The Knowledge Engineering Review, Vol. 00:0, 1–24. © 2004, Cambridge University Press
DOI: 10.1017/S0000000000000000 Printed in the United Kingdom

Overcoming Incorrect Knowledge in Plan-Based Reward Shaping

KYRIAKOS EFTHYMIADIS, SAM DEVLIN and DANIEL KUDENKO

Department of Computer Science, University of York, UK
E-mail: kirk@cs.york.ac.uk, sam.devlin@york.ac.uk, daniel.kudenko@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.

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 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) (Grześ & Kudenko 2008*b*, Devlin et al. 2011) 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 sub-optimal steps it takes and so directing it towards the optimal policy quicker. Plan-based reward shaping (Grześ & Kudenko 2008*b*) 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 (Grześ & Kudenko 2008*b*) 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 erroneous domain knowledge whilst learning and continue to use the then corrected knowledge to guide the RL process.

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 domain 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 (Sutton & Barto 1998). 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 (Puterman 1994).

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 (Bertsekas 2007).

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 (Sutton & Barto 1998). After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

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

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 (Sutton & Barto 1998).

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 (Sutton & Barto 1998).

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.

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 (Ng et al. 1999, Randløv & Alstrom 1998). This concept can be represented by the following formula for the SARSA algorithm:

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

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 (Randløv & Alstrom 1998). To deal with such problems, potential-based reward shaping was proposed (Ng et al. 1999) as the difference of some potential function Φ defined over a source s and a destination state s':

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \tag{3}$$

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

Potential-based reward shaping, defined according to Equation 3, has been proven to not alter the optimal policy of a single agent in both infinite- and finite- state MDPs (Ng et al. 1999) .

More recent work on potential-based reward shaping, has removed the assumptions of a single agent acting alone and of a s static potential function from the original proof.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 (Devlin & Kudenko 2011). 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 (Devlin & Kudenko 2012).

2.3 Plan-Based Reward Shaping

Reward shaping is typically implemented bespoke for each new environment using domain-specific heuristic knowledge (Devlin et al. 2011, Randløv & Alstrom 1998) but some attempts have been made to automate (Grześ & Kudenko 2008*a*, Marthi 2007) and semi-automate (Grześ & Kudenko 2008*b*) the encoding of knowledge into a reward signal. Automating the process requires no previous knowledge and can be applied generally to any problem domain. 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. STRIPS is a popular and well studied formalism used to express automated planning instances. We briefly explain the STRIPS formalism here but refer the reader to Fikes & Nilsson (1972) for further details on this topic.

The description of a planning problem in STRIPS includes the operators (or actions), the initial state and a goal state. Operators have a set of preconditions which have to be satisfied for the action to be executable, and a set of effects which occur when executing the operator. The start state contains a set of conditions which are initially true, and the goal state consists of conditions which have to be true or false for the state to be classified as a goal state.

The output of the planner is the sequence of actions, that will satisfy the conditions set at the goal state. The STRIPS plan is translated from an action-based, to a state-based representation so that, whilst acting, an agent's current state can be mapped to a step in the plan[1] (as illustrated in Figure 1).

The potential of the agent's current state then becomes:

$$\Phi(s) = CurrentStepInPlan * \omega \tag{4}$$

[1]Please note that 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.

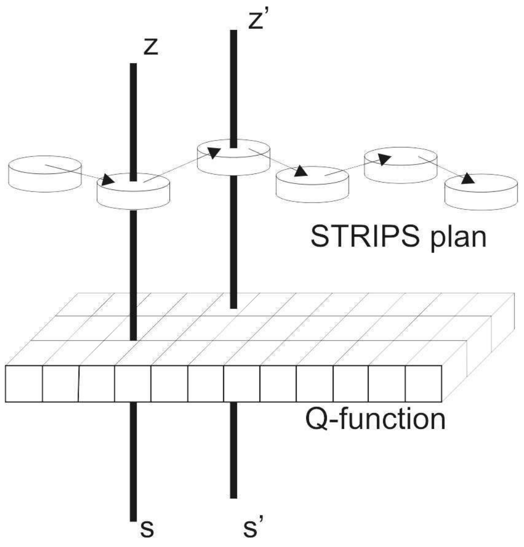

Figure 1: Plan-Based Reward Shaping.

