Knowledge-Based Reward Shaping with Knowledge Revision in Reinforcement Learning

KYRIAKOS EFTHYMIADIS

PHD
UNIVERSITY OF YORK
COMPUTER SCIENCE

September 2014

For all those most loved to me people, who have supported me through this research degree and without whom this would not have been possible.

Abstract

Reinforcement learning has proven to be a successful artificial intelligence 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 and in time manages to identify which actions are more beneficial for each situation.

Typically 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 reinforcement learning tasks is available by domain experts, and can be used to improve the learning performance. One way of imparting knowledge to an agent is through reward shaping which guides an agent by providing additional rewards.

One common assumption when imparting knowledge to an agent, is that the domain knowledge is always correct. Given that the provided knowledge is of a heuristic nature, there are cases when this assumption is not met and it has been shown that in cases where the provided knowledge is wrong, the agent takes longer to learn the optimal policy. As reinforcement learning methods are shifting more towards informed agents, the assumption that expert domain knowledge is always correct needs to be relaxed in order to scale these methods to more complex, real-life scenarios. To accomplish that, the agents need to have a mechanism to deal with those cases where the provided expert knowledge is not perfect.

This thesis investigates and documents the adverse effects erroneous knowledge can have to the learning process of an agent if care is not taken. Moreover, it provides a novel approach to deal with erroneous knowledge through the use of knowledge revision principles, in order to allow agents to use their experiences to revise knowledge and thus benefit from more accurate shaping. Empirical evaluation shows that agents that are able to revise erroneous parts of the provided knowledge, can reach better policies faster when compared to agents that do not have knowledge revision capabilities.

Contents

List of Tables

List of Figures

Acknowledgements

I want to express my gratitude to my supervisor Dr. Daniel Kudenko for his advice, encouragement and excellent supervision during the 4 years that this PhD lasted. It has been a great honour working with you. In addition I want to thank my assessor Alan Frisch for his useful feedback all these years. His comments and attention to detail have helped immensely with the organisation of this thesis. In addition I would like to thank my examiner Karl Tuyls for his time and feedback.

I address my acknowledgements to our QinetiQ collaborators for their support and financial help. Discussions with David Salmond and Joel Goodman at the beginning of our project were very stimulating for me and drove part of this research.

I must express my deepest love to my family who have made me who I am. Their support and love provides me with the energy to pursuit all my dreams no matter how difficult they seem to be.

I am very grateful to Liana Kafetzopoulou who supported me with her love and encouraged me to go through the ups and downs of my PhD research.

Last but not least, I want to thank all the people that I met here in York. Especially Thanasis Zolotas, Anna Ladi, Eva Karadimou, Tasos Tsompanidis and Sotiris Banatas for those great fun days we had while I was in York.

Special thanks to Sam Devlin and Kleanthis Malialis for the fruitful discussions on RL and the great collaboration we had over the past 4 years. Big thank you also to Marek Grzès for his code on the flag-collection domain which he developed during his PhD in York and which I used for some of the experiments in my research.

Thank you everyone for what has been an amazing period. I will surely miss you all.

May the force be with you !!!

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of York. This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references.

Some of the material contained in this thesis has appeared in the following published or awaiting publication papers:

1. K. Efthymiadis and D. Kudenko. Knowledge Revision for Reinforcement Learning with Abstract MDPs, *In Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2015.

2. K. Efthymiadis and D. Kudenko. Using Plan-Based Reward Shaping To Learn Strategies in StarCraft: Broodwar, *In Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*, 2013.

3. K. Efthymiadis, S. Devlin, and D. Kudenko. Overcoming Erroneous Domain Knowledge in Plan-Based Reward Shaping (Extended Abstract), *In Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

4. K. Efthymiadis, S. Devlin, and D. Kudenko. Overcoming Incorrect Knowledge in Plan-Based Reward Shaping, *In Knowledge Engineering Review (KER)*, Cambridge Journals. (In Press)

5. K. Efthymiadis and D. Kudenko. A Comparison of Plan-Based and Abstract MDP Reward Shaping, *In Connection Science*. (In Press)

6. K. Efthymiadis, S. Devlin, and D. Kudenko. Abstract MDP Reward Shaping for Multi-Agent Reinforcement Learning, *In 11th European Workshop on Multi-Agent Systems*, (EU-MAS), 2013.

CHAPTER 1

Introduction and Motivation

The research topic of this thesis falls within the general field of Artificial Intelligence (AI). The goal of AI is to build processes, or agents as they are widely referred to that act in an intelligent manner.

AI is a vast research area but the specific field of interest in this case is Reinforcement Learning (RL). In RL agents are deployed in an environment in which they must learn how to adapt and complete certain tasks. To do so, agents can perform actions which might change the state of the environment which in turn provides a numerical reward signal that can be positive or negative depending on the action chosen. Through continuous interaction the agents can learn to identify those actions that best fit certain situations so as to maximise their future rewards (Sutton and Barto 1998).

Typically RL assumes no prior domain knowledge and the agents must build their knowledge through trial and error. This seems counter intuitive as in most cases the designer of the system will have some form of high level knowledge of what the agents need to achieve. This domain knowledge can be imparted to an agent so as to improve its learning process. As a result RL research is moving towards techniques of exploiting existing domain knowledge and the general term referred to is knowledge-based RL. One popular approach of providing domain knowledge to an agent is potential-based reward shaping (Randløv and Alstrom 1998; Ng et al. 1999). The results of this approach show that agents are guaranteed to find the same policy as with a non-shaped agent, but the convergence time of the latter is significantly greater.

While potential-based reward shaping has been shown to speed up the learning process significantly, one limiting assumption that remains is that the provided domain knowledge is always correct. This naive assumption can hurt an agent's performance given that expert domain knowledge is often of a heuristic nature and can be erroneous. For example, it has been shown in (Grześ and Kudenko 2008) that if the provided knowledge is flawed then the agent's learning performance drops and in some cases is worse than not using domain knowledge at all.

A solution to this assumption can be to add the capability of revising knowledge to RL agents. This thesis will explore this idea and will empirically demonstrate the impact that erroneous knowledge can have to agents utilising reward shaping and methods to overcome the negative effects that it can cause.

1.1 Hypothesis

The overall aim of this thesis is to demonstrate:

> Adding knowledge revision capabilities to reinforcement learning agents utilising reward shaping can alleviate the adverse effects of erroneous domain knowledge by improving its quality and thus agents can reach a better overall performance in terms of convergence speed and learnt policy compared to agents without knowledge revision and agents that receive no shaping. Agents without knowledge revision receiving erroneous knowledge may still reach a better performance than agents without shaping.

1.2 Scope

This thesis explores knowledge revision within the context of two potential based reward shaping methods; plan-based and abstract Markov Decision Process (MDP) reward shaping. These methods were chosen in order to utilise knowledge revision principles as they provide a very intuitive mechanism to handle erroneous knowledge. Plan-based reward shaping can utilise basic belief revision operations based on the Alchourrón, Gärdenfors and Makinson (AGM) postulates and abstract MDPs can be handled by updating probabilities. More information is provided in Chapter 2.

While many other approaches of reward shaping can be chosen to design knowledge revision methods for reinforcement learning agents there have not been any published studies within the context of potential-based shaping prior to the start of this thesis. Therefore, the question of the effects of erroneous domain knowledge remained unanswered as the majority of reward shaping methods assumed that the provided knowledge is always correct.

1.3 Thesis Overview

The next chapter provides an overview of the field, focusing on the necessary aspects of the literature this work builds upon, in order to make the latter parts of the thesis accessible to all readers.

Chapter 3 presents an empirical study on plan-based reward shaping with knowledge revision by providing algorithms in order to handle different aspects of erroneous knowledge; incorrect, incomplete and a combination of both. In order to assess the performance of the agents two deterministic environments were used, an extended navigation domain and a real-time strategy game. These studies demonstrate that using knowledge revision with plan-based reward shaping by building upon the AGM postulates, can help agents improve the quality of the provided domain knowledge and thus reach a better overall performance compared to agents that do not revise knowledge and agents that do not use any reward shaping methods.

Chapter 4 explores knowledge revision methods for agents that use abstract MDP reward shaping. It documents algorithms for knowledge revision that update probabilities of a high level MDP that represents the environment. This study expands knowledge revision capabilities to also cover non-deterministic environments as the methods are also evaluated in a Micro UAV problem which was provided by our industrial collaborators at QinetiQ.

Finally, Chapter 5 focuses on using abstract MDP reward shaping for conflict resolution in a multi-agent setting. Previous research demonstrated that agents receiving decentralised shaping encounter problems while learning due to conflicting goals. Centralised agents on the other hand are able to co-ordinate efficiently. I am interested in cases where information sharing is not allowed and this study shows that abstract MDP reward shaping, even when provided as decentralised shaping, can help agents resolve conflicting goals and co-ordinate efficiently when compared to plan-based reward shaping agents and agents that receive no shaping.

Lastly, the thesis concludes in Chapter 6 with a summary of all contributions documented in this thesis, a few comments on the limitations of this research and ways to improve it in future work.

CHAPTER 2

Background and Field Review

This chapter focuses on the fundamentals and current research state in RL necessary to understand the topic of this thesis. Section 2.1 provides an introduction to the area of RL covering all the basic concepts. Section 2.2 expands into knowledge-based methods for RL and Section 2.3 provides an introduction to knowledge revision and how it can benefit knowledge-based RL methods.

2.1 Reinforcement Learning (RL)

RL falls within the area of machine learning. In machine learning computer programs learn how to improve their performance at a specific task through experience (Mitchell 1997).

RL is a goal directed paradigm where learning occurs through *reinforcements* (Sutton and Barto 1998). The learning entity, the *agent*, is situated in a particular environment. The environment is the "world" that the agent acts upon and poses the problems that the agent is to find solutions to (Russell and Norvig 2002). The agent makes decisions based on its own motivations using the experience gained within the environment. Unlike supervised methods where the agent receives guidance on how to behave given certain situations, in RL the agent builds up its experiences through trial and error.

Figure 2.1 illustrates how the agent gains these experiences. The interaction begins with the environment presenting the current state of the world, s, to the agent. The agent then chooses, from a set of available actions, how to behave. The chosen action, a, may affect the state of the world the agent is in. The environment then presents a new state and a numerical feedback based on the state-action-state tuple. This feedback can be either positive or negative and is the

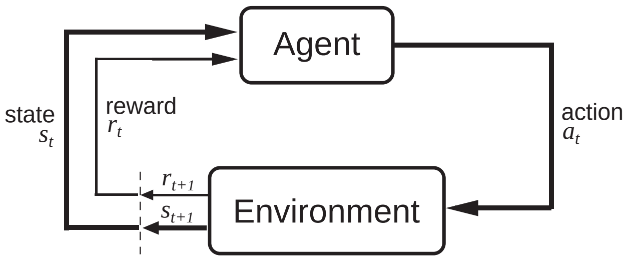


Figure 2.1: The agent-environment interaction in RL. (Sutton and Barto 1998)

reinforcement the agent uses to build up its experience on what is the correct way to behave. The same interaction then repeats with the agent choosing a new action depending on the new state of the world. (Sutton and Barto 1998)

The agent learns how to behave by using the reinforcements provided by the environment to form a *policy*. A policy $\pi : S \rightarrow A$, is a mapping from states, $s \in S$, to possible actions, $a \in A(s)$, when in state s, such that it maximizes the *cumulative reward* R_t. The cumulative reward R_t is the reward that the agent receives over time and which for episodic tasks is

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T, \tag{2.1}$$

with T being the final time-step. If the task is continuing, $T = \infty$ and as a result the reward itself might be infinite then a *discounted* cumulative reward is used

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2}$$

where γ is a parameter, $0 \leq \gamma < 1$, called the *discount factor*. The discount factor is used in order to "weight" the future rewards. Greater values of γ result in stronger future rewards and the agent becomes far-sighted while lower values result in an agent that tries to maximize only the immediate reward, r_{t+1}, and becomes myopic. Equation 2.2 can also be used in finite tasks and is not restricted to tasks where T is infinite. Sutton and Barto (1998) however state that trying to maximize only the immediate reward, could eventually also maximize equation 2.2, but might result in ignoring future rewards that could possibly lead to a greater cumulative reward R_t.

