Local Coordination in Online Distributed Constraint Optimization Problems

Tim Brys, Yann-Michaël De Hauwere, Ann Nowé, and Peter Vrancx

Computational Modeling Lab - Vrije Universiteit Brussel,
Pleinlaan 2, B-1050 Brussels, BELGIUM
{timbrys,ydehauwe,anowe,pvrancx}@vub.ac.be,
WWW home page: http://como.vub.ac.be [*]

Abstract. In cooperative multi-agent systems, group performance often depends more on the interactions between team members, rather than on the performance of any individual agent. Hence, coordination among agents is essential to optimize the group strategy. One solution which is common in the literature is to let the agents learn in a joint action space. Joint Action Learning (JAL) enables agents to explicitly take into account the actions of other agents, but has the significant drawback that the action space in which the agents must learn scales exponentially in the number of agents. Local coordination is a way for a team to coordinate while keeping communication and computational complexity low. It allows the exploitation of a specific dependency structure underlying the problem, such as tight couplings between specific agents. In this paper we investigate a novel approach to local coordination, in which agents learn this dependency structure, resulting in coordination which is beneficial to the group performance. We evaluate our approach in the context of online distributed constraint optimization problems.

1 Introduction

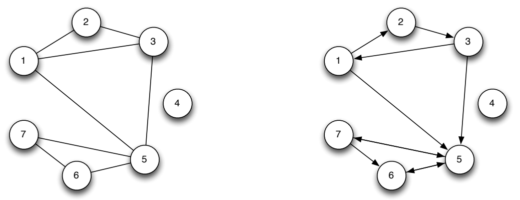
A key issue in multi-agent learning is ensuring that agents coordinate their individual decisions in order to reach a jointly optimal payoff. A common approach is to let the agents learn in the joint action space. Joint Action Learning (JAL) enables agents to explicitly take into account the actions of other agents, but has the significant drawback that the action space in which the agents must learn scales exponentially in the number of agents [5], quickly becoming computationally unmanageable. In this paper, we investigate a novel approach in which agents adaptively determine when coordination is beneficial. We introduce *Local Joint Action Learners* (LJAL) which specifically learn to coordinate their action selection only when necessary, in order to improve the global payoff, and evaluate our approach in the context of distributed constraint optimization. We investigate teamwork among a group of agents attempting to

[*] M. Cossentino et al. (Eds.):
EUMAS 2011 Selected and Revised Papers, LNAI 7541, pp.31-47, 2012
©Springer-Verlag Berlin Heidelberg 2012

optimize a set of constraints in an online fashion. Agents learn how to coordinate their actions using only a global reward signal resulting from the actions of the entire group of agents.
The remainder of this paper is laid out as follows: in the next section we review some background material and related work on agent coordination. Section 3 introduces our local coordination method. Section 4 introduces the optimization problems we consider in this work. We demonstrate how optimization problems can have an inherent structure that can be exploited by LJALs. In Section 5, we propose and evaluate a method that allows LJALs to learn a coordination structure optimized for the specific problem task at hand. Finally, we offer some concluding remarks in Section 6.

2 Background and Related Work

The Local Joint Action Learner (LJAL) approach proposed below relies on the concept of a Coordination Graph (CG) [6], which describes action dependencies among agents. Coordination graphs formalize the way agents coordinate their actions. In a CG, vertices represent agents, and edges between two agents indicate a coordination dependency between these agents. Figure 1(a) is an example of a CG with 7 agents. In this graph, agent 1 coordinates with agents 2, 3 and 5; agent 4 does not coordinate and thus corresponds to an independent learner; and agent 6 coordinates with agents 5 and 7. Figure 1(a) represents an undirected CG where both agents connected by an edge explicitly coordinate. A CG can also be directed, as shown in Figure 1(b). In this graph, the same agents are connected as in Figure 1(a), but the edges are directed and the meaning of the graph thus differs. In Figure 1(b), agent 1 now coordinates with agents 2 and 5, but not with 3; agent 4 is still an independent learner; and agent 6 only coordinates with 5.



(a) Undirected Coordination Graph

(b) Directed Coordination Graph

Fig. 1. Two coordination graphs with 7 agents

Guestrin [6] and Kok and Vlassis [8] propose algorithms where agents, using a message passing scheme based on a CG, calculate a global joint action by communicating their perceived local rewards. Below we describe a new approach which is an alternative to Independent Learning (IL) and Joint Action Learning (JAL) [5] based on CGs, where agents optimize their local joint actions without extensive communication, using global reward.

3 Local Joint Action Learners

We now introduce our Local Joint Action Learner (LJAL) framework. LJALs are a generalization of the Joint Action Learners proposed in [5]. The main idea is that agents keep estimates of expected rewards, not just for their own actions, but for combinations of actions of multiple agents. Contrary to the JALs, however, LJALs do not coordinate over the joint actions of all agents, but rather coordinate with a specific subset of all agents. An LJAL relies on a coordination graph to encode coordination, and will keep estimates only for the combinations of its own actions with those of its direct neighbors in the graph.