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 (Devlin & Kudenko 2012).

To preserve the theoretical guarantees of potential-based reward shaping, the potential of all terminal states is set to zero.

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 (Grześ & Kudenko 2008*b*).

However, this method struggles when the given knowledge is partially erroneous 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 erroneous knowledge online allowing the agent to still benefit from the 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 (Grześ & Kudenko 2008*b*); 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, in which an agent learns to collect flags spread throughout a maze and take them to a goal or exit location. 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

```
MOVE(hallA,roomD)
TAKE(flagD,roomD)
```

Listing 1 Example Partial STRIPS Plan

up, down, left or right and will deterministically complete its move provided it does 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.

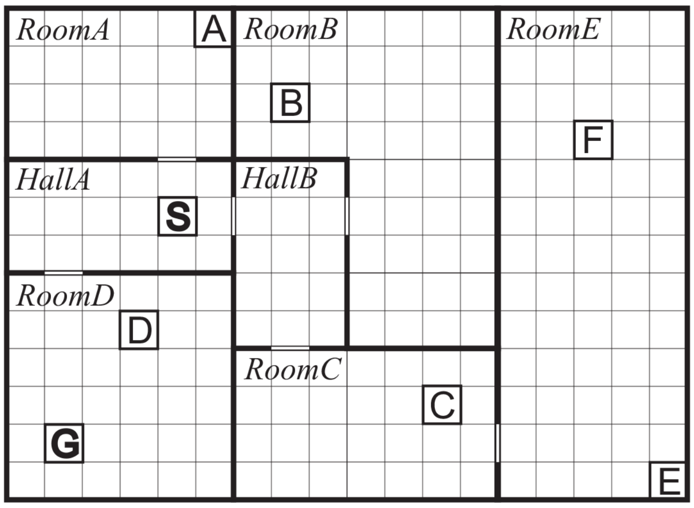

Figure 2: Flag-Collection Domain.

Figure 2 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.

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 1). 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.

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 (Grześ & Kudenko 2008*b*), 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.

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 (Gärdenfors 1992). 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.

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 */
```

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.

It is worth noting that while there are more types of wrong knowledge that can be provided to an agent e.g. incomplete information, wrong plan sequence etc., these are not within the scope of this report and are a part of future work.

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 3 in which the grey nodes ($N1 - N3$) have had all their edges expanded and green nodes ($N4 - N7$) have unexpanded edges ($E9 - E14$).

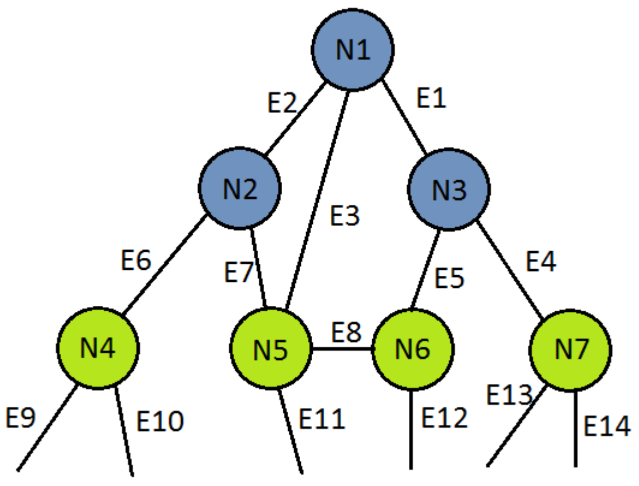

Figure 3: Instance of the Search Tree.

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.

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.

In order to allow the agent to create an estimate of the provided knowledge, and to discourage knowledge verification and revision to take place instantly, this procedure only takes place after a few hundred episodes have been completed.

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 is made possible by the fact that the agent is running in simulation, as mentioned in 3.1, and its state can easily be reset. 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.

The search finishes once the agent has either found the failing flag or all of the nodes that were

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
```

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 (Gärdenfors 1992). 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[2]. 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.