Learning the policy π, can be based on estimating *value functions*. A value function is a mapping from states and actions, to an estimation of the future reward the agent will receive if it performs action a in state s and then follow the same policy π for the rest of the interactions (Sutton and Barto 1998).

Formally, the expected value of a state s under policy π, when starting in s and following π thereafter is defined as:

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \tag{2.3}$$

V^{π} is the state-value function for policy π. The action-value function for policy π can be defined as:

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\{R_t|s_t = s, a_t = a\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \end{aligned} \tag{2.4}$$

The value functions used throughout RL satisfy the Bellman equation. The Bellman equation is defined as:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t|s_t = s\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \\ &= \sum_{a}\pi(s,a)\sum_{s'}P_a(s,s')[R_a(s,s') + \gamma V_{\pi}(s')], \end{aligned} \tag{2.5}$$

The Bellman equation expresses the relationship of the value of a state and the values of its successor states i.e. the value of the start state equals the discounted value of the next state, plus the expected future rewards.

There are three ways a value function can be initialised each with its own merits and drawbacks; optimistic, pessimistic and random initialisation. Optimistic initialisation sets the values of state-action pairs to the maximum possible value returned by the environment. Optimistic initialisation results in the agent trying all the state-action pairs before converging to a fixed policy. This can have a detrimental effect in very large domains but ensures that the optimal policy will be discovered. In contrast, pessimistic initialisation sets all the state-action values to the lowest value returned by the environment. This results in the agent following promising policies much quicker, but relies on exploration to find the optimal policy. Random initialisation can be used to balance the two approaches.

It is apparent that the actions the agent chooses are very important in discovering the optimal policy. The action selection mechanism must be set up in such a way so that the agent finds those state-action pairs that lead to higher rewards. This is known in RL as the *exploration* vs *exploitation* problem. Specifically, in each state the agent must choose whether it will exploit its knowledge i.e. select an action that the agent knows to be worthwhile, or explore new options which might lead to better rewards. It is clear that while the agent is exploring, it cannot behave optimally but such a behaviour is needed in order to find the optimal policy.

The action selection mechanisms which are common in RL are *greedy*, $\epsilon-$*greedy* and *softmax*. A greedy agent will always pick the best possible action given its knowledge. An $\epsilon-$greedy agent will pick the best action according to the probability ϵ and an exploratory action with probability $1 - \epsilon$. The $\epsilon-$greedy method is often implemented with ϵ gradually being reduced to 0 so as to favour exploitative actions as time progresses. In softmax action selection, instead of having the agent randomly exploring actions, a probability distribution is used to select exploratory actions that look promising to the agent. Very common probability distributions that are used in this approach are the Gibbs and Boltzmann distributions. (Sutton and Barto 1998)

2.1.1 Markov-Decision Processes (MDP)

The MDP is a general model for interaction that is extensively used in RL. It defines a mathematical framework to model problem domains that have the *Markov property* and is defined as a 4-tuple $< S, A, T, R >$ where:

- **S** is the state-space.
  It defines the set of possible states;
- **A** is the action-space.
  It defines the set of possible actions;
- **T** is the transition model: $T(s, a, s') = Pr(s'|s, a)$.
  It defines the probability of reaching state s' when in s, after performing action a;
- **R** is the reward function: $R(s, a, s') \in \mathbb{R}$.
  The numerical feedback provided by the environment when the agent transitions to state s' after performing action a in state s.

The Markov property defines a model where the current state summarizes all the past percepts in a compact way, however retaining all the necessary information for decision making i.e. the optimal action can be chosen just by knowing the current state while all past states and actions have no effect on the decision. (Puterman 1994)

2.1.2 Solving the RL problem

Solving the reinforcement learning problems requires the use of algorithms. Common algorithms used in RL are Monte Carlo methods, temporal difference learning and dynamic programming.

Monte Carlo Methods

In order to find the optimal policy Monte Carlo (MC) methods consider the entire history of an agent's interactions during an episode. A backup for each state is performed based on the entire sequence of observed rewards from that state until the end of the episode. A backup diagram is shown in Figure 2.2 with the root being a state node followed by the entire sequence of transitions until the terminal state.

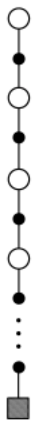


Figure 2.2: Monte Carlo Back Up Diagram (Sutton and Barto 1998).

The cumulative reward R_t is used in order to update the values of the states. This means that the update can only occur after the episode has finished and therefore all the visited states must be stored. For example, in first-visit MC, $V^{\pi}(s)$ is estimated by averaging all the returns following the first visit to state s and is updated as shown in Formula 2.6. On the other hand every-visit MC estimates $V^{\pi}(s)$ as the average of the returns following all occurrences of state s within an episodes. Storing all the states however can cause problems since the sequence of states can grow significantly in size especially in large and complex domains.

$$V(s) \leftarrow average(Returns(s)) \tag{2.6}$$

The benefit of using Monte Carlo methods can be seen however in domains where the Markov property does not hold i.e. domains where the sequence of states and rewards are important in choosing an action. Since Monte Carlo methods store all the interactions of the agent with the environment they are better suited to such domains. (Sutton and Barto 1998)

Temporal Difference Learning

Unlike Monte Carlo methods, temporal difference learning updates the state-action values not at the end of an episode, but at each time step thus eliminating the need to store sequences of interactions. At each time step, the agent uses update rules to gradually move towards the optimal policy.

The most common algorithms of this class are: Q-Learning (Watkins and Dayan 1992) and SARSA (Rummery and Niranjan 1994). Q-Learning is an off-policy method i.e. the agent updates the value function but follows an independent policy while doing so. Specifically the agent uses the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{2.7}$$

where s_t is the current state, a_t is the action taken, α is the learning rate, γ the discount factor and s_{t+1} the new state the agent transitioned to. The α parameter is set to change the way in which updates are performed. It affects the magnitude of the changes in the estimated Q-values. The γ parameter is used to weigh the importance of future and immediate rewards. Both these parameters are set differently for each experiment according to the designer's goals.

SARSA is an on-policy method and in contrast to Q-Learning does not follow an independent policy. Instead it uses the current value function in its update rule. SARSA agents use the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \tag{2.8}$$

where s_t is the current state, a_t is the action taken, α is the learning rate, γ the discount factor and s_{t+1} the new state the agent transitioned to.

Both these methods have been proven to converge to the optimal policy given the following conditions (Sutton and Barto 1998):

1. All state-action pairs are visited an infinite amount of times,
2. Exploration gradually reduces to 0,
3. The learning rate α reduces to 0,
4. The Markov property holds.

Dynamic Programming

In the cases where the MDP dynamics, such as transition probabilities T and reward function R, are known in advance there is no need to simulate the agent interactions with the environment and instead dynamic programming can be used. Dynamic programming refers to a number of algorithms that discover optimal solutions to problems presented as an MDP, given perfect information. Policy iteration and value iteration are two common dynamic programming algorithms. For an in depth analysis of dynamic programming the reader is directed towards (Bertsekas 2007).

2.1.3 Function Approximation

In theory, RL methods need to visit each state-action pair an infinite amount of times in order to converge. In small domains this is feasible and a tabular representation can be used to store each state-action pair's expected reward. This approach provides a very accurate representation of the environment but is almost impossible to use in large environments. Not only does it require a great amount of memory in order to store all the pairs, but also makes the learning task intractable. Regardless of the action selection mechanism, in very large state spaces there might still be states that cannot be visited sufficiently often, unless the training period increases significantly. Therefore more clever methods need to be used to satisfy this theoretical requirement.

This issue of handling large state or state-action spaces is known in RL as the *curse of dimensionality*. Function approximation tries to tackle the curse of dimensionality and state space explosion. Instead of storing the state-action values in lookup tables, it tries to form a function that estimates the environment. Although this approach can lose the accuracy of the tabular representation, it tries to form a good enough estimate that would make learning an optimal policy possible in large environments.

There are many different function approximation techniques, each with a different approach of generalising sensations. One of the most widely used approaches is *tile coding*. In order to create an estimate, tile coding represents the state space by the use of important *features* of the environment denoted as ϕ_i. Multiple states, or state-action pairs are represented as tiles. Each

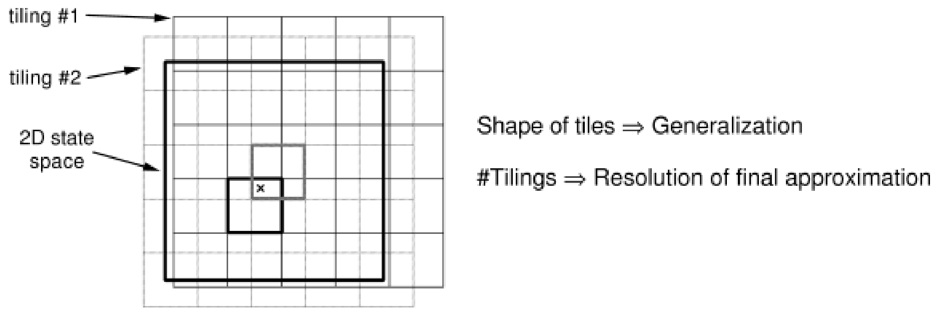

tiling #1
tiling #2
2D state space
Shape of tiles ⇒ Generalization
#Tilings ⇒ Resolution of final approximation

Figure 2.3: Tile Coding (Sutton and Barto 1998).

tile has stored information about the expected value of those multiple states or state-action pairs which results in a significant reduction of the state space. In order to increase sensitivity, multiple tilings can be overlaid as shown in Figure 2.3.

Each tile is a binary feature and is activated if the given state falls within the region denoted by that tile. The value function represented by the tile coding is determined by a set of weights, one for each tile, and is given by

$$V(s) = \sum_{i=1}^{n} b_i(s) w_i \tag{2.9}$$

where n is the number of tiles, $b_i(s)$ is the value of the ith tile which can be 0 or 1 and w_i is the weight of that tile. In practice it is not necessary to iterate through all tiles but only those that are activated at any given time. Since only one tile can be activated at any tiling, given m tilings, the indices of the m active tiles can be computed and their associated weight summed.

