teammate modeling. Different Multiagent RL methods and their drawbacks can be found in this detailed survey by Busoniu et al [4]. In addition to these methods, reward shaping methods handle the agents learning independently, and provides them individual rewards such that optimizing the individual rewards will optimize team behavior [12, 6, 8, 11]. As a reward shaping method, Difference Rewards are used and explained in Section 4 where HELM is extended to multiagent problems via integrating the difference rewards [2].



(a) State action flow for traditional RL



(b) State action flow for two level learning schema

Figure 2: Comparison of traditional and two level learning approaches for a problem with two goal states, 3 and 5 steps away. Since the traditional learner propagates the values from the best next state, the current value of the state is γ^3. The two level learner has two different tasks where Q values of the current state are γ^5 and γ^3 resulting in the better choice having a higher Q value.

3. HELM AND SINGLE AGENT LEARNING

Problems containing different choices of tasks can quickly become complex problems for a learning agent. Considering that each option has to be presented in the state representation of the agent, the resulting state space grows exponentially in terms of number of tasks. Moreover, for problems that have many choices between similar goals, giving a decision on a high level action and then choosing a primitive action is more intuitive for a human being.

As explained in Section 2, problems that have repetitive characteristics can be handled using a hierarchical structure in learning mechanism. For problems with multiple tasks to choose from, the most intuitive way of structuring an agent would be using a higher level decision mechanism for the selection of the task and a lower level decision mechanism that decides on the primitive action. Although this schema would help reduce the complexity of the problem, it introduces two levels of concurrent learning. When two levels of concurrent learning are involved, learning can become hard with one reward corresponding to the consequences of two actions taken on different levels.

In a hierarchical schema, if we summarize each level's responsibility, the lower level is responsible for how good each action is for a specific task, and the upper level is responsible for how good each task is in a given situation. The problems can be heterogeneous or homogeneous in terms of tasks. When the agent has a selection of similar tasks it is called homogeneous. For example, in the taxi problem discussed before, picking up one of the 4 passengers in a map is a selection from the same type of task. The tasks are the same, but the parameters or the conditions are different. In such type of problems, all these choices can be handled using one task: picking up a passenger from a given point. At this point, we construct a simpler MDP containing the simple task containing only one goal. As reinforcement learning is used for learning this MDP, the lower level learning mechanism contains a perfect feature that can be used for upper level evaluation: Q values.

The reinforcement learner has a Q table which contains Q values of each state action pair, and these values are mainly used for decision making between different possible actions in a specific state. These values not only signify the best action for that state, they also propagate to the other states which contain a transition to that state. Each of these Q values are actually estimations of the expected sum of the discounted rewards starting from that state and following the current policy. These estimated values are not only beneficial for action selection, they give the agent perfect opportunity to judge how "good" a specific state is. This property results in a Q table where values become higher when the agent is closer to a goal.

If the agent uses flat learning, for the problems which are composed of choices between similar tasks and goals, Q values are propagated from the closest goal state because the algorithm propagates the value of the best state. The Figure 2-a illustrates this propagation at the decision point. In a hierarchical learning case, considering the existence of a learner for a lower level MDP, for each of the tasks, the state of the agent will be transformed to a state specific to the new MDP (Figure 2-b). Q values will be propagated separately instead of merging. This leads to independent Q values for different tasks. Looking at the overall picture, the state given to the agent is transformed into a different state for each different task. For each of these states containing one task, the agent can use the learning with a simpler MDP which only contains one task, and check the Q values that these states give. When the learning for a task is converged to optimal policy, these Q values for each task will increase depending on how close the agent is to that specific task. As RL on MDPs is proven to converge to the optimal policy, and the low level learning is defined on an MDP, low level