It can easily be seen that LJALs cover the entire range of possible coordination settings from Independent Learning (IL) agents, who only consider their own actions, to Joint Action Learners (JAL), who take into account the actions of all agents. As LJALs keep estimates for joint actions with their neighbours in the graph, ILs can be represented with a fully disconnected graph, whereas the coordination between JALs can be represented with a fully connected or complete graph.

Figure 2 illustrates the CGs for ILs and JALs, as well as showing another possible LJAL graph. Note that this representation is not directly related to the underlying structure of the problem being solved, but rather represents the solution method being used. In the experiments below, we will evaluate the effect of matching the CG to the problem structure on the performance, this in terms of learning speed and final performance.

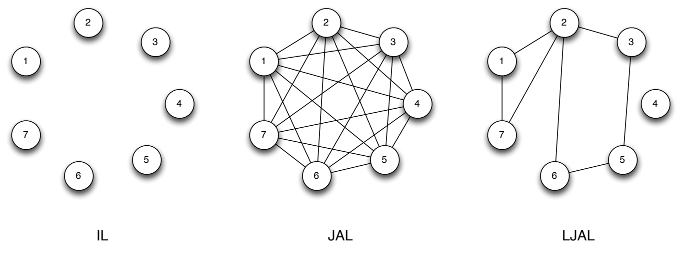


Fig. 2. Coordination graphs for independent learners and joint action learners and an example graph for local joint action learners

3.1 Action selection

We view the learning problem as a distributed n-armed bandit problem, where every agent must individually decide which of n actions to execute and the reward depends on the combination of all chosen actions. In the case that the reward for each agent is generated by the same function, the game is said to be **cooperative**. It is with such cooperative or coordination games that we are concerned in this paper. Below, we describe the action estimation and action selection method used by LJALs.
Each agent estimates rewards for (possibly joint) action a according to following incremental update formula [11]:

$$Q_{t+1}(a) = Q_t(a) + \alpha \left[r(t+1) - Q_t(a) \right] \tag{1}$$

where α is the step-size parameter, balancing the importance of recent and past rewards, and $r(t)$ is the reward received for action a at time t. (L)JALs also keep a probabilistic model of the other agents' action selection, by using empirical distributions, i.e. counting the number of times C each action has been chosen by each agent. Agent i maintains the frequency $F^i_{a_j}$, that agent j selects action a_j from its action set A_j:

$$F^i_{a_j} = \frac{C^j_{a_j}}{\sum_{b_j \in A_j} C^j_{b_j}} \tag{2}$$

Using their estimates for joint actions and their probabilistic models of other agents' action selection, agents can evaluate the expected value for selecting a specific action from their individual action set:

$$EV(a_i) = \sum_{\mathbf{a} \in \mathbb{A}^i} Q(\mathbf{a} \cup \{a_i\}) \prod_j F^i_{\mathbf{a}[j]}, \tag{3}$$

where $\mathbb{A}^i = \times_{j \in N(i)} A_j$ and $N(i)$ represents the set of neighbors of agent i in the CG. This means that the expected value for playing a specific action, is the average reward of the observed joint actions in which the action occurs, weighted by their relative frequencies.

Agents choose their actions probabilistically according to a Boltzmann distribution over the current estimates EV of their actions [11]. The probability of agent i selecting action a_i, at time t is given by:

$$Pr(a_i) = \frac{e^{EV(a_i)/\tau}}{\sum_{b_i=1}^{n} e^{EV(b_i)/\tau}} \tag{4}$$

The parameter τ is called the temperature and expresses how greedy the actions are being selected. Low values for τ represent a more greedy action selection mechanism.

3.2 LJAL performance

In this section, we briefly evaluate empirically how different types of LJALs relate to each other in terms of solution quality and computation speed. Specifically, we will evaluate the effect of increased graph density on performance; it results in more information, but also higher complexity for agents. Intuitively, we expect that ILs and JALs will lie at extreme ends of the performance spectrum that LJALs encompass. ILs possess little information and thus should yield the worst solutions, while JALs, who in theory have all possible information, should find the best solutions. On the other hand, JALs need to deal with the total complexity of the problem, resulting in long computation times, while ILs only reason about themselves and should logically compute fastest of all LJALs.

We compare respectively ILs, LJALs using randomly generated, directed CGs with an out-degree of 2 for each agent, random LJALs with out-degree 3, and JALs, see Figure 3. These types of learners were evaluated on randomly generated distributed bandit problems, i.e. for each possible joint action of the team, a fixed global reward is drawn from a normal distribution $\mathcal{N}(0, 50)$ ($50 = 10 \times \#$ agents). A single run of the experiment consists of 200 iterations, also referred to as plays, in which 5 agents choose between 4 actions, and receive a reward for the global joint action, as determined by the problem. Every run, LJAL-2 and LJAL-3 get a new random graph with the specified out-degree. All learners employ softmax action selection with temperature function $\tau = 1000 \times 0.94^{play}$. Figure 4 displays the results of this experiment averaged over 10000 runs and Table 1 shows the speed (running time needed to complete the experiment) and solution quality for the various learners, relative those of the JALs.