Given an MDP, the value estimate of a state can be computed by:

$$\Delta V(s) = \max_a [R(s,a) + \gamma V(s')] - V(s) \tag{2.10}$$

and each weight can be updated by:

$$w_i \leftarrow w_i + \frac{\alpha}{m} b_i(s) \Delta V(s) \tag{2.11}$$

Like before it is not necessary to update all weights but only the m weights of those tiles which are activated by state s.

As an example, an agent that learns how to drive a car, would not save the entire percept of the environment, but would be interested in important features such as the speed of the vehicle, the distance with the other cars, the traffic lights and road signs etc. Those selected features are weighted by w and it is left to the agent to learn those values that lead to the optimal policy.

The selection of features is a very important part in the design of the state space. There must be a selection of significantly enough features to accurately represent the environment, but not so many so as to inhibit learning.

Further information on function approximation can be found in (Sutton and Barto 1998).

2.1.4 Eligibility Traces

As discussed previously, an agent receives a reward after performing an action and then transitions to a different state. The value of that state is then updated taking into account the received reward of that particular transition. However, one problem with this approach is that all previous transitions that led to that reward are ignored and do not receive any credit. In most cases it is not the last transition the led to a high reward, but a series of transitions that allowed the agent to reach a high reward state. If those transitions are completely ignored the learning process can be slow. Especially in delayed reward environments where the reward is not received immediately, but a sequence of actions and state transitions must first occur, this problem is even more prominent. This problem is known as the *credit assignment problem.*

A method that tackles this problem and speeds up the learning process is Eligibility traces. The aim of eligibility traces is to assign a value to all visited state-value and action-value pairs following every transition. This is achieved by weighting those pairs when they are visited, and then gradually decrease their importance over time. As an example, in the case of temporal difference the eligibility trace for a state s at time t is defined as $e_t(s) \in \mathbb{R}^+$ (Sutton and Barto 1998). At each time step, the eligibility trace for all states is decayed by $\gamma\lambda$ and the eligibility trace for the current visited state is incremented by 1:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases} \tag{2.12}$$

where γ is the reward discount factor and λ is the decaying parameter used to weight the importance of the eligibility trace.

At each time-step, the agent performs the update using the equation

$$V_t(s_t) = V_t(s_t) + \alpha\delta_t e_t(s_t), \tag{2.13}$$

with

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t),$$

where α is again the learning rate and γ the reward discount factor.

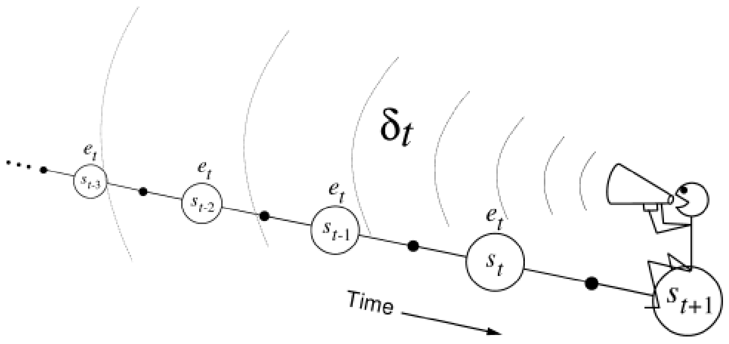


Figure 2.4: Back propagation using eligibility traces. (Sutton and Barto 1998)

Using these formulas to update the state-value or action-value pairs ensures that credit is given to all the states that led to a reward according to their recency. This is very important is order to speed up the learning process and especially when trying to scale to more complex environments. Figure 2.4 depicts very efficiently the role of the eligibility traces.

2.2 Knowledge-Based Reinforcement Learning (KBRL)

Typically RL algorithms assume that an agent initially starts with no knowledge of which actions to perform and which states are desirable in any given environment. Its value function is initialised either randomly, optimistically or pessimistically. A part that has not been given much attention when implementing RL, is that the designer of the system typically has some domain knowledge about the agent's goals that could guide the agent.

Knowledge-based RL is concerned with incorporating knowledge into an RL agent to guide its exploration. By providing informative domain knowledge, the number of sub-optimal decisions an agent makes while learning can be significantly reduced and thus the effects of the exponential state explosion can be mitigated. The intuition of incorporating domain knowledge in RL is borrowed from planning, where it has been shown that the use of admissible heuristics greatly improves search. Therefore a similar technique should also be beneficial in RL.

2.2.1 Reward Shaping

One common method of imparting knowledge to a RL agent is reward shaping. Reward shaping provides an additional reward to the agent, independent of the reward provided by the environment.

The additional reward is representative of domain knowledge and 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 and Alstrom 1998). This concept can be represented by the following formula for the SARSA algorithm:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + F(s,s') + \gamma Q(s',a') - Q(s,a)] \tag{2.14}$$

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

Although reward shaping has been powerful in many experiments it quickly became apparent that, when used improperly, it can change the optimal policy. A well known example is that of an agent trying to ride a bicycle from point A to point B (Randløv and Alstrom 1998). The agent was given additional reward when it managed to stay balanced in order to speed up the learning process. However, instead of moving towards point B, the agent found it more beneficial to ride in circles and receive the local reward. With such a poor shaping function the agent never managed to find the optimal policy.

Potential-Based Reward Shaping (PBRS)

To deal with such problems, Ng et al. (1999) proposed the use of PBRS. PBRS defines the additional rewards as the difference of some potential function Φ defined over a source s and a destination state s'. More formally:

$$F(s,s') = \gamma\Phi(s') - \Phi(s) \tag{2.15}$$

where γ must be the same discount factor as used in the agent's update rule. The potential function is a representation of the designer's, or domain expert's preference regarding specific states. If the provided knowledge is correct, then PBRS will encourage the agent to move towards the goal.

PBRS, defined according to Equation 2.15, has been proven to not alter the optimal policy of a single agent in both infinite- and finite- state MDPs (Ng et al. 1999).

Wiewiora et al. (2003) proved that PBRS is equivalent to Q-table initialisation i.e. an agent using PBRS and an agent with no shaping, but initialised with the same potential function of the PBRS agent, will exhibit the same behaviour. Figure 2.5 shows the typical behaviour of an agent receiving PBRS, presuming a good heuristic. The graph shows the performance of agents in the mountain car domain both with and without shaping. The mountain car domain is a classic testbed for reinforcement learning algorithms. The goal of the agent is to drive a car up a mountain top. The car however does not have enough power to climb it. Instead of trying to simple drive up the mountain relying on power, the agent must learn to back up in order to gain enough momentum to propel the car over the mountain top.

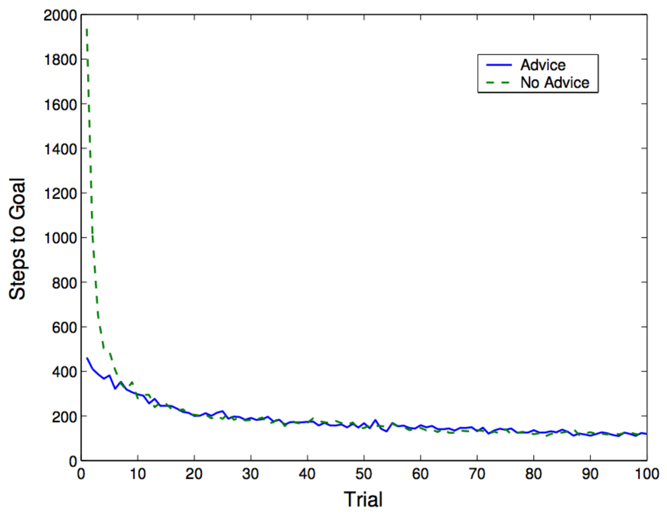


Figure 2.5: Typical behaviour of a PBRS agent in the mountain car domain. (Wiewiora et al. 2003)

Very early in the experiment the PBRS agent manages to significantly outperform the agent with no shaping. This illustrates the equivalence to Q-table initialisation as the PBRS agent instantly starts with a much better policy than the agent with no shaping.. At the end of the experiment both agents have learnt the optimal policy but in between there is a period where the agent with PBRS significantly outperforms the agent without. What PBRS offers is an increased convergence rate.

More recent work on potential-based reward shaping, has removed the assumptions of a static potential function from the original proof with the existing guarantees maintained even with a dynamic potential function (Devlin and Kudenko 2012a).

Plan-Based Reward Shaping

Reward shaping is typically implemented bespoke for each new environment using domain-specific heuristic knowledge (Devlin et al. 2011; Randløv and Alstrom 1998) but some attempts have been made to automate (Grześ and Kudenko 2008; Marthi 2007) and semi-automate (Grześ and Kudenko 2008) 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 (Grześ and Kudenko 2008), 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 as shown in Figure 2.6.

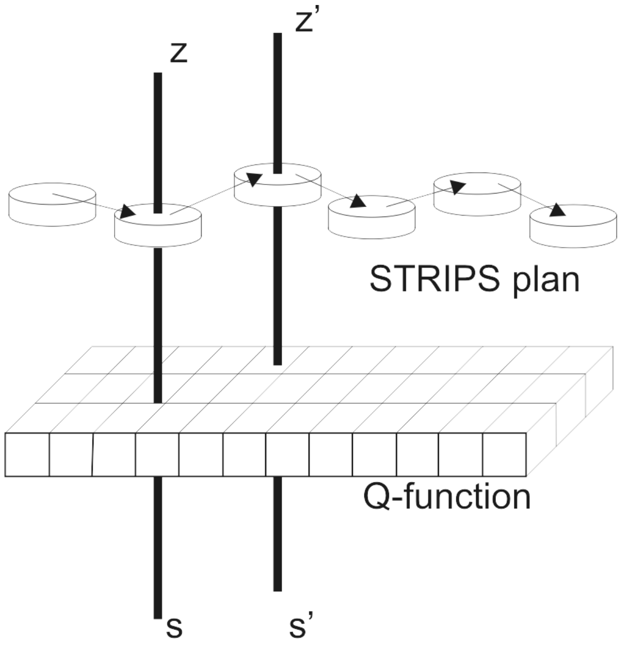


Figure 2.6: Plan-Based Reward Shaping. (Grześ and Kudenko 2008)

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

$$\Phi(s) = CurrentStepInPlan * \omega \tag{2.16}$$

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 and Kudenko 2012a).

These potentials are then used as in Equation 2.15 to calculate the additional reward given to the agent and so encourage it to follow the plan without altering the agent's original goal.

[1]This translation is automated by propagating and extracting the pre- and post- conditions of the high level actions through the plan.

Abstract MDP Reward Shaping

Marthi (2007) proposed a general automatic framework to learn the potential function by solving an abstract MDP. The shaping algorithm obtains the potential function by firstly sampling the environment in order to learn dynamics for options (i.e. actions at the abstract level) and secondly solving an abstract MDP. Options can be defined as policies of low level actions. Once the agent spends a number of episodes sampling the environment, it uses the resulting value function as a source for reward shaping.

In addition, by providing an abstraction of the low-level states of the environment to high-level abstract states[2], the abstract MDP can be solved using dynamic programming before the main learning process begins and the obtained value function is used directly as the potential function:

$$\Phi(s) = V(z) * \omega, \tag{2.17}$$

where $V(z)$ is the value function over the abstract state space Z and it represents a solution to the corresponding MDP-based planning problem, and ω is an optional scaling factor. The abstract MDP task can be solved using the following formula which is a special case of value iteration:

$$V_{k+1}(z) = \max_{z'} Pr_{zz'}[R_{zz'} + \gamma V_k(z')], \tag{2.18}$$

with $Pr_{zz'}$ being the probability of transitioning to the abstract state z' from the abstract state z, $R_{zz'}$ the reward received when transitioning to z' from z, γ the discount factor and $V_k(z)$ the value of state z at time k.