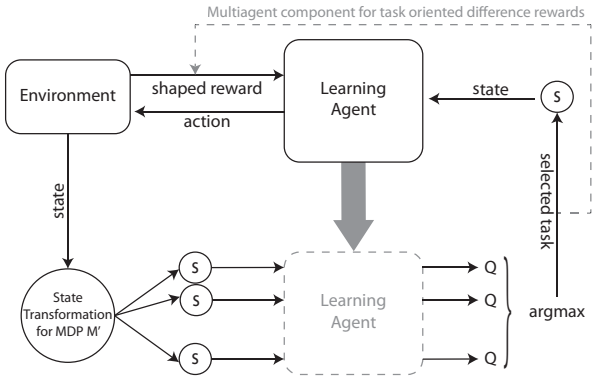


Figure 3: The learning diagram of HELM. The state s is transformed into states s'_i for each task i. These states are evaluated through reinforcement learner for MDP M'. The task giving the maximum value selected so that state s' is given to the agent to learn M'. In a multiagent setting, the selected task is used to shape the reward to provide the agent with difference rewards on a specific task.

will also converge to the optimal policy.

Since the agent now has the ability to evaluate current state for each of the tasks, the decision of which task to choose becomes straightforward: the maximum one. The upper level of the hierarchy becomes selecting the maximum of the potential values for each task. After the task with maximum value is selected, the learning becomes a flat model similar to an MDP containing single task. The agent selects the primitive action, and updates it with the reward given by the environment.

Formally, if we restate the idea described, the agent gets a state s in MDP M containing many tasks. The problem is reduced to a simpler MDP M' that contains only one task. Between M and M' the difference is number of goal states. For every task i, the state s is transformed into state s'_i that belongs to MDP M'. For every task, the current situation is evaluated by collecting Q values of s'_i. On a high level, the task with maximum Q value is selected, and then the problem is treated as MDP M'. Figure 3 illustrates this process. From one perspective, this is a reduction from MDP M to M' using the transformation function that gives the highest Q value in M'. On the other hand, this can also be called as **H**igh level **E**valuation of **L**ow level **M**DP using Q values (**HELM**).

Although the intuition behind the described method is a hierarchical model, the simple selection of the task with the maximum Q value prevents the model from being a true hierarchical model. The method becomes a reduction to an easier MDP which is simpler than a hierarchical model, but still contains most of the benefits of the hierarchical models such as exploiting the repetitive nature of the problem by reusing the previous experience, simplifying the state space by decomposition and simplifying the task to learn.

One great benefit of this model is the ability to use a simpler state representation for the learning agent. Since the problem is transformed into a smaller task, the amount of information that the agent requires is reduced significantly. Instead of learning on a state representation of the complete environment, the state representation that contains information related to the specific task is enough. Another benefit

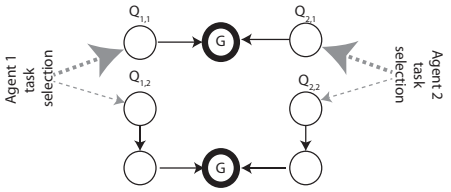


Figure 4: Example of two agents two goals scenario for usage of HELM. $Q_{i,j}$ represents Q value of agent i if selecting task j. It can be seen that, in the ideal case, as the closer goal state Q values will be higher, both agents will select G_1. Since one of the agents will not be able get a good reward, Q values will no longer represent closeness of the goal, which will also damage high level selection of the task in the future.

of the learning model is the reuse of the knowledge. Since the agent uses the same learning mechanism for any task, at every time step, instead of learning a specific task, it learns a generic policy for all the tasks. This greatly reduces learning time of the agent.

4. MULTIAGENT LEARNING USING HELM AND REWARD SHAPING

The HELM method described in the previous section has many benefits over flat learning, but the range of the problems that it is applicable to is limited to single agent problems with choices. One commonly studied subset of learning problems is multiagent problems where there are multiple agents learning a cooperative task. In this specific subset, multiple tasks to choose from becomes multiple tasks to share between the agents. The environment contains many agents and many tasks where the agents learn to collaborate and share the resources.