[2]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 $K \dot{-} \phi$. (Gärdenfors 1992)

```
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

To illustrate the use of this method consider a domain similar to that shown in Figure 2 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*.

Let us 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 (Asmuth et al. 2008). Furthermore, it has been shown before without eligibility traces (Ng et al. 1999, Devlin et al. 2011) and proven for any action selection method that chooses actions based on relative difference and not absolute magnitude (Asmuth et al. 2008).

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 50000 episodes and was repeated 10 times for each instance of the erroneous knowledge. The agent is compared to the plan-based RL agent without knowledge revision when provided with the same erroneous knowledge as our agent and when the same agent is provided with correct knowledge. The agents are compared against the total discounted reward they achieve. 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. A comparison on the number of steps each agent performs per episode is also presented in order to illustrate the impact of wrong knowledge in the number of steps the agent needs to perform to complete an episode. For clarity all the graphs only display results up to 5000 episodes, after this time no significant change in behaviour occurred in any of the experiments. The graphs also include error bars showing the standard error. The averaged results are presented in Figures 4, 5, 6 and 7.

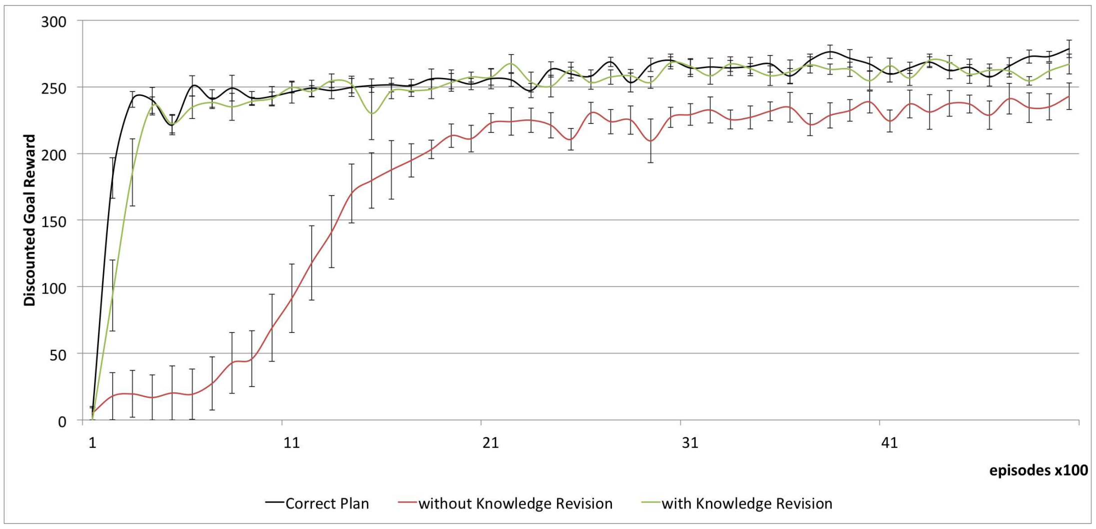

Figure 4: Non-existing flags: 1 flag.

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 correct knowledge.