These potentials are then used as in Equation 2.15 to calculate the additional reward to be given to the agent. Learning low-level actions in order to satisfy a high-level abstract MDP is significantly easier than learning low-level actions to maximise rewards in an unknown environment and as a result agents tend to learn a policy quicker.

2.2.2 Alternate Methods for Informed Agents

Reward shaping is not the only method to provide expert domain knowledge to an agent. Many other approaches exist and many more will be discovered as researchers tackle this very promising area. While this thesis will focus only on knowledge revision in agents using reward shaping, it is worth mentioning a few alternative common methods to impart domain knowledge as they could also potentially benefit from the idea of revising beliefs.

[2]Please note that a high-level abstract state will map to many low level states. Therefore, even when provided with the correct knowledge, the agent still needs to learn the optimal path to finish the episode.

Value Function Initialisation

One of the earlier approaches in adding domain knowledge, was that of value function initialization. Different initializations of the value function, pessimistic, optimistic or random, can have different results both in time and in the policy that is learnt.

As noted earlier this approach has been proven to be equivalent to PBRS (Wiewiora et al. 2003). However in large state or state-action space domains the task of initializing the value function is very complex and time consuming.

Feature Selection

A simple method of KBRL is to partition the state-space by only including those features that the designer believes are relevant to the goal. Although this results in agents learning much quicker, as the environment is much smaller, if the selected features are not selected carefully, it might lead the agent away from the optimal policy as there are no theoretical guarantees.

Hierarchical RL

Hierarchical RL (HRL) introduces abstractions in order to discard information that is irrelevant about the task at hand. It uses *macro-operators* i.e. a series of operators which can be invoked as a single primitive action. This results in a decomposition of the overall task to subtasks which are what the agent tries to learn. The selection of macros along with their policies can be provided to the agent a priori in terms of domain knowledge and thus improve the learning rate. For more information on some of the representative approaches of HRL, the reader is referred to (Barto and Mahadevan 2003).

Relational RL

Relational RL (RRL) is concerned with objects and their relationships. RRL uses a first-order representation to encode states, actions and rewards in a given environment with the goal of the agent now being to learn the abstract policy. This form of first-order representation allows domain knowledge to be incorporated into an agent whenever there is a clear higher level knowledge of object relations at a given task. More information on this approach can be found in (Dzeroski et al. 2001; Lau et al. 2013).

2.3 Knowledge Revision (KR)

Knowledge based methods have been shown to improve the learning process as discussed in Section 2.2. However, all of the KBRL approaches assume that the provided knowledge is always correct. This is not a realistic assumption and as RL shifts from tabula-rasa approaches to methods where some heuristic knowledge can be given to an agent, a more clever design needs to be applied to tackle the impact of erroneous knowledge to an agent's learning process.

Specifically, an agent needs to be able to identify parts of the domain knowledge that are erroneous and through the use of knowledge revision methods, rectify its knowledge base to benefit from more accurate shaping.

The most efficient way to introduce belief revision is by the use of an example. A well known example is that of the swan colour problem (Gärdenfors 1992). Consider the following database; or knowledge base:

α : All European swans are white.
β : The bird caught in the trap is a swan.
γ : The bird caught in the trap is from Sweden.
δ : Sweden is a part of Europe.

A logical inference program can easily derive the following fact from the given knowledge base:

ϵ : The bird caught in the trap is white.

What happens if it turns out that the bird caught in the trap is not white but black? The rule $\neg\epsilon$ should now be inserted in the knowledge base but a problem arises; the knowledge base becomes inconsistent. In order to keep the knowledge base consistent, it needs to be revised i.e. certain rules need to be removed to accommodate the new information.

Given the logical representation in the example, revising the knowledge base is not a straightforward procedure. Logical considerations themselves do not provide the means by which the knowledge base should change e.g. if rule α is retracted, then its logical consequences have to be considered as well; a': All European swans except the one caught in the trap are white, a'': All European swans except some of the Swedish are white. What should be kept in the revised knowledge base?

The above example presents the need for belief revision and can be summarised as the following:

- What representations can be used for a knowledge base?
- What is the relation between derived and explicit facts of a knowledge base?
- What governs the choice of what is to be retracted when performing belief revision?

2.3.1 Actions for Belief Change

A belief revision process starts whenever a belief system receives new information that is inconsistent with the current set of beliefs. The goal is to include this new information to the system in such a way that the revised belief base remains consistent. 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 system K; *expansion*, *revision*, *contraction*. A brief informal definition of those three actions is listed below:

1. **Expansion:** A newly arrived information ϕ is added to the current belief base K disregarding consistency. The result is denoted as $K + \phi$.
2. **Revision:** A newly arrived information ϕ, inconsistent with the current belief system K, is added to the system. In order to maintain consistency, some of the current beliefs need to be retracted. The resulting belief base is denoted as $K \dot{+} \phi$.
3. **Contraction:** A rule ϕ, along with its consequences is retracted from the set of beliefs K. To retain logical closure, other rules might need to be retracted. The contracted belief base is denoted as $K \dot{-} \phi$.

Formally defining the expansion action is trivial. It can simply be defined as the logical closure of the belief system K with ϕ:

$$K + \phi = \{\psi : K \cup \{\phi\} \vdash \psi\}$$

In this case logical closure is retained and the belief case K will be consistent, if and only if ϕ is consistent with K.

The types of actions for belief change that this thesis will focus on are the expansion and contraction actions. The knowledge bases that are provided to the agents in this study are all belief bases consisting of independent beliefs. By independent beliefs we refer to those cases where literals in a knowledge base are atomic, i.e. their value does not depend on the values of other beliefs, and the knowledge base does not contain statements that logically follow from other beliefs.

Given that the provided knowledge is based on independent beliefs, it is clear that maintaining consistency in the knowledge base is trivial since it is guaranteed. As a result, performing the expansion action will simply require adding a new belief in the database while performing the contraction action will simply be deleting a belief from the database.

2.3.2 Representing a belief system

The most common approach of representing a belief system is using *sentences* or *propositions* to model beliefs. The chosen method in this thesis is modelling belief sets. Belief sets the simplest way of modelling beliefs. A belief state in this case is a set of sentences K, from a logical language L in which:

If K logically entails ψ, then $\psi \in K$.

K is a logically closed set and contains the sentences that are accepted in the modelled state i.e. if $\phi \in K$ then ϕ is accepted in K; if $\neg\phi \in K$, then ϕ is rejected in K.

There are other ways of modelling beliefs such as belief bases or the possible worlds model but are irrelevant to this study. Choosing the right representation is often a design decision that is largely dictated by the domain at hand. Using belief sets is a good solution when dealing with the domains that will be used for evaluation in this thesis but extending to other classes of domains might require reconsideration of the belief representation system.

2.3.3 The AGM Postulates

Rationality postulates are rules that the revision process should adhere to when deciding how revision will be executed. One of the first, and most popular, formulation of rationality postulates are the AGM postulates (Alchourrón et al. 1985). The AGM postulates follow the concept of information economy i.e. when performing revision or contraction, the beliefs that are to be retracted should be kept to a minimum. Rationality postulates do not themselves provide the means of choosing the minimum amount of information to discard, but work in conjunction with the chosen constructive models some of which are presented later in subsection 2.3.4. Obviously more than one set of rationality postulates exist but it is worth mentioning the important aspects of the AGM postulates in order to illustrate the basic principles of a belief revision process.

It is assumed that the belief representation is that of belief sets. K represents a belief set, and L is a logical language which is closed under logical consequences.

1. For any sentence ϕ and any belief set K,$K\dot{+}\phi$ is a belief set.

2. $\phi \in K\dot{+}\phi$ follows from ϕ being accepted in K.

3. A revision process makes sense when the new information ϕ contradicts the current set of beliefs. If $\neg\phi \notin K$ then a revision is a simple expansion process. The following two postulates are concerned with revision.
   $K\dot{+}\phi \subseteq K + \phi$.
   If $\neg\phi \notin K$, then $K + \phi \subseteq K\dot{+}\phi$.

4. $K\dot{+}\phi = K_\perp$ iff $\vdash \neg\phi$.
   A revision process should produce a new consistent belief set unless ϕ is logically impossible.

5. If $\vdash \phi \Leftrightarrow \psi$, then $K\dot{+}\phi = K\dot{+}\psi$.
   Belief revision should work on the knowledge level, ergo logically equivalent sentences should lead to the same revisions.

The postulates for contraction are similar to those of revision but will not be listed since the intent is to only show the basics of revising a knowledge base. It is however important to mention that a revision process can be expressed as a contraction, and vice versa. The *Levi identity* shows that a revision process is in fact a contraction, followed by an expansion,

$$K\dot{+}\phi = (K\dot{-}\neg\phi) + \phi.$$

The *Harper identity* shows that a sentence ϕ is accepted in a contraction if and only if it is accepted in K and in $K\dot{+}\neg\phi$,

$$K\dot{-}\phi = K \cap K\dot{+}\neg\phi$$

The two identities are very interesting especially from a computational point of view since, revision and contraction being interchangeable, a method for revision can automatically handle contractions as well.

As mentioned previously, the belief bases in this thesis consist of independent belief. Therefore the rationality postulates presented in this section regarding revision are not relevant in their entirety. However, they are very important when the agent is not provided with independent beliefs but a knowledge base that consists of multiple contradictions in the face of new information. The revision process in that case should take into account the AGM postulates in order to maintain a consistent belief base.

2.3.4 Models of Belief Revision

The models of belief revision present the mechanism that chooses what information is to be kept, or retracted during a contraction or revision operation. The models can vary depending on the designer's goals. There is no single construction that can be used in all cases, but the mechanism to be used is dictated by the environment where an agent is modelled e.g. a probabilistic environment would require a mechanism that can efficiently handle uncertainty.

Epistemic entrenchment

Epistemic entrenchment is the method that will be later used for evaluation and is presented here is detail. Consider a consistent belief base K. Since K is consistent, every belief in the set is either accepted or is a fact. When performing certain tasks e.g. planning, not all beliefs have the same value; some beliefs might be more important than other for decision making. The main idea of this model is to retract the beliefs which hold the lowest value or *epistemic entrenchment*, when performing revision or contraction. As an example consider the sentences ϕ and ψ of a logical language L. An ordering of $\psi \leq \phi$ would mean that in the current belief set, ϕ holds a higher or equal value of epistemic entrenchment compared to ψ. The ordering over beliefs can vary depending on the goals the designer is trying to achieve e.g. it can be based on possibility theory, or an ordering might be chosen according to the recency of information.

For example consider the database that was presented at the beginning of this section:
α : All European swans are white.
β : The bird caught in the trap is a swan.
γ : The bird caught in the trap is from Sweden.
δ : Sweden is a part of Europe.

As mentioned previously in this example a logical inference program can derive the following fact from the given knowledge base:

ϵ : The bird caught in the trap is white.