The multiagent learning problems are harder than the single agent one in many aspects. First, the environment is highly dynamic. Every time step, the resulting state also depends on the other agents. Second, the performance of the system affecting the rewards not only depend on a single agent, it also depends on the other agents. Third, the environment becomes more crowded, and the state space gets exponentially bigger. Moreover, considering the method proposed, probably the most important difference is the nature of the task. From choosing a task and achieving a goal state, it becomes a more complex problem where the agents learn both the tasks and also to collaborate with the other agents.

If we try to apply the HELM directly to a multiagent problem, at every timestep all the agents will be choosing the easiest task according to the Q values of the tasks. Imagine the scenario in the previously described situation with two tasks. If we have two agents at the similar situation (Figure 4), looking at the Q values, they will both learn to pick the easy task. This problem not only breaks collaboration, it will also affect their performance on learning to achieve the low level task, because picking the same goal will result in unexpected rewards for the agents. Since the whole learning mechanism relies on the Q values based on the low level learning, high level task selection will also fail.

Although the multiple agent setting breaks the method completely, the problem faced is nothing more than a credit assignment problem as explained in Section 2. Assuming that the agents could get a perfect reward which takes into

consideration the consequences of the collaboration, it would guarantee a good reward when a low level task is achieved in a cooperative way. The consequences of such a reward will not only stabilize the Q values and bring back the proposed method to life, it will also improve it as a collaboration method. First, it will provide the agents a good enough reward to learn the low level MDP. By learning the low level MDP, Q values would have the incremental flow property that the method exploits. Second, the collaboration aspect of the reward will encourage cooperation as it does with agents using a flat learning method.

As a reward shaping method, the Difference Rewards is well studied and proven reward shaping method that holds the properties that HELM needs [1]. Briefly, it gives an agent a specific reward signifying the effect of the agent on the system. Mathematically, it is defined as the subtraction of the team reward and the team reward without the specific agent. Before explaining difference rewards, let us discuss two natural (unshaped) rewards that can be provided to an agent operating in a team. First, one can provide the "Global" reward (G), which represents full team performance, and second, one can provide a "Local" reward (L) which represents an agent's direct contribution to the global reward. In large systems, providing the global reward becomes problematic as the impact of an agent's actions are obscured by the actions of all the other agents. Similarly, having each agent simply aim to pursue its own contribution does not lead to coordinated behavior.

Based on these observations, the degree of factoredness of a reward defines the proportion of the individual rewards that are aligned with the global reward. This allows an agent to determine if increasing its own reward will also increase the reward of the entire system. On the other hand, learnability measures the impact of the agent's actions on the reward that it gets. Considering the concepts explained above, the local reward is highly learnable but less factored, and global reward is perfectly factored but not highly learnable. To overcome the problems of these two rewards, the Difference Reward (D) was developed to provide a shaped reward that is more learnable than global reward and more factored than local reward. It is defined as:

$$D_i \equiv G(z) - G(z - z_i + c_i) \ , \tag{1}$$

Where first term $G(z)$ is the global reward of the state z, second term $G(z - z_i + c_i)$ is the global reward of the system where the agent i is taken out and is replaced by a default action c_i. Subtraction of the second term from the first term gives the effect of the agent on the system. It is shown to perform better than G and L in many different domains [19, 9].

Since the low level MDP in HELM is a 1 task MDP, the application of the difference reward requires task specific rewards. For that purpose, instead of using difference rewards on the whole environment, we use difference rewards for the new MDP which only contains one task (and 1 goal) of the environment. Figure 3 shows this additional multiagent component with a dashed arrow from the selected task to the shaped reward. Using this information, While shaping the reward with the Equation 1 the function G is replaced by G_i which is global reward of the team for a given task i. We call these modified rewards "task specific difference rewards". Although the shaped reward looks like a new type of reward, it is nothing more than a difference reward for the low level MDP M' instead of MDP M.