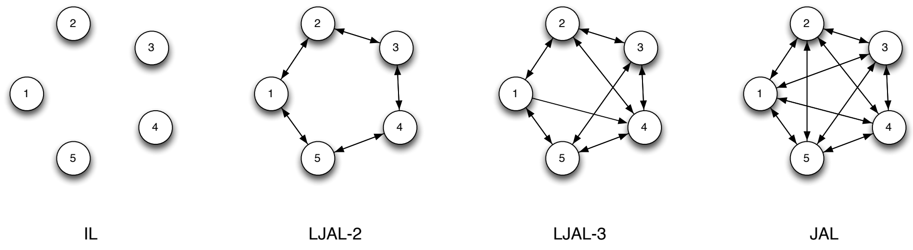


Fig. 3. Coordination graphs for independent learners and joint action learners, and examples of random coordination graphs for local joint action learners with out-degrees 2 and 3.

These results corroborate our hypothesis that ILs and JALs are both ends of the LJAL performance spectrum. Since any LJAL possesses no more information than JALs and no less than ILs, their solution quality lies in between these two extreme approaches. Moreover, because the complexity of LJAL joint actions lies in between ILs and JALs, we also observe that LJALs perform computationally no faster than ILs and no slower than JALs. As expected, as the complexity of the CG used increases, so does the solution quality, but at the cost of a longer computation time.

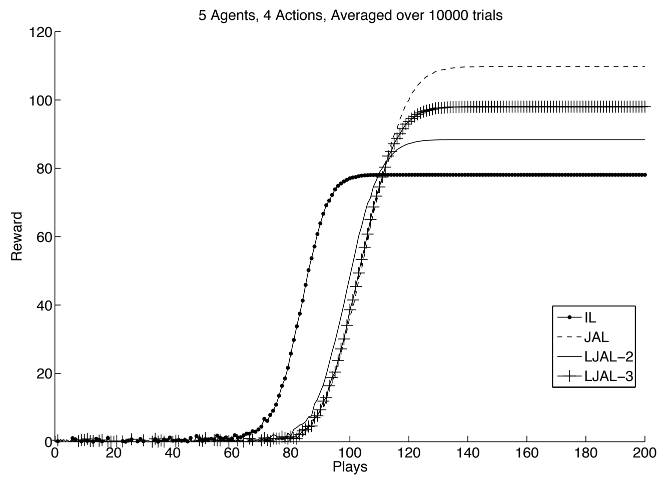


Fig. 4. Comparison of independent learners, joint action learners and local joint action learners on a typical distributed bandit problem.

Learner	Avg # partners	Speed	Solution Quality
IL	0	$\times 31.5$	71.1%
LJAL-2	2	$\times 12.1$	80.5%
LJAL-3	3	$\times 4.4$	89.3%
JAL	4	$\times 1$	100%

Table 1. Comparison of speed and solution quality for independent learners, joint action learners and local joint action learners solving a typical distributed bandit problem. All differences are significant, $p < 0.05$.

4 Distributed Constraint optimization

In the previous section, we have shown that it is possible to use our proposed local coordination method to balance the trade-off between solution quality and computation speed, a problem often encountered in real settings. In this section, we take this a step further, as we aim to show that we can exploit a problem's structure using local coordination, reducing computational complexity, but minimizing the corresponding loss in solution quality. Since the simple bandit problem of the previous section does not have such a structure, as the reward for every joint action is generated independently, we look at another type of problem, which is ideally suited to represent problems with an inherent structure, i.e. Distributed Constraint Optimization Problems.
A Constraint Optimization Problem (COP) describes the problem of assigning values to a set of variables, subject to a number of soft constraints. Each constraint takes the form of a function assigning rewards to variable assignments. A solution to a constraint optimization prob-

lem assigns a value to each variable and has an associated total reward, which is the sum of the rewards for every constraint. Solving a COP means maximizing this reward. A Distributed Constraint Optimization Problem (DCOP) describes the distributed equivalent of constraint optimization. A group of agents must solve a COP in a distributed way, each agent controlling a subset of the variables in the problem.
Formally, a DCOP is a tuple ($\mathcal{A}$, $\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, f), where:

- $\mathcal{A} = \{a_1, a_2, ..., a_\ell\}$, the set of agents.
- $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, the set of variables.
- $\mathcal{D} = \{D_1, D_2, ..., D_n\}$, the set of domains. Variable x_i can be assigned values from the finite domain D_i.
- $\mathcal{C} = \{c_1, c_2, ..., c_m\}$, the set of constraints. Constraint c_i is a function $D_a \times D_b \times ... \times D_k \to \mathbb{R}$, with $\{a, b, \ldots, k\} \subseteq \{1, \ldots, n\}$, projecting the domains of a subset of variables onto a real number, being the reward.
- $f : \mathcal{X} \to \mathcal{A}$, a function mapping variables onto a single agent.