If however the bird caught in the trap is not white but black, what should be retracted? Let's assume that this system uses the epistemic entrenchment model to perform revision or contraction and the ordering over beliefs is based on recency. In this example let $\alpha < \beta < \gamma < \delta$ be the ordering of beliefs. When the system receives information that the bird caught in the trap is black it must choose what to retract in its database. The ordering over beliefs maintains that belief α hold the lowest epistemic entrenchment and will thus be removed, contracted, from the database to accommodate the new information ϵ which states that the bird caught in the trap is black. This ensures that the new information is added in the database while maintaining consistency.

There are various other methods when deciding on a model for belief change such as autonomous belief revision (Galliers 1992), conditionals (Kern-Isberner 2001) and more and like the belief representation, choosing the right method is a design decision that is based on the nature of the domain. For the domains presented in this thesis epistemic entrenchment is a method that covers our needs.

2.3.5 Current research

Many different branches have evolved in knowledge revision and some of the most active fields are those of iterated belief revision and agents that act in environments where the changes in the world are a result of the agent's action (Hunter and Delgrande 2011); this setting resembles the RL scenario that was discussed in 2.1. Conditionals and non-monotonic reasoning in scenarios of computer vision (Leopold et al. 2008; Kern-Isberner 2001), and merging, especially in semantic web. Belief revision is an exciting field however for the purpose of this thesis, it is going to be treated as a tool rather than a research topic.

2.4 Summary

All KBRL approaches presented in this chapter can be utilised to increase an agent's learning rate. Incorporating expert domain knowledge is therefore vital in order to scale to larger and more complex domains.

However, all approaches assume perfect information is order to improve performance i.e. the domain expert knowledge provided to the agent is always assumed to be correct. This is not a realistic assumption as in many cases the knowledge provided can be incorrect or incomplete, or both. In order to acquire the full benefits of KBRL methods there needs to be a clever mechanism to handle those cases.

The remainder of this thesis presents my study on utilising knowledge revision methods in conjunction with KBRL, and more specifically reward shaping. It documents the problems of agents being guided by erroneous knowledge and how they can be overcome by the use of knowledge revision algorithms.

CHAPTER 3

Plan-Based Reward Shaping with Knowledge Revision

This chapter presents the adverse effects of erroneous domain knowledge on agents utilising plan-based reward shaping and how the use of knowledge revision principles can help the agents learn a better policy quicker. Alternative approaches have been developed that can revise knowledge but not within the context of potential-based reward shaping in RL (Leopold et al. 2008; Maclin et al. 2007; Kunapuli et al. 2011). This chapter 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, now correct, knowledge to guide the RL process.

The developed knowledge revision algorithms are empirically evaluated in two different domains: a grid-world flag collection domain that was first presented in the original work on plan-based reward shaping (Grześ and Kudenko 2008), and a very popular real-time strategy game (RTS) developed by Blizzard, StarCraft: Broodwar (SC:BW). It is demonstrated that adding knowledge revision capabilities to a RL agent receiving plan-based shaping can improve its performance, compared to an agent without knowledge revision, when both agents are provided with wrong domain knowledge.

3.1 Plan-Based Reward Shaping Revisited

As discussed in Section 2.2 plan-based reward shaping is a semi-automated method for imparting knowledge to a RL agent through the use of a STRIPS plan. STRIPS is a highly popular and widely used and studied formalism used to express automated planning instances. It is easy to set up and therefore serves as a useful method for setting up reward functions for new environments when using plan-based reward shaping. The STRIPS formalism is briefly explained here and the reader is referred to (Fikes and Nilsson 1972) for further details.

The description of the planning problem in STRIPS includes the operators, actions, which specify the behaviour of the system, and start and goal states (Fikes and Nilsson 1972). Actions have a set of preconditions which have to be satisfied for the action to be executable, and a set of effects which are made true or false by executing the action. 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. Note that one step in the plan will map to many low level states. Therefore, the agent must learn how to execute this plan at the low level.

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

$$\Phi(s) = CurrentStepInPlan * \omega \tag{3.1}$$

where $CurrentStepInPlan$ is the corresponding state in the state-based representation of the agent's plan and ω is a scaling factor. Specifically, $CurrentStepInPlan$ is a number representing the position of a particular state in the agent's state-based plan. The scaling factor ω affects how likely the agents are to follow the heuristic knowledge. When comparing to other agents with different plan lengths, ω helps contain a constant maximum and thus ensure fair comparison.

To better illustrate the inner workings of plan-based reward shaping, the flag collection domain is presented here which will be used for evaluation later and provide some examples of how a potential function can be designed for such a domain.

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 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.

At a less abstract level, the maze can be thought of as a house with doors between rooms. Each room might, or might not contain a flag and it is up to the agent to find where the flags

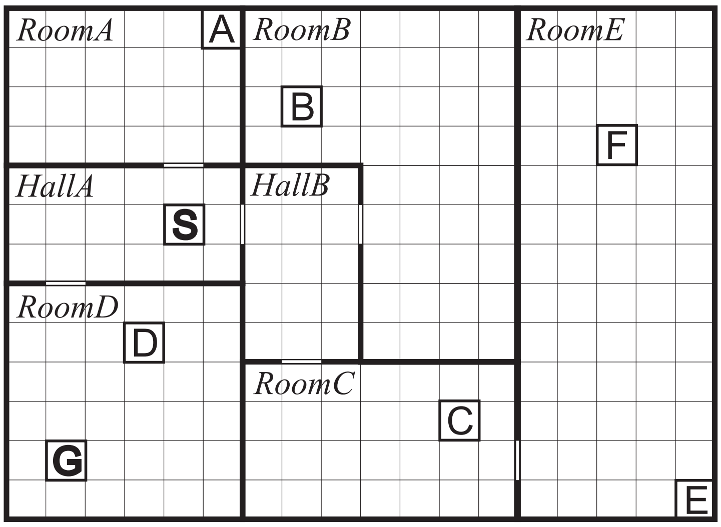


Figure 3.1: Flag Collection Domain.

```
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 )
```

Figure 3.2: STRIPS Plan in the Flag Collection Domain

are located. Figure 3.1 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, the expected STRIPS plan is given in Figure 3.2.

As mentioned previously the plan needs to be transformed into a state-based representation for it to be usable by the RL agent. This process can be automated by extracting the pre- and post-conditions of the actions in the plan in order to form the state-based representation. The corres-

```
0 robot_in_hallA
1 robot_in_hallB
2 robot_in_roomC
3 robot_in_roomC taken_flagC
4 robot_in_roomE taken_flagC
5 robot_in_roomE taken_flagC taken_flagE taken_flagF
6 robot_in_roomC taken_flagC taken_flagE taken_flagF
7 robot_in_hallB taken_flagC taken_flagE taken_flagF
8 robot_in_roomB taken_flagC taken_flagE taken_flagF
9 robot_in_roomB taken_flagC taken_flagE taken_flagF taken_flagB
10 robot_in_hallB taken_flagC taken_flagE taken_flagF taken_flagB
11 robot_in_hallA taken_flagC taken_flagE taken_flagF taken_flagB
12 robot_in_roomA taken_flagC taken_flagE taken_flagF taken_flagB
13 robot_in_roomA taken_flagC taken_flagE taken_flagF taken_flagB
   taken_flagA
14 robot_in_hallA taken_flagC taken_flagE taken_flagF taken_flagB
   taken_flagA
15 robot_in_roomD taken_flagC taken_flagE taken_flagF taken_flagB
   taken_flagA
16 robot_in_roomD taken_flagC taken_flagE taken_flagF taken_flagB
   taken_flagA taken_flagD
```

Figure 3.3: State-Based Plan in the Flag Collection Domain

ponding state-based plan used for shaping is given in Figure 3.3 with the $CurrentStepInPlan$ used by Equation 3.1 noted in the left hand column.

3.2 Knowledge Revision Algorithms

As with most reward shaping algorithms for RL, in plan-based reward shaping (Grześ and Kudenko 2008) there was no mechanism in place to deal with erroneous knowledge. If an erroneous plan is used the agent is misguided throughout the course of an experiment and this can lead to undesired behaviour; long convergence time and poor quality in terms of total reward.

This section presents the revision algorithms that were developed in order to overcome the problems of erroneous domain knowledge in plan-based reward shaping. The types of erroneous knowledge can take the form of 1) incorrect knowledge e.g. the provided plan contains steps which the agent cannot achieve, and 2) incomplete knowledge e.g. the provided plan is missing important steps which the agent should achieve. Examples instantiated to the flag collection domain are provided in order to explain the concepts of the presented algorithms. In addition a few assumptions are presented that must be made for knowledge revision to be feasible in plan-based reward shaping.

Assumptions

To implement plan-based reward shaping with knowledge revision the following assumptions will be made:

- 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 2.6). 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. In addition, we assume deterministic transitions.

- The transition and reward functions are not known beforehand and the environment is not fully observable.

Domains limited by only these assumptions include many domains typically used throughout RL literature. The chosen domain allows the agent's behaviour to be efficiently extracted and analysed, thus providing useful insight especially when dealing with novel approaches. Plan-based reward shaping with knowledge revision is not, however, limited to this environment and could be applied to any problem domain that matches these assumptions. These assumption will later be relaxed in Chapter 4 which focuses on abstract MDP reward shaping with knowledge revision.

3.2.1 Overcoming incorrect knowledge

Identifying incorrect knowledge

At each time step t the agent performs a low level action a and traverses to a different state s' while receiving guidance by a STRIPS plan. Since the agent is performing low level actions it can gather information about the environment which then can be compared against the provided knowledge and as a result discover potential errors. Algorithm 1 shows the generic method of identifying incorrect knowledge.

The input to the algorithm is a set of preconditions of all the plan states which we will call K and each precondition in this set which we will call p. A plan state is defined as a step in the provided plan, e.g. `3 robot_in_roomC taken_flagC` as shown in Figure 3.3, and since it is based on STRIPS planning, the preconditions of each plan step can be easily extracted from the planner to form the set K. Note that the preconditions that will be included in this set can be a designer's decision e.g. one might choose to include all the preconditions associated with a plan

Algorithm 1 Incorrect Knowledge Identification Algorithm.

```
  form a set $K$ of preconditions $p$ of each state in the provided plan
  initialise confidence values $Cn(p)$ of each precondition in $K$ to a random number
Input: set of preconditions $K$
  for $episode = 0$ to $max\_number\_of\_episodes$ do
    for $current\_step = 0$ to $max\_number\_of\_steps$ do
      if $p$ marked for verification then
        switch to verification mode
      else
        plan-based reward shaping RL
    /* next step */
    for all $p$ in $K$ do
      /* update the confidence values */
      update $Cn(p)$
      /* check preconditions which need to be marked for verification */
      if $Cn(p) < \epsilon$ then
        mark $p$ for verification
    /* next precondition */
  /* next episode */
Output: set of preconditions marked for verification
```

state, while in another situation choose only those preconditions that are associated with a high probability of being erroneous. Each of the preconditions p in the set is assigned a numerical confidence value denoted as $Cn(p)$ which can be initialised randomly. This process takes place at the start of each experiment and before the agent-environment interaction has begun.

Once this process is over the agent starts its learning process and interacts with the environment. Given that there is a translation of the low level states to the high level states in the plan, as mentioned in the assumptions, the agent can realise whether a plan state has been encountered or not during the learning process. At the end of the episode, the preconditions p in the set of preconditions K are updated to reflect if their associated plan state was encountered or not. How the values are updated can be decided depending on the domain.