The effects of difference rewards on HELM can be further investigated using an example scenario with two agents. Consider a case where two agents have two choices of tasks similar to the example used in single agent scenario (Figure 4). While using the transformation to the simpler MDP, both agents decompose the problem into two tasks. The figure includes reward structures and the propagated values that affect agents' choices. As explained before, global reward results in total confusion to agent, because it also depends on other agent's effect. Not only they have problems selecting the right task, they also have problems learning to achieve a task. Local reward is a perfect signal to learn the low level tasks, but it encourages each agent to go for their own benefit even if it is not beneficial for the team. On the other hand, the difference reward distributes the reward encouraging best action for the team. There is another interesting result in this example, for the choice of task that is not beneficial for the team, the propagated value does not signify how close the agent is to the task, it results in zero. In conclusion, the propagated values signify how close the agents are to the goals but this value is non zero only for the tasks which would be beneficial for the team. Since these values discourage the agents from selecting selfish decisions, the resulting policy gives better results for the team.

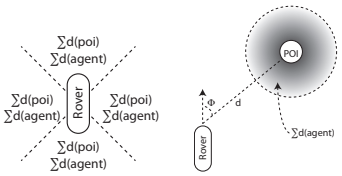
5. APPLICATION TO CONTINUOUS ROVER PROBLEM

The Continuous Rover Problem is a toy domain inspired from the idea of multiple rovers exploring a field containing points of interests (POIs). The domain is previously studied to demonstrate the credit assignment problem and reward shaping methods such as difference rewards and its estimations [19]. Although it is a toy problem, real robot applications and transfer learning from the toy domain to physical robots are also studied [9]. The domain consists of a 2D plane containing Points of Interests (POI) and a team of rovers. The goal of the rovers is to observe the POIs by getting close. The global reward function is defined as:

$$G = \sum_i \sum_j \frac{V_i}{min_i \delta(L_j, L_i)} \tag{2}$$

where V_i is the value of the POI i and L is the location of the POI or agent. $\delta(L_j, L_i)$ is defined as $min\{||x-y||, \delta_{min}\}$ where δ_{min} is the minimum distance required to observe a POI.

When we apply Reinforcement Learning to the Continuous Rover Problem, one has to consider the state representation, action choices and reward functions. As explained earlier in the paper, different types of reward functions such as global, local and difference rewards will be tested. For state representation, although there are many possibilities, the decision was made on previous success of the 4 quadrants (Figure 5-a) [9] . The state is represented with 8 numbers while 4 of them represent POI densities in 4 directions, another 4 represent rover densities in 4 directions. Discretization of the action space is done by defining 3 actions: turning left, keep straight and turn right. Rovers are assumed to move 1 unit in all of the actions, while they change direction. As the state space is continuous, the adaptation to RL is done by using a function approximation method called tile coding. For each variable, the range is divided into 11 tiles,



(a) Classical (b) Task Oriented

Figure 5: State Representations used for the Rover Domain. Classical State Representation consists of 8 variables representing density of Agents and POIs in 4 quadrants. Task oriented state representation consists of the angle ϕ, the distance between rover and POI and density of agents around the POI.

and 4 different tilings. Further details about tile coding and how it works can be found in [18].

The application of HELM to the rover problem is straightforward. The problem contains multiple agents that needs to observe different POIs. Here, the low level MDP is "observing a single POI with existence of other agents in the environment". This new MDP contains all the agents and only the specific POI, as opposed to all the POIs. When the problem is transformed into the low level MDP, it is transformed into different problems with the goal being in different positions. As the task simplifies to observing 1 POI, the state representation can change significantly. Here, we picked a goal oriented state representation that will make the lower level task easier to learn, which will be more precise and indirectly result in better coordination by the nature of the algorithm. For that purpose, the selected state representation is composed of 3 variables: the angle between rover's direction and POI, the distance between rover and the POI and rover density around the POI (Figure 5). Since, the size of the state space is significantly smaller than the previous representation, a more detailed representation can also be used for the task.