The total reward of a variable assignment S, assigning value $v(x_i) \in D_i$ to variable x_i, is:

$$C(S) = \sum_{i=1}^{m} c_i(v(x_a), \ldots, v(x_k)) \tag{5}$$

For simplicity, we assume only one variable per agent and only binary constraints. Unary constraints can easily be added and higher arity constraints can be constructed using unary and binary constraints.
Distributed Constraint Problems are used to model a variety of real problems, ranging from disaster response scenarios [2] and distributed sensor network management [7], to traffic management in congested networks [9].

4.1 Relation of LJAL to other DCOP algorithms

As noted in [12], a DCOP can be reformulated as a distributed n-armed bandit problem. Assign one variable to each agent and let it choose from the values in the domain corresponding to the variable as it would select an arm from an n-armed bandit. With such a formulation, we can apply our previously described learners to DCOPs. In this section, we briefly evaluate the relation of LJAL to other DCOP algorithms and in which context LJALs are best applied.
Comparing LJAL to the unifying DCOP algorithm framework proposed by Chapman et al. in [3], we see that it relates most to the "local iterative, approximate best response algorithms". Algorithms in this class are incomplete – they are not guaranteed to find the optimal solution –, but on the other hand, they only use local information, having neighbouring agents communicate only their state, and thus do not suffer from exponential complexity in the size of the problem. These algorithms typically converge to local optima, or Nash equilibria, and are often preferred in real-world settings, as these require a balance between solution quality and computational complexity, or timeliness, and communication overhead. In contrast, "distributed complete algorithms", such as ADOPT

[1] are proven to find the optimal solution for a DCOP, although with an exponential communication or computational complexity[4, 10].
We are not specifically interested in developing a state-of-the-art DCOP solver, but rather a multi-agent reinforcement learning technique which can trade-off solution quality and complexity, taking advantage of a problem's structure. Therefore, we explore solving DCOPs in an online reinforcement learning scenario. This means that agents do not have any prior knowledge of the reward function and must sample actions in order to solve the problem. In conventional DCOP settings, local reward functions are assumed to be deterministic and available to the agent. As such the problem can be treated as a distributed planning problem. In our setting, the rewards associated with constraints can be stochastic and agents may have few opportunities to sample rewards. Moreover, the agents cannot directly observe the local rewards resulting from their actions, but only receive the global reward resulting from the joint action of all agents.
Finally, and most importantly, we do not assume knowledge of the constraint graph underlying the problem is always available, an assumption found all over the literature, and often not justifiable in real-world settings.

4.2 Experiments

Since each constraint in a DCOP has its own reward function and the total reward for a solution is simply the sum of all rewards, some constraints can have a larger impact on the solution quality than others, i.e. when there is a higher variance in their rewards. Therefore, coordination between specific agents can be more important than between others. In this section, we will investigate the performance of LJALs on DCOPs where some constraints are more important than others. We will generate random, fully connected DCOPs, drawing the rewards of every constraint function from different normal distributions. The variance in rewards is controlled by means of weights, formalizing the importance of specific constraints with respect to the whole problem. We attach a weight $w_i \in [0, 1]$ to each constraint c_i; the problem's variance σ is multiplied with this weight when building the reward function for constraint c_i. A weight of 1 indicates the constraint is of the highest importance, while 0 makes the constraint of no importance. When building a DCOP, rewards for constraint c_i are drawn from this distribution:

$$\mathcal{N}(0, \sigma w_i) \tag{6}$$

Figure 5 visualizes the structure of the problem we will compare different LJALs on in the first experiment. The colors of constraints or edges indicate the importance of that constraint. The darker the constraint, the higher the weight. The rewards for each constraint function are fixed before every run with $\sigma = 70$ ($10 \times \#$ agents). The black edges in the figure correspond to weights of 0.9, light-grey edges are weights of 0.1. What this graph formalizes, is that the constraints between agents 1, 2 and 3, and 5 and 6 are very important, while the contribution of all other constraints to the total reward is quite limited.

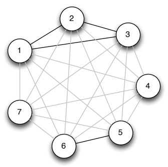


Fig. 5. Distributed constraint satisfaction problem used in the experiments. Dark edges mean important constraints, light edges are unimportant constraints.

We state again that we are interested in using knowledge of the problem's underlying structure to minimize the loss in solution quality when reducing computational complexity. Therefore, in addition to independent learners (IL), joint action learners (JAL), and local joint action learners with a random 2-degree CG (LJAL-1), we compare LJALs with a CG matching the problem structure (LJAL-2), and the same graph, augmented with coordination between agents 1 and 5 (LJAL-3), see Figure 6.

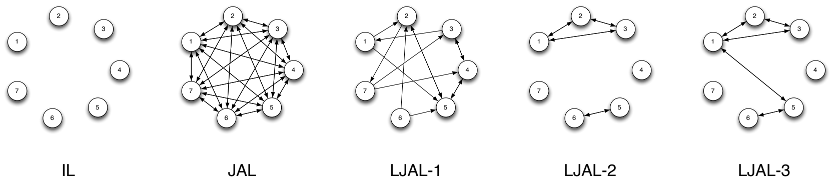


Fig. 6. Different local joint action learners, visualized by their coordination graphs. LJAL-1 is an example graph with outdegree 2.