Having updated all of the plan states preconditions each one is then compared against a verification threshold ϵ. Those preconditions that were found to be less than the specified verification threshold are then marked for verification. This method of ranking beliefs is similar to epistemic entrenchment in belief revision that was presented in section 2.3.4. The value of ϵ can be set high or low depending on how often the designer of the system wants verification to take place. Note however that there must be a correlation between the value of ϵ and the confidence values of the preconditions $Cn(p)$ e.g. if $Cn(p)$ is based on percentages, so should ϵ. In addition, when initialising the values of the preconditions care must be taken not to be lower than the verification threshold as this will cause all the preconditions to be marked for verification.

The output of this algorithm is a set of preconditions that have been marked for verification. When the agent finds itself in a situation where a precondition has been marked for verification, it switches its mode of operation to verification.

We illustrate this algorithm with an instantiation to the flag collection domain. This study evaluates the algorithm using erroneous knowledge which involves the presence of flags. The same techniques however can be used when dealing with erroneous knowledge in terms of rooms and connections between them.

Let P be the set of all preconditions p of those actions which achieve a given plan state. Let the set $K \subseteq P$ contain those preconditions which refer to the presence of flags in the plan e.g. `flagA_in_roomA`, and $Cn(p)$ the confidence value associated with each precondition $p \in K$ and ϵ the verification threshold.

At the start of each experiment the agent receives a set, K, of all the preconditions p which refer to the presence of flags. These preconditions are then assigned a confidence value, Cn. The confidence value of each precondition p 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 associated with p up to the current episode, and $failures$ the times it failed to do so. Given that the confidence value is computed at the end of each episode it can be initialised randomly but must be strictly greater than the chosen verification threshold. If the confidence value of a precondition drops below the verification threshold, ϵ, that precondition is marked for verification.

Knowledge verification

When a precondition p is marked for verification the agent's mode of operation changes to perform actions in order to try and verify the existence of the flag associated with p. Algorithm 2 shows the generic method of verifying incorrect knowledge. In this thesis depth first search (DFS) can be used for knowledge verification, however the method by which the verification of a precondition can be achieved can vary depending on the environment. For instance, if the agent was a robotic guard moving within a building, a search based on the sensors of the robot would be more suitable.

The input to this algorithm is a set of preconditions that have been marked for verification. When the agent is in a situation where a precondition has been marked it switches to verification mode, which in this case is a DFS of the low level environment. The bounds of the DFS can be set according to the high level knowledge that is being provided e.g. in a maze it can be specified as the north-west physical boundary.

When the agent starts the DFS its current position, node, is saved in the graph G along with the available actions from the position representing the edges. The agent then chooses to expand a random unexpanded edge and moves to a new node. This node is then added to the graph and

Algorithm 2 Knowledge Verification Algorithm.

```
Input: set of preconditions marked for verification
  initialise empty graph $G$
  get state $s$
  add $s$ to $G$
  get precondition $p$
  if all nodes in the graph are marked as $fully\ expanded$ then
    mark $p$ for revision
    stop search
    $break$
  if $s$ is not present in the graph then
    add $s$ and available actions $a$ as node and edges in the graph
  if all edges of current node have been expanded then
    mark node as $fully\ expanded$
    move to a node with unexpanded edges
    $break$
  expand random unexpanded edge
  mark edge as expanded
  if $p$ has been verified then
    reset $Cn(p)$
    stop search
Output: set of preconditions marked for revision
```

the edge is marked as expanded. When a node has had all of its edges expanded it is marked as visited and the process continues to unexpanded edges and unvisited nodes.

If all of the nodes have been visited i.e. all of the edges have been expanded, and the precondition in question has not been verified, then it is marked for revision. If at any point during the DFS the precondition is verified, then it's confidence value is reset.

The output of the algorithm is a set or preconditions that have been marked for revision. In order to make this process clearer this algorithm is also explained instantiated to the flag-collection domain.

To verify the existence of the flag associated with p the agent performs a DFS of the low-level state space within the bounds of the high-level abstract state of the plan the flag appears in. 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.

After 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.

The bounds used in DFS are easy to extract given that there is a direct translation from the high level to the low level environment the agent is acting in. For example, assume that `flagA` is under examination which is present in `roomA`. The bounds of `roomA` are the corresponding coordinates in the low level grid states. If we assume that roomA is a 6×6 grid then in the worst case scenario, if the flag is not found, the DFS will be performed on a graph $G_{6,6}$ that contains 36 nodes and 60 edges.

In order to be fair when comparing with other approaches, each edge expanded while the agent performs DFS takes a time step to complete i.e. it counts the same as one action taken.

The search finishes once the agent has either found the flag or all of the nodes that were added to the graph have been marked as *fully expanded*. If found, the confidence value of the precondition p which refers to that flag is reset and the agent returns to normal operation. If not, the agent returns to normal operation but the precondition is marked for revision.

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.

Revising the knowledge

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.

Depending on the details of the domain, one of these three actions will be performed when the agent has reached the point of having to revise its knowledge. In this thesis we explore those cases where the errant knowledge the agent has to deal with can be either verified or not verified in simulation. As a result, the agent will only have to perform a contraction when dealing with incorrect knowledge. Furthermore, we explore the cases where the knowledge base contains independent beliefs i.e. beliefs that are considered to be true, are not depending on other belief to also be true or false and vice-versa. This further simplifies the problem of contraction to a simple deletion of a literal from the knowledge base.

As an example consider the flag-collection domain. 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. 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. As a result revising the knowledge base in this case requires the removal of a literal e.g. flagA_in_roomA, from the initial conditions of the plan. The revised knowledge base is then used to compute a more accurate plan.

It is worth noting that since the knowledge the agent is provided with is refined, by the use of belief revision, the potentials are not static but dynamic. As mentioned previously, even when using a dynamic potential function, the theoretical guarantees of potential-based reward shaping are maintained provided the potential of a state is evaluated at the time the state is entered and

```
0 robot_in(hallA)
1 robot_in(hallA) taken(flagH)
2 robot_in(roomD) taken(flagH)
3 robot_in(roomD) taken(flagH) taken(flagD)
```

Figure 3.4: Example Incorrect Plan in the Flag Collection Domain

```
0 robot_in(hallA)
1 robot_in(roomD)
2 robot_in(roomD) taken(flagD)
```

Figure 3.5: Example Correct Plan in the Flag Collection Domain

used in both the potential calculation on entering and exiting the state. More information on dynamic potential-based reward shaping can be found in (Devlin and Kudenko 2012a).

To illustrate the use of this method consider a domain similar to that shown in Figure 3.1 which contains one flag, `flagD` in `roomD`. The agent is provided with the plan shown in Figure 3.4. This plan contains an extra flag which is not present in the simulator, `flagH` in `hallA`. According to the plan the agent starts at `hallA` and has to collect `flagH` and `flagD` and reach the goal state in `roomD`.

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. Since `flagH` does not appear in the simulator its confidence value will be equal to 0 and the flag will be marked for verification.

During the next episode the agent will switch into verification mode for `flagH`. At this point the agent will perform a DFS within the bounds of `hallA` to try and satisfy `flagH`. The search will reveal that `taken(flagH)` cannot be satisfied and as a result `flagH` will be marked for revision. When the episode ends the knowledge base will be contracted to remove `flagH` and a new plan will be computed. The new plan is shown in Figure 3.5.

3.2.2 Overcoming incomplete knowledge

Identifying incomplete knowledge

As mentioned previously an agent can identify potential errors in the provided plan through continuous interaction with the environment. This section presents the case where the provided plan is incomplete. Algorithm 3 shows the generic method of identifying incomplete knowledge. We illustrate this algorithm with an instantiation to the flag collection domain.

The input to the algorithm is a set of preconditions of all the plan states which we will call K and each precondition in this set which we will call p. A plan state is defined as a step in the provided plan, e.g. `3 robot_in_roomC taken_flagC` as shown in Figure 3.3, and since it is based on STRIPS planning, the preconditions of each plan step can be easily extracted from the planner to form the set K as it was mentioned in the case of incorrect knowledge. Like before, the

Algorithm 3 Incomplete Knowledge Identification Algorithm.

```
  form a set $K$ of preconditions $p$ of each state in the provided plan
Input: set of preconditions $K$
  for $episode = 0$ to $max\_number\_of\_episodes$ do
    for $current\_step = 0$ to $max\_number\_of\_steps$ do
      plan-based reward shaping RL
    /* next step */
    /* check preconditions */
    for all $p_s$ in $S$ do
      if $p_s \notin K$ then
        mark precondition $p_s$ for revision
    /* next precondition */
  /* next episode */
Output: set of preconditions not present in K and marked for revision
```

preconditions included in this list is a design decision and it can either be all of the preconditions that are associated with a plan state or a selection of those. This process takes place before the learning process has begun.

Once this process is completed the agent can start interacting with the environment and try to satisfy its goals while receiving guidance from a STRIPS plan. Since there is a direct translation of the low level states the agent is acting on to the high level states, the agent can realise whether certain states are in its knowledge base or not.

The agent can extract a set of the preconditions p_s that are associated with all of the states it visited during an episode called S. At the end of each episode, this set is compared against the initial set of preconditions K. If any of the preconditions in S are not present in K they are marked for revision.

The output of the algorithm is a set of preconditions which are not present in the current knowledge base of the agent and which have been marked for revision.

To further illustrate this process consider an instantiation to the flag-collection domain. Let L be the set of all preconditions p of those actions which achieve a given plan state. We define the set $K \subseteq L$ to contain those preconditions which refer to the presence of flags in the plan and the set S to contain all the preconditions p_s which refer to the presence of flags in the simulator.

Specifically, at the start of each experiment the provided plan is used in order to extract a set, K, of all the preconditions which refer to the presence of flags. When an episode ends the flags the agent collected, set S, are compared against the flags which appear in the plan. If the agent is found to have collected extra flags which do not appear in the plan, and by extension in the knowledge base, those flags are marked for revision. Note that there is no verification in the case of incomplete knowledge.

```
0 robot_in(hallA)
1 robot_in(roomD)
2 robot_in(roomD) taken(flagD)
```

Figure 3.6: Example Incorrect Plan in the Flag Collection Domain

```
0 robot_in(hallA)
1 robot_in(hallA) taken(flagA)
2 robot_in(roomD) taken(flagA)
3 robot_in(roomD) taken (flagA) taken(flagD)
```

Figure 3.7: Example Correct Plan in the Flag Collection Domain

Revising the knowledge

As mentioned previously there are three actions for belief change: expansion, revision and contraction. Once the agent reaches the point of having to revise it must perform one of those actions. In the case of incomplete knowledge the agent will only need to perform an expansion of its knowledge base to include all those preconditions that were marked for revision. Given that the agent is dealing with independent beliefs then all the precondition that were encountered and were not already in the agent's knowledge base, are simply added since maintaining consistency is guaranteed.