The agents were provided with more instances of incorrect knowledge reaching up to eight

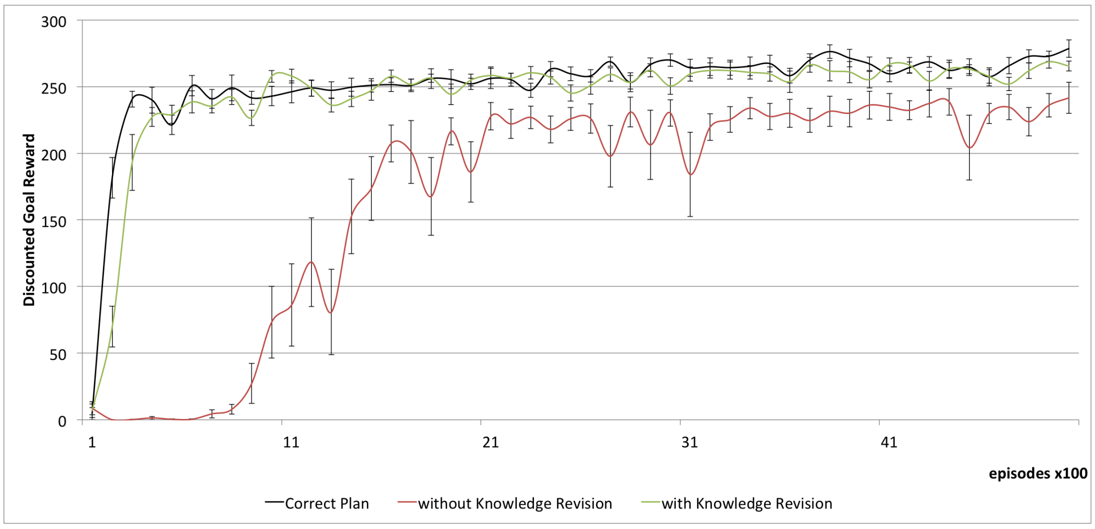

Figure 5: Non-existing flags: 2 flags.

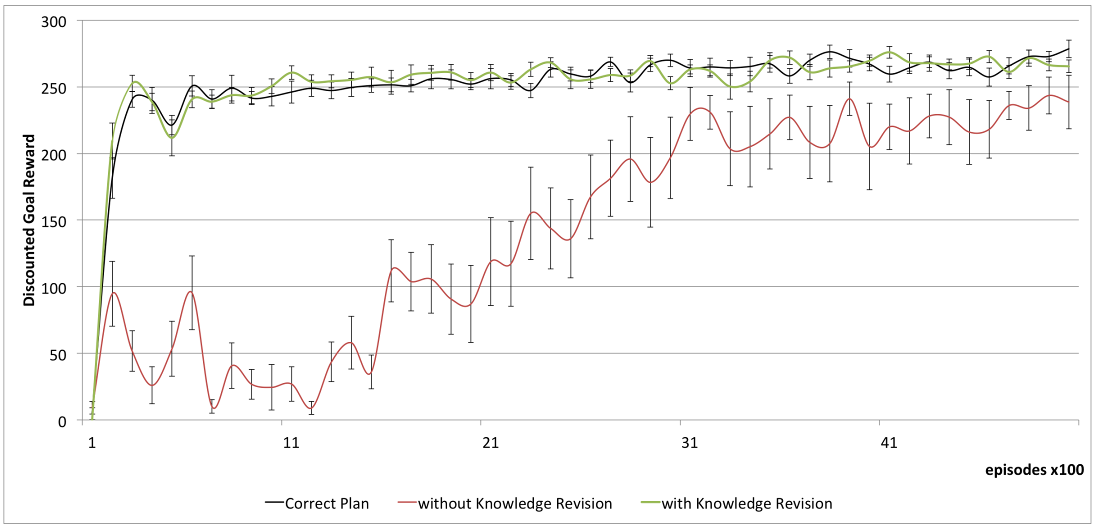

Figure 6: Non-existing flags: 3 flags.

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 7 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 and spend less time choosing suboptimal exploratory action early in the experiments. 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.

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.

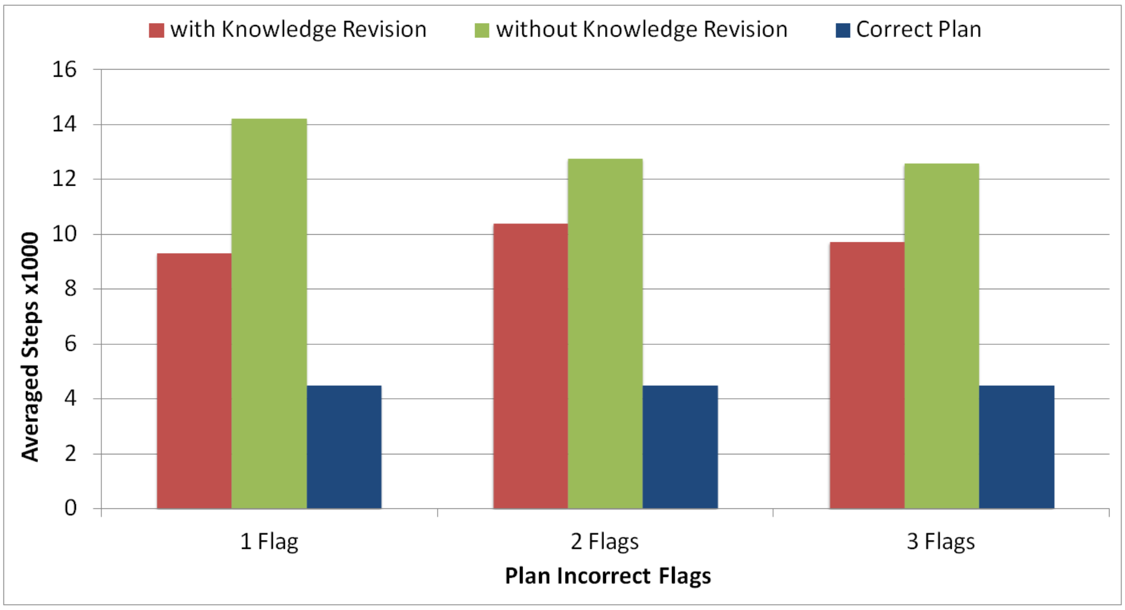

Figure 7: Average number of steps taken per experiment.

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 demonstrated 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 e.g. plan-based reward shaping has been applied to the real-time strategy game Starcraft: Broodwar to learn agent strategies (Efthymiadis & Kudenko 2013).

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.

Acknowledgements

This study was partially sponsored by QinetiQ under the EPSRC ICASE project *"Planning and belief revision in reinforcement learning"*.

References

Asmuth, J., Littman, M. & Zinkov, R. (2008), Potential-based shaping in model-based reinforcement learning, *in* 'Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence', pp. 604–609.

Bertsekas, D. P. (2007), *Dynamic Programming and Optimal Control (2 Vol Set)*, Athena Scientific, 3rd edition.

Devlin, S., Grześ, M. & Kudenko, D. (2011), 'An empirical study of potential-based reward shaping and advice in complex, multi-agent systems', *Advances in Complex Systems* .

Devlin, S. & Kudenko, D. (2011), 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'.