Learner	Avg # partners	Speed	Solution Quality
IL	0	×442	86.2%
LJAL-1	2	×172	86.4%
LJAL-2	1.14	×254	91.6%
LJAL-3	1.43	×172	90.2%
JAL	6	×1	100%

Table 2. Comparison of speed and solution quality for independent learners, joint action learners and local joint action learners solving a distributed constraint optimization problem. All differences are significant $p < 0.05$.

The results, averaged over 100000 runs, are shown in Figure 7 and Table 2. As seen in the previous section, ILs and JALs perform respectively best and worst in terms of solution quality. More importantly, as we

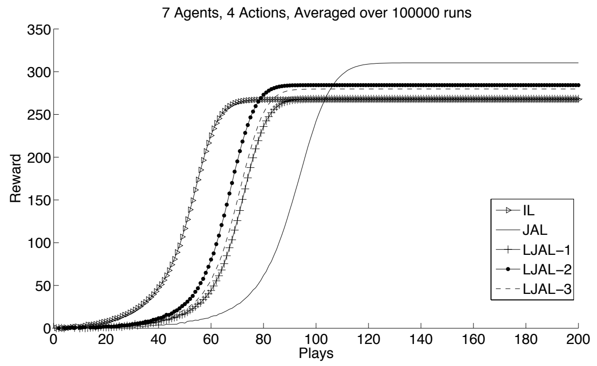


Fig. 7. Comparison of independent learners, joint action learners and local joint action learners on a distributed constraint optimization problem.

compare LJAL-1 and LJAL-2, we see that LJAL-2 perform 6% better, while being at the same time 1.5× faster. The higher solution quality results from matching coordination with the problem structure, and lower computation times are due to the lower complexity (in LJAL-1, each agent coordinates with two partners, in LJAL-2, an agent coordinates with only 1.14 partners on average[1]). This shows that using a specific CG can help LJALs solve a problem better, using less computational resources.

A more surprising result is the performance difference between LJAL-2 and LJAL-3. Although agents 1 and 5 in LJAL-3 possess more information than in LJAL-2 through increased coordination, LJAL-3 performs worse in terms of solution quality (and speed, due to the increased coordination). We hypothesise that the extra information about an unimportant constraint complicates the coordination on important constraints.

We set up an experiment to evaluate the effect an extra coordination edge has on solution quality. It compares LJAL-2 and LJAL-3 from the previous experiment with LJAL-4, which like LJAL-2 uses a graph matching the problem structure, only now augmented with a coordination edge between agents 4 and 7. As agents 4 and 7 are otherwise not involved in important constraints, we predict that adding this coordination will improve performance, as opposed to the extra edge between 1 and 5 in LJAL-3. Figure 8 and Table 3 show the results this experiment.

Since agents 4 and 7 are not involved in important constraints as defined by the problem, the addition of this edge improves performance slightly; the agents will learn to optimize the marginally important constraint between them, without complicating the coordination necessary for important constraints. These results show that the choice of the graph is very important and even small changes influence the agents' performance. In [12], Taylor et al. also conclude that increasing team work is not necessarily beneficial to solution quality.

[1] Three agents with two partners, two with one and two without partners

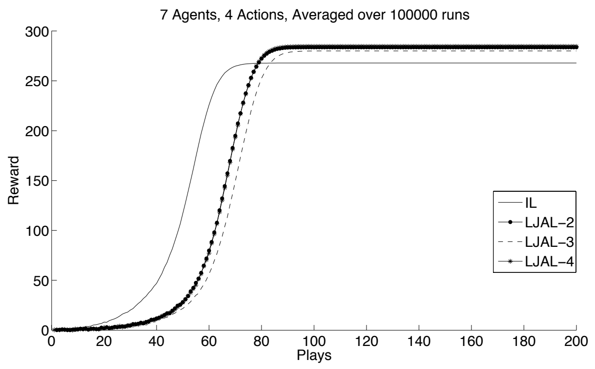


Fig. 8. Evaluating the effect of extra coordination edges on solution quality.

Learner	Solution Quality
IL	100%
LJAL-2	105.9%
LJAL-3	104.5%
LJAL-4	106.2%

Table 3. Evaluating the effect of extra coordination edges on solution quality. Solution qualities are relative to that of independent learners. All differences are significant $p < 0.05$.

5 Learning Coordination Graphs

In the previous sections, we have shown that matching the CG of local joint action learners to the inherent structure of a problem helps to improve solution quality without having to deal with the total complexity of the problem. The next problem we consider is learning this graph. In some problems, such as the graph colouring problem, this graph may be obvious. In others, the structure of the problem may not be known beforehand and thus the designer of the system has no way of knowing what graph to implement. In this section, we will investigate a way to allow the local joint action learners to optimize the CG themselves.