6. EXPERIMENTAL RESULTS

As explained in Section 4, given a multiagent problem, using the learning schema proposed and difference rewards together is expected to learn faster and converge to a better policy. To be able to test how well HELM and difference rewards work together, we conducted experiments with various scenarios and number of agents. For every experiment, 20 statistical runs are made to show consistency of the results. For all of the experiments calculated standard errors are small enough that they are neglected on the resulting graphs.

The first experiment is on a small environment (50 by 50) with 4 agents and 4 POIs. 4 POIs are distributed to the corners and the agents are trying to observe as much as possible during episodes of 300 timesteps. The ideal behavior is having 1 rover per POI to observe. As Figure 6 shows, HELM combined with difference rewards (D) outperforms the rest in terms of learning speed. With a simpler task to learn, the agents have almost instant peak for convergence compared to flat learning methods. The benefit on learning speed comes from the partially hierarchical schema of the decomposition. As the information learned is reused within the low level learning, the agent basically learns the domain

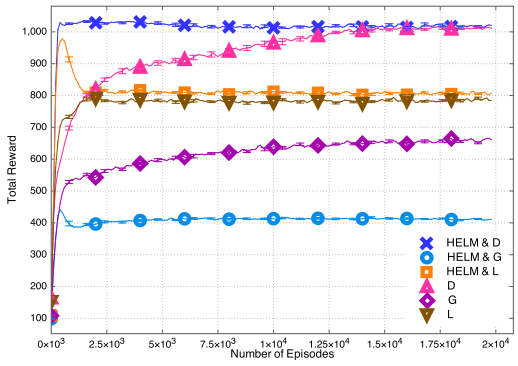


Figure 6: Results of HELM coupled with different reward functions on an environment containing 4 agents and 4 POIs that are spread out to different corners. D, G and L represent Difference, Global and Local Rewards respectively. The results show that HELM coupled with D has almost an instant learning experience and converges to the same point as the agents using flat learning and D

when it learns the simpler task resulted by the decomposition.

On the other hand, HELM reaches a good policy only when it is combined with the difference rewards method. As explained earlier, when the low level task is not learned correctly, the theory behind using HELM on multiagent problems collapses, because global reward is not good enough even to learn the low level task. When the low level task cannot be learned high level selection is almost random, which causes more confusion to the agent. This can be clearly seen looking at the results of decomposition coupled with global rewards (G).The agents using HELM and global reward even performs worse than flat learning with G. Comparing flat learning methods with different types of rewards, the results are as expected. When the agents get shaped reward D, they outperform agents using global and local rewards.

The second experiment is relatively more complicated with a larger environment and 4 POIs concentrated at the same corner. Using this setup, we can see the effects of the more precise and goal oriented state representation that could be used after transformation to the new MDP. In this setting, the difference between HELM & D and others becomes more clear. Figure 7 shows that when HELM is coupled with difference rewards it outperforms both in learning time and converged policy. The difference in converged policy mainly comes from the abilities of the goal oriented state representation that is simpler and more precise than representing the whole environment (Figure 5-a vs Figure 5-b). The performance of the agents using HELM model and local reward is also related to the state representation and small number of agents in the environment. Although both rewards (local and difference) provide learning for the low level task, local rewards do not encourage cooperation. As the number of agents will be higher, the difference caused by the reward structures will be higher. The rest of the graph shows that coupling HELM with global reward or using flat learning with other types of rewards results in worse performance as expected.