Devlin, S. & Kudenko, D. (2012), Dynamic potential-based reward shaping, *in* 'Proceedings of The Eleventh Annual International Conference on Autonomous Agents and Multiagent Systems'.

Efthymiadis, K. & Kudenko, D. (2013), Using plan-based reward shaping to learn strategies in starcraft: Broodwar, *in* 'Computational Intelligence and Games (CIG)', IEEE.

Fikes, R. E. & Nilsson, N. J. (1972), 'Strips: A new approach to the application of theorem proving to problem solving', *Artificial intelligence* **2**(3), 189–208.

Gärdenfors, P. (1992), 'Belief revision: An introduction', *Belief revision* **29**, 1–28.

Grześ, M. & Kudenko, D. (2008*a*), 'Multigrid Reinforcement Learning with Reward Shaping', *Artificial Neural Networks-ICANN 2008* pp. 357–366.

Grześ, M. & Kudenko, D. (2008*b*), Plan-based reward shaping for reinforcement learning, *in* 'Proceedings of the 4th IEEE International Conference on Intelligent Systems (IS'08)', IEEE, pp. 22–29.

Marthi, B. (2007), Automatic shaping and decomposition of reward functions, *in* 'Proceedings of the 24th International Conference on Machine learning', ACM, p. 608.

Ng, A. Y., Harada, D. & Russell, S. J. (1999), Policy invariance under reward transformations: Theory and application to reward shaping, *in* 'Proceedings of the 16th International Conference on Machine Learning', pp. 278–287.

Puterman, M. L. (1994), *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., New York, NY, USA.

Randløv, J. & Alstrom, P. (1998), Learning to drive a bicycle using reinforcement learning and shaping, *in* 'Proceedings of the 15th International Conference on Machine Learning', pp. 463–471.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, MIT Press.