5.1 Method

We encode the problem of learning a CG as a distributed n-armed bandit problem. In the simplest case, each agent is allowed to pick one coordination partner and has as many actions as there are agents in the problem. For example, agent 2 choosing action 5 means a directed coordination edge in the CG from agent 2 to 5. Agent 3 choosing action 3 means agent 3 chooses not to coordinate, so no additional edge in the CG. The combined choices of the agents describe the coordination graph structure. In the experiments, we limit the learners to either one or two coordination partners, to evaluate how low complexity systems can perform on more

complex problems. We map the two-partner selection to an n-armed bandit problem by making actions represent pairs of agents instead of single agents, e.g. action 10 means selecting agents 2 and 3. This is feasible in small domains, but with more agents and a higher complexity limit per agent, the problem of choosing multiple partners should be modelled as a Markov Decision Process, with partner selection spread out over multiple states, i.e. multi-stage.
After choosing coordination partners, the agents solve the learning problem using that coordination graph. The reward achieved after learning is then used as feedback for the choosing of coordination partners; agents estimate rewards for the partner choices. This constitutes one play at the meta-learning level. This process is repeated until the graph has converged due to decreasing temperature in the meta-bandit action selection. We choose to make the agents in the meta-bandit independent learners, although it would also be possible to allow them to coordinate. Only then the question of which CG to pick would arise again.

5.2 Learning in DCOPs with a particular structure

In our first experiment, we make agents learn a CG on the problem used in previous sections and illustrated in Figure 5. As such, we can compare the learned CGs with the (to us) known problem structure. One meta-bandit run consists of 1500 plays. In each play, the chosen CG is evaluated in 100 runs of 200 plays; 100 runs to account for the inherent stochasticity of the learning process so as to get relatively accurate estimates for the quality of the chosen graph. This evaluation is basically the same setup as the experiments in Section 4.2. The average of the reward achieved over these 100 runs is the estimated reward for the chosen CG.
In addition to ILs, JALs and LJALs with a CG matching the problem structure, LJAL-1, we compare two teams of LJALS who optimize their CG, with respective complexity limits of one, OptLJAL-1, and two, OptLJAL-2, coordination partners. Figure 9 and Table 4 show the results of this experiment, averaged over 1000 runs, each time a newly generated problem, although with the same inherent structure. Temperature τ in the meta-bandit is decreased as such: $\tau = 1000 \times 0.994^{play}$.

Learner	Avg # partners	Speed	Solution Quality
IL	0	$\times 374$	86.2%
LJAL-1	1.14	$\times 243$	91.1%
OptLJAL-1	0.81	$\times 290$	94.5%
OptLJAL-2	1.28	$\times 240$	94.7%
JAL	6	$\times 1$	100%

Table 4. Comparing the computation speeds and solution qualities of independent learners, joint action learners, local joint action learners with the supposedly optimal coordination graph and local joint action learners who optimize their coordination graph, respectively limited to one and two coordination partners per agent. All differences are significant $p < 0.05$, except between OptLJAL-1 and OptLJAL-2.

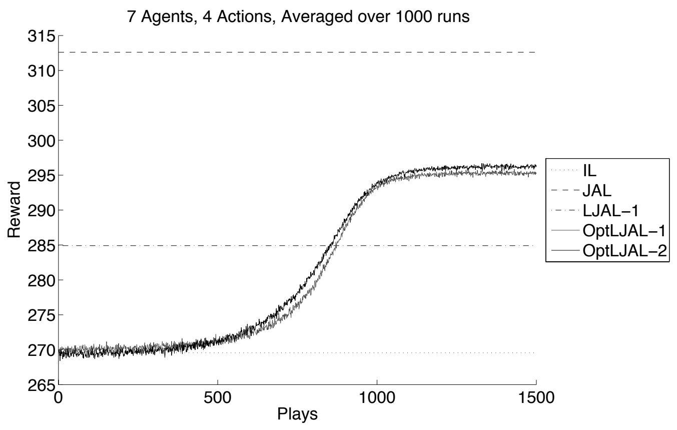


Fig. 9. Comparing the solution qualities of independent learners, joint action learners, local joint action learners with the supposedly optimal coordination graph and local joint action learners who optimize their coordination graph.

The results show that not only can the agents adapt their coordination graph to the problem and thus improve performance over agents with random graphs, they also manage to outperform the LJALs that use the CG mimicking the problem structure. That graph is surprisingly not the optimal coordination structure, as the optimizing agents in general find better graphs, graphs with a lower complexity; a maximum complexity of one coordination partner in the case of OptLJAL-1, as opposed to two partners in the graph matching the problem. OptLJAL-2 has similar performance as OptLJAL-1, although with a slightly higher complexity and thus longer computation time. It is important to note that graphs optimized by OptLJAL-2 in general have a complexity of 1.28, which is very low considering the highest possible complexity is 2. More coordination again does not appear to be always beneficial. Compare for example the average complexities of the resulting graphs, 0.81 and 1.28 for limits 1 and 2 respectively, with that of the random graphs in the exploration stages: 0.86 and 1.59.