Instantiated to the flag collection domain an easier explanation, new information comes in the form of flags which are present in the simulator, but not in the provided plan. Revising the knowledge to include new information in this case requires an expansion. As a result, the initial conditions of the plan are expanded to include new literals e.g.`flagA_in_roomA`. Since the agent is dealing with independent beliefs it does not need to worry about conflicts and there fore expansion equals addition. 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 3.1 which contains two flags, flagA in `hallA` and `flagD` in `roomD`. The agent is provided with the plan shown in Figure 3.6. This plan contains only one of the flags which are present in the simulator, `flagD` in `roomD`. According to the plan the agent starts at `hallA` and has to collect `flagD` and reach the goal state in `roomD`.

Let's assume that the agent, through exploration, manages to pick up `flagA` during an episode. When the episode ends the agent's collected flags will be compared against the flags which appear in the plan. Since `flagA` does not appear in the plan, it will be marked for revision.

The knowledge base will then be expanded to include `flagA` and a new plan will be computed using the revised knowledge base. The new plan is shown in Figure 3.7.

3.2.3 Evaluation in the Flag Collection domain

The algorithms are evaluated by using agents that are provided with erroneous knowledge. Specifically the agents are given different instances of wrong knowledge: 1) incorrect knowledge, 2) incomplete knowledge and 3) a combination of both incorrect and incomplete knowledge.

All agents implemented SARSA with $\epsilon-$greedy action selection and eligibility traces (Sutton and Barto 1998). For all experiments, the agents' parameters were set such that $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon = 0.1$ and eligibility traces $\lambda = 0.4$.

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 to be successful and hold the same theoretical guarantees with Q-learning, RMax and any action selection method that chooses actions based on relative difference and not absolute magnitude as well as without eligibility traces (Asmuth et al. 2008; Ng et al. 1999; Devlin et al. 2011). SARSA with eligibility traces was our algorithm of choice since there was legacy code that was already set-up and optimised for this domain within our lab, but any other method could be used without sacrificing performance.

In all experiments, the scaling factor of Equation 2.16 was set to:

$$\omega = MaxReward/NumStepsInPlan \tag{3.2}$$

As the scaling factor affects how likely the agents are 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 30 times for each instance of the erroneous knowledge. The agent with knowledge revision is compared to an agent without knowledge revision when both agent are provided with the same erroneous 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 agents needs to perform to complete an episode. For clarity all the graphs only display results up to 2500 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.

Incorrect knowledge

In this setting the agents are provided with a plan which contains extra flags which do not appear in the simulation. Figures 3.8 and 3.10 present the averaged results when providing a plan with two and three extra flags. In addition the steps taken to complete an episode are included in

Figures 3.9 and 3.11 to better show the convergence speed. An mentioned previously all steps taken during knowledge verification have been accounted for since any action during DFS costs one time step.

The plan-based RL agent without knowledge revision is not able to overcome the incorrect knowledge and performs sub-optimally throughout the duration of the experiments. The agent without reward shaping does not benefit from domain knowledge and as a result has the worst performance.

The agent with knowledge revision manages to identify the flaws in the plan and quickly rectify its knowledge. During the experiments, at any point the agent needed to re-plan the output was saved so that it is easy to check what revisions are taking place and when. Close examination showed that the agent started revising its knowledge around the 50_{th} episode and had managed to build a correct knowledge by the 400_{th} episode. 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.

It is worth noting that when the agent decides to revise its knowledge, it is not possible to revise beliefs that are correct i.e. information that is present in the simulation. The reason is the deterministic nature of the environments that are examined in this chapter. Consider for example that the agent cannot pick up `flagA` which is present in the simulation in `roomA` because exploration has not yet led it to that flag. After a few episodes this flag will be marked for verification and when the agent enters `roomA` it will perform a DFS. The DFS will reveal that the flag exists at that place and its confidence value will be reset. In contrast, if `flagA` was not present in the simulation, DFS would not reveal its existence and it would be deleted from the knowledge base. In both situations, the information in question is completely deterministic and as a result it is not possible for the agent to revise information that is correct.

It should be pointed out that if the agents were let to run an infinite amount of time, they would eventually converge to the same optimal policy, as the methods used preserve the RL theoretical convergence guarantees.

In addition, experiments were conducted with varying number of incorrect flags ranging from 3 up to 8. Varying the number of extra flags did not exhibit any difference in behaviour and the same results where witnessed with the agent using knowledge revision outperforming the agent without. All the agent parameters were set to $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon = 0.1$ and eligibility traces $\lambda = 0.4$, as the agents reported here, and each experiment lasted for 50000 episodes and was repeated 30 times.

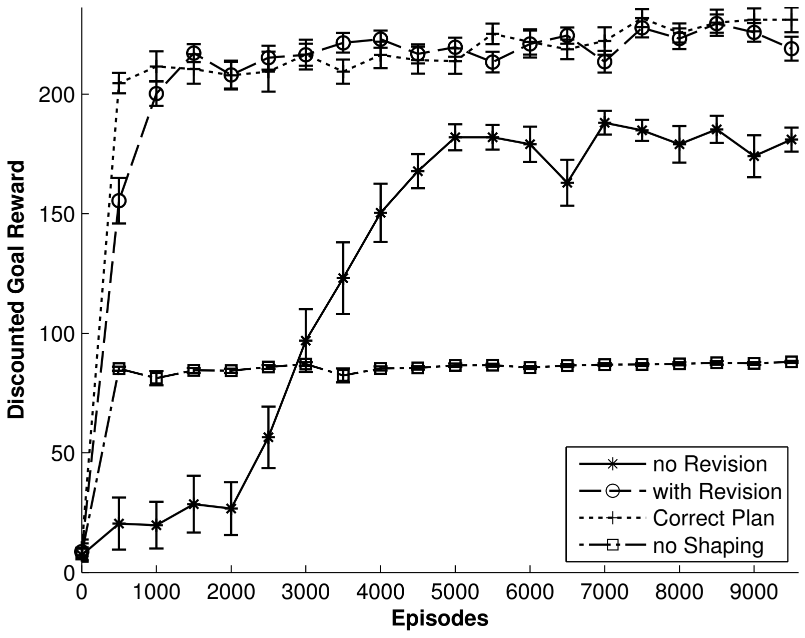


Figure 3.8: Incorrect knowledge. Extra flags: 2

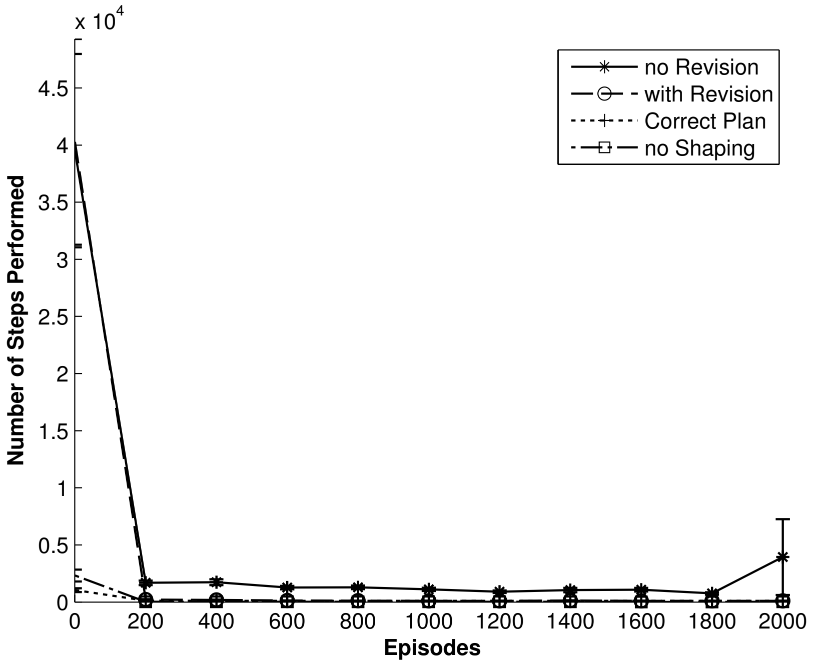


Figure 3.9: Incorrect knowledge. Steps taken, Extra Flags: 2

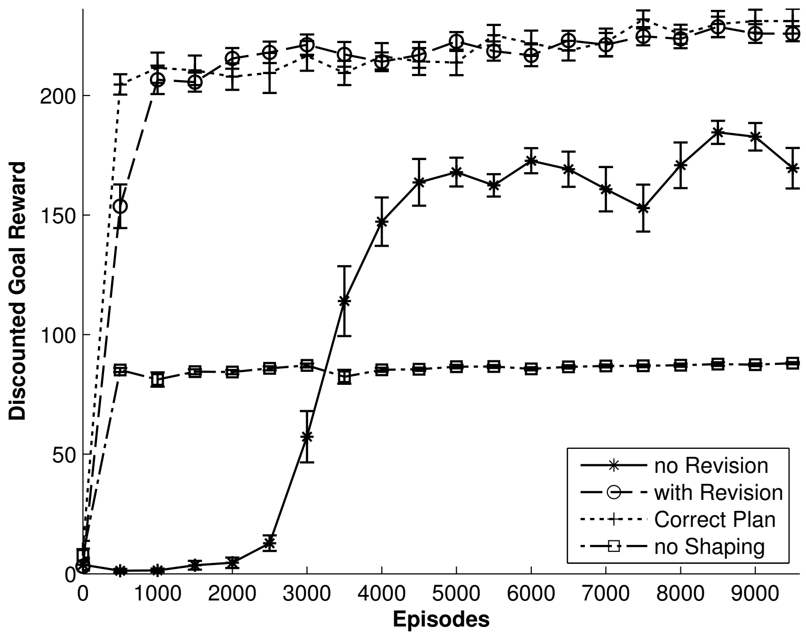


Figure 3.10: Incorrect knowledge. Extra flags: 3

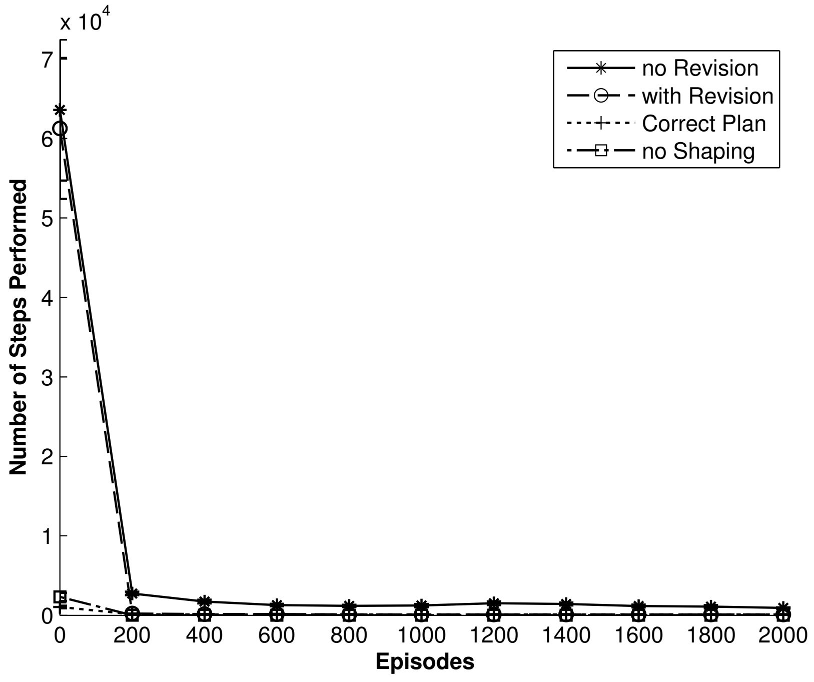


Figure 3.11: Incorrect knowledge. Steps taken, Extra flags: 3

Incomplete knowledge

In the setting of incomplete knowledge the agents are provided with a plan which is missing flags that are present in the simulation. Figures 3.12 and 3.14 present the averaged results when providing a plan with two and three missing flags as well as the steps taken shown in Figures 3.13 and 3.15.