When the number of agents is increased to 12 and 24, the Figures 8 and 9 show the consistent performance of coupling

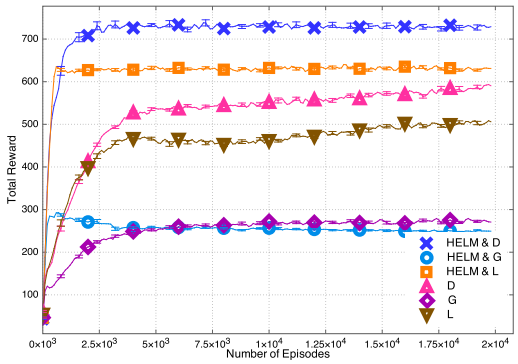


Figure 7: Results of HELM coupled with different reward functions on an environment containing 4 agents and 4 POIs which are concentrated in one corner. With a harder problem to learn, the difference between the methods becomes bigger. HELM & D not only converges faster, it also converges to a better policy than other methods.

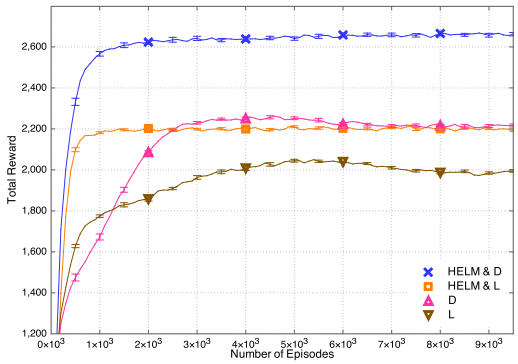


Figure 8: Results of HELM coupled with different reward functions on an environment containing 12 agents and 12 POIs with uniform distribution. In a crowded environment that requires more coordination, HELM & D gives an increase of 20% on overall performance of the agents

HELM with difference rewards. Again, both the converged policy and learning time is better compared to other combinations. Note that these graphs do not contain results for global reward as the performance was notably below the other four mehods.

Comparing the agents that use HELM to flat learners, it can be seen that time to converge is around $1.5*10^3$ episodes instead of $3 * 10^3$. From the learning time aspect, the time requires to converge is less than half in all of the scenarios, and for smaller problems learning can almost be considered as instant (Figure 6). For a multiagent learning algorithm, this is a significant improvement.

It is shown that the converged behavior can score around 400 points higher than flat learning. Considering that flat learning scores around 2200, this gives an increase around 20% on top of the initial performance. Considering the nature of the problem, the difference can be critical in a real mars rover observation mission. We also analyzed the results from another perspective. Table 1 shows how many POIs are being observed at the end of each episode. Al-

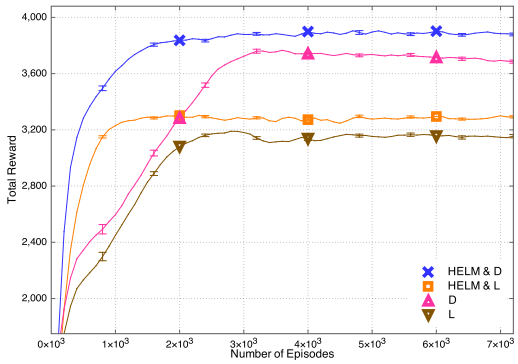


Figure 9: Results of HELM coupled with different reward functions on a larger environment containing 24 agents and 24 POIs that are concentrated in one area of the environment

	12 POIs, 12 Agents			24 POIs, 24 Agents		
	D	G	L	D	G	L
HELM	11.9	9.8	11.0	20.5	13.5	18.0
Flat	11.7	11.0	11.4	18.0	15.0	16.5

Table 1: Number of POIs observed for different learning algorithms and rewards. There is a significant improvement when the agents use HELM and D.

though the agents learn to increase amount of observation, it can be seen that HELM and D observes more POIs than any other combinations. Moreover, the difference is 2 POIs when there are 24 POIs and 24 agents in the system.