To get a better insight into how OptLJAL-1 and OptLJAL-2 can outperform LJAL-1, we look at some of the optimized graphs for this problem. Figure 10 shows the graphs learned by OptLJAL-1 and OptLJAL-2 respectively on five instances of the given problem. These graphs represent cases where optimizing agents significantly outperformed LJAL-1, who mimick the problem structure in their CG.

When viewing these optimized graphs, we would expect to find at least some of the problem structure reflected in them. This is clearly the case. In every single graph, we find that agents 5 and 6 learn to coordinate. There is also always some coordination in the agents 1-2-3 cluster. This is also reflected in Table 5, where the average number of edges between any two agents in a cluster is shown. Agents 1, 2 and 3, and agents 5 and 6 coordinate significantly more than they would in random graphs, while 4 and 7 coordinate less. Counting the incoming edges, we note that agents 1, 2, 3 have on average 1.0 agents adapting to them, 5 and 6 have 1.2 such agents, while 4 and 7 only 0.1.

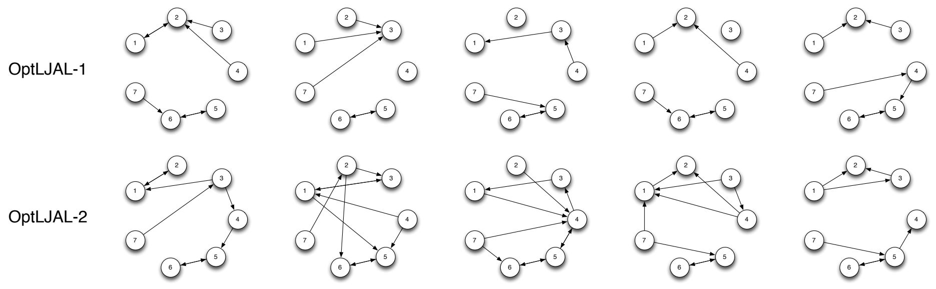


Fig. 10. Optimized coordination graphs. Graphs in the top row are limited to one coordination partner per agent, graphs in the bottom row are limited to two partners.

	1-2-3	5-6	4-7
OptLJAL-1	0.68	1.53	0.13
Random-1	0.28	0.28	0.28
OptLJAL-2	0.85	1.54	0.33
Random-2	0.53	0.53	0.53

Table 5. The average number of directed edges between any two agents in a cluster. Agents in important problem substructures coordinate significantly more often in optimized graphs than in random graphs. The inverse is true for agents in unimportant substructures.

This shows that the agents can determine which agents are more important to coordinate with. Still, this does not explain how the agents with an optimized graph can perform better with a lower coordination complexity than those who use the problem structure as a coordination graph. We believe the explanation is two-fold. First, in optimized graphs, agents often practice something we like to call "follow the leader". Basically, this comes down to one agent performing as leader, often an independent learner, while other agents coordinate unilaterally with that agent. This allows the other agents to choose actions in function of the same leader, while that leader can learn without knowing that other agents are coordinating, or rather adapting, to him, simplifying the problem for every agent by concentrating the exploration in certain parts of the search space. This is especially beneficial when only a limited amount of trials is allowed. Secondly, agents that do not coordinate directly are independent learners relative to each other. Independent learners have been shown to be able to find an optimum by climbing, i.e. each agent in turn changing an action [5]. The starting point for this climbing, in a two-dimensional game, is usually the row and column with the highest average reward. If the global optimum can be reached by climbing from this starting point, independent learning suffices to optimize the problem. When analysing the reward functions for these agents that choose to be independent learners, we see that they are involved in games where such climbing is possible. This is also the reason why a team of independent learners can perform reasonably well in this setting.

5.3 Learning in DCOPs with random structure

We have previously only focused on one specific problem, with only two very distinct categories of constraint importance, i.e. very important and very unimportant (respectively 0.9 and 0.1 as weight parameters). Such clear distinctions are not realistic and therefore we shall now investigate problems with constraints of varying importance. One issue with such problems is that, even if the structure of the problem is known, it is not easy to decide when coordination is important and when not. Is it necessary to coordinate over the constraint with weight 0.6, and not over the one with weight 0.59? Learning the graph should prove to be a better approach than guessing or fine-tuning by hand, as evidenced by the previous experiment where the preprogrammed graph was shown not to be optimal compared to other graphs of similar and even lower complexity.
The next experiment compares ILs, JALs, LJALs with a fixed CG, LJAL-1, and two teams of LJALs learning a CG, OptLJAL-1 and OptLJAL-2, on DCOPs with a randomly generated weights graph. The non-optimizing LJALs have a CG derived from the problem's weight graph; all constraints with weight 0.75 and higher are included in the graph. The results of this experiment are shown in Figure 11 and Table 6.