It is clear that the plan-based RL agent without knowledge revision struggles to overcome the incomplete plan and performs sub-optimally throughout the course of the experiment. Once again the agent without reward shaping has the worst performance. The agent using knowledge revision manages very early on in the experiment to identify the flags which are missing from the plan and update its knowledge base. As a result it reaches a performance similar to the agent receiving the correct plan within a few hundred episodes.

Closely examining the planner output during the re-planning periods of the agent showed that the agent started revising its knowledge to include new information at the 100_{th} episode and had build up a correct knowledge base by the 600_{th} episode. There were cases reported in which the agent did not manage to discover all the parts that were missing from its knowledge base, and therefore did not receive guidance from a correct plan, but those cases were only 450 in a total of 50000 experiments.

Combination of incorrect and incomplete knowledge

In this setting the agents are provided with a plan which contains extra flags which are not present in the simulation, as well as missing flags which the agent should be able to pick up. The averaged results are presented in Figures 3.16 and 3.17 showing the discounted goal reward and the number of steps that the agents performed in order to complete an episode.

As expected, the results show that the plan-based RL agent without knowledge revision cannot overcome the difficulties posed by the combination of incorrect and incomplete knowledge and performs sub-optimally throughout the experiments. The performance of the agent without knowledge revision however does not seem to be heavily impacted when comparing to the incorrect and incomplete cases and still performs better than the agent without shaping.

Furthermore, when comparing the number of steps each agent performs in order to complete an episode, the graphs show that the agent with knowledge revision achieves a behaviour similar to the agent using the correct plan very quickly. Whereas, the agent without knowledge revision spends a lot of time performing sub-optimal actions early in the experiment. The agent without shaping fails to pick up all the flags and moves fast to the goal state, hence the low number of steps which is also witnessed in the incomplete knowledge experiment.

These empirical results demonstrate that, when a RL agent using SARSA is provided with incorrect, incomplete and a combination of incorrect and incomplete knowledge, knowledge revision allows the agent to incorporate its experiences into the provided knowledge base and thus benefit from more accurate plans. In all experiments the agent using knowledge revision man-

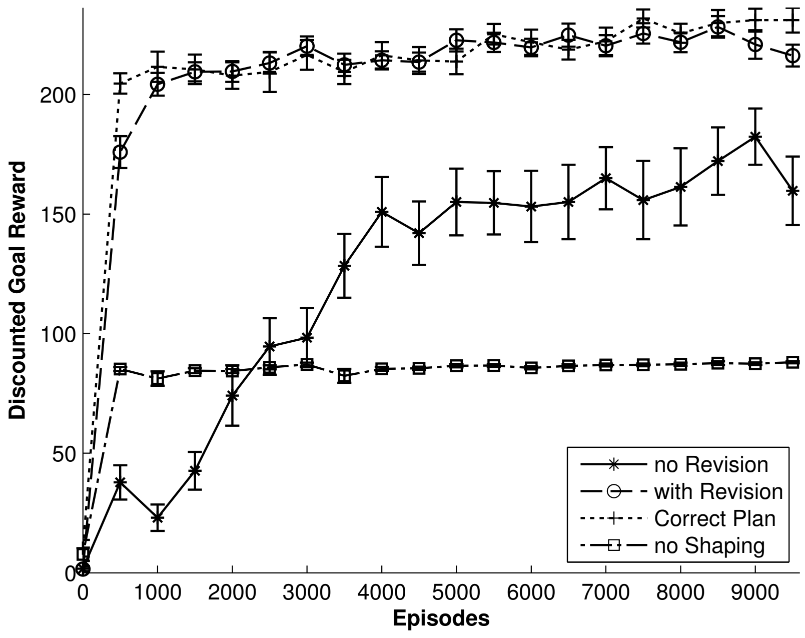


Figure 3.12: Incomplete knowledge. Missing flags: 2

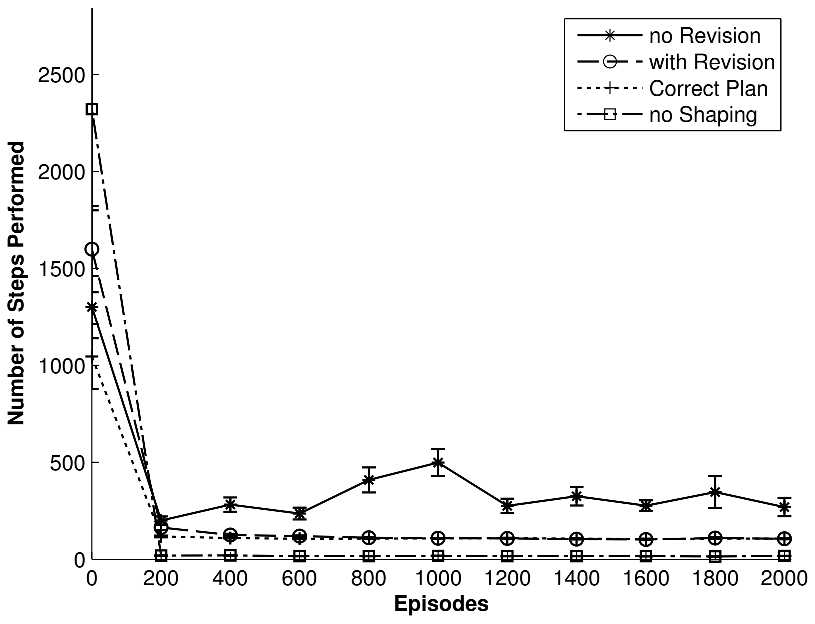


Figure 3.13: Incomplete knowledge. Steps taken, Missing flags: 2

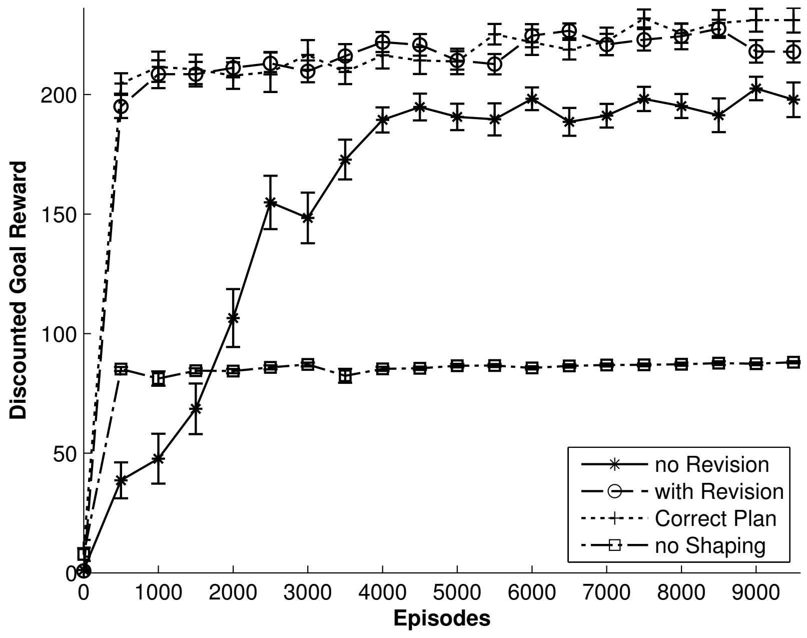


Figure 3.14: Incomplete knowledge. Missing flags: 3

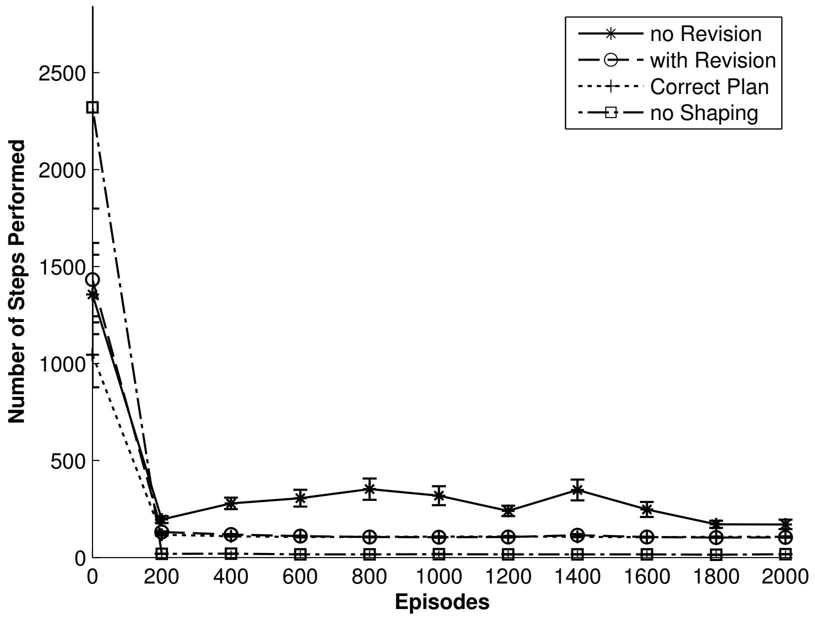


Figure 3.15: Incomplete knowledge. Steps taken, Missing flags: 3

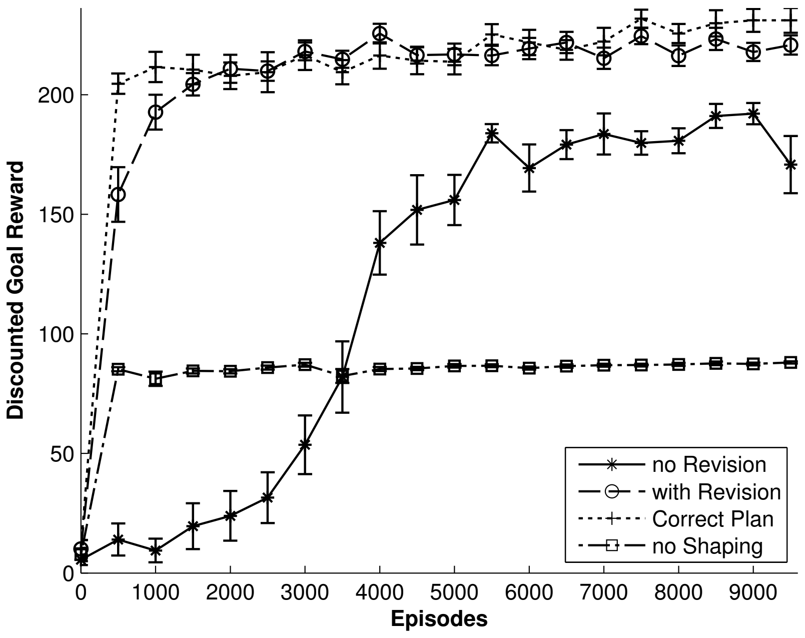


Figure 3.16: Discounted