The experiments tested the proposed algorithm in a rover domain with different settings. With different sizes of the environment, different POI distributions, varying number of agents, and different reward structures, the results agreed with the claims made in the Section 4. When coupled together, decomposition and task oriented difference rewards results in agents learning faster and better.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we first introduced HELM as a single agent Reinforcement Learning method by transforming the problem with multiple tasks into a simpler problem with one task. The selection of the different tasks are evaluated through Q values in the simpler task. Although the proposed schema looks similar to a hierarchical learning method, the learning happens only at the bottom layer which is responsible for low level MDP. The method is first introduced and explained on single agent multiple task problems. The benefits of using HELM were simplifying the problem to learn, allowing a task oriented state representation, reuse of the knowledge and exploiting propagation properties of Reinforcement Learning by using Q values as a metric for high level choice.

For multiagent problems, we explained that unless low level learning can be established, upper level coordination mechanism would not work. To be able to establish both coordination and low level learning, we integrated reward shaping to the method by introducing a task oriented version of the difference rewards. With the reward shaping introduced, it is explained that the agents can learn low

level MDP. In addition, using the credit assignment aspect of difference rewards, propagated Q values encourages cooperation, transforming the method into a framework suitable for learning to cooperate.

The method proposed is studied on the Rover Domain that is previously used for both reward shaping and transfer learning to physical robots. Transformation into a simpler, one task problem allowed a task oriented state representation that is simpler and more precise. The new MDP was learning to observe a single POI in existence of other agents in the environment. The method is tested in various scenarios with different numbers of agents to cooperate with. The experimental results showed that agents using HELM with difference rewards results in significant improvement on converged policy and significant decrease on time to learn. From the results presented, it can be concluded that HELM can be a great framework for multiagent learning as long as the agent is provided with a good agent specific and task oriented reward signal such as difference rewards.

We are planning to improve the proposed method in many different ways. First, by including reward shaping, the method is used as an implicit coordination method, but it can be improved with communication and explicit coordination methods. As the agents learn lower level tasks, and use the Q values to evaluate different tasks, the agents have a brief important information about each task to choose from. At that point, HELM chooses to pick the task with maximum value, but using an explicit coordination by exchange of these values, we are investigating ways to further improve the coordination between agents. Using such communication will be less costly than communicating whole state information.

Second, the method can be extended for scenarios where the tasks in the environment cannot be completely separated from each other. In that case, tasks with dependencies would gain another dimension to the coordination problem. If these dependencies can be included in the state representations of each task, we believe that the method can be adapted to a broader range of problems. For example, partially observable domains is a class of problems that we have not tested our algorithm in, however to the best of our knowledge, with the reward shaping and task oriented state representations, the method should be able to adapt itself to partially observability; which will be investigated as a further research topic.

8. REFERENCES

[1] A. Agogino and K. Tumer. Multiagent reward analysis for learning in noisy domains. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, Utrecht, Netherlands, July 2005.

[2] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17:320–338, October 2008.

[3] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:341–379, October 2003.

[4] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, Mar. 2008.

[5] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida, 2010.

[6] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 225–232, Richland, SC, 2011.

[7] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13:227–303, November 2000.

[8] M. Grzes and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, volume 2, pages 10–22 –10–29, sept. 2008.

[9] M. Knudson and K. Tumer. Adaptive navigation for autonomous robots. *Robot. Auton. Syst.*, 59:410–420, June 2011.

[10] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 246–253, New York, NY, USA, 2001. ACM.

[11] B. Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 601–608, New York, NY, USA, 2007. ACM.

[12] M. J. Mataric. Reward functions for accelerated learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189. Morgan Kaufmann, 1994.

[13] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and MultiAgent Systems*, 11(3):387–434, 2005.

[14] D. Precup, R. S. Sutton, and S. P. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *Proceedings of the 10th European Conference on Machine Learning*, pages 382–393, London, UK, 1998. Springer-Verlag.

[15] S. Proper and P. Tadepalli. Solving multiagent assignment markov decision processes. AAMAS '09, pages 681–688, Richland, SC, 2009.

[16] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.

[17] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[18] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[19] K. Tumer and A. K. Agogino. Multiagent learning for black box system reward functions. *Advances in Complex Systems*, 12:475–492, 2009.

[20] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1st edition, 2000.