Learner	Avg # partners	Speed	Solution Quality
IL	0	$\times 315$	88.9%
LJAL-1	1.5	$\times 101$	90.2%
OptLJAL-1	0.8	$\times 254$	94.2%
OptLJAL-2	1.28	$\times 204$	94.3%
JAL	6	$\times 1$	100%

Table 6. Comparing the computation speeds and solution qualities of independent learners, joint action learners, local joint action learners with the supposedly optimal coordination graph and local joint action learners who optimize their coordination graph, respectively limited to one and two coordination partners per agent. All differences are significant $p < 0.05$, except between OptLJAL-1 and OptLJAL-2.

Although the LJALs with fixed CG coordinate over a quarter of all the constraints, and the most important ones at that, they do not manage to improve much over the solutions found by ILs. These LJALs have a

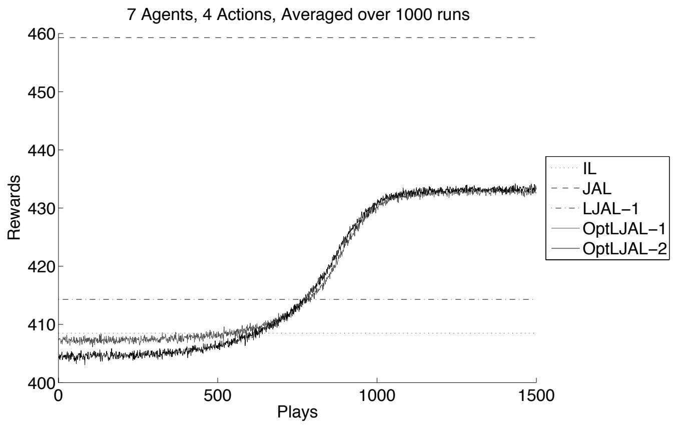


Fig. 11. Comparing the solution qualities of independent learners, joint action learners, local joint action learners with fixed coordination graph and local joint action learners who optimize their coordination graph on distributed constraint optimization problems with a random weights graph.

CG with an average complexity of $\frac{7\times6\times0.25}{7} = 1.5$ coordination partners per agent. Compare that to the average complexity of 0.8 in OptLJAL-1. With less coordination and therefore less computation, they again manage to improve much on the solution quality.

6 Conclusion

In this paper, we investigated local coordination in a multi-agent reinforcement learning setting as a way to reduce complexity. *Local joint action learners* were developed as a trade-off between independent learners and joint action learners. Local joint action learners make use of a *coordination graph* that defines which agents need to coordinate when solving a problem. The density of the graph determines the computational complexity for each agent, and also influences the solution quality found by the group of agents.
Problems that have an inherent structure, making coordination between certain agents more important, can be solved by local joint action learners that have a coordination graph adapted to the structure of the problem. Learners using such a graph can perform better than those using a random graph of higher density, both in terms of solution quality and computation time.
We have also shown that the coordination graph itself can be optimized by the agents to better match the potentially unknown structure of the problem being solved. This optimization often leads to unexpected graphs, where important constraints in the problem are not mimicked in the coordination graph by a direct coordination link. Instead, this coordination is achieved through mechanisms such as leader-follower relationships and relative independent learning.

References

1. Ali, S., Koenig, S., Tambe, M.: Preprocessing techniques for accelerating the dcop algorithm adopt. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. pp. 1041–1048. AAMAS '05, ACM, New York, NY, USA (2005)
2. Chapman, A.C., Micillo, R.A., Kota, R., Jennings, N.R.: Decentralised dynamic task allocation: a practical game-theoretic approach. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 915–922. AAMAS '09, Richland, SC (2009)
3. Chapman, A.C., Rogers, A., Jennings, N.R., Leslie, D.S.: A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems. Knowledge Eng. Review 26(4), 411–444 (2011)
4. Chechetka, A., Sycara, K.: No-commitment branch and bound search for distributed constraint optimization. In: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. pp. 1427–1429. ACM, New York, NY, USA (2006)
5. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Proceedings of National Conference on Artificial Intelligence (AAAI-98. pp. 746–752 (1998)
6. Guestrin, C., Lagoudakis, M., Parr, R.: Coordinated reinforcement learning. In: In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning. pp. 227–234 (2002)
7. Kho, J., Rogers, A., Jennings, N.R.: Decentralized control of adaptive sampling in wireless sensor networks. ACM Trans. Sen. Netw. 5(3), 19:1–19:35 (Jun 2009)
8. Kok, J.R., Vlassis, N.: Using the max-plus algorithm for multiagent decision making in coordination graphs. In: RoboCup-2005: Robot Soccer World Cup IX (2005)
9. van Leeuwen, P., Hesselink, H., Rohling, J.: Scheduling aircraft using constraint satisfaction. Electronic Notes in Theoretical Computer Science 76(0), 252 – 268 (2002)
10. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: An asynchronous complete method for distributed constraint optimization. In: Autonomous Agents and Multiagent Systems. pp. 161–168 (2003)
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press (Mar 1998)
12. Taylor, M.E., Jain, M., Tandon, P., Yokoo, M., Tambe, M.: Distributed on-line multi-agent ooptimization under uncertainty: Balancing exploration and exploitation. Advances in Complex Systems (ACS) 14(03), 471–528 (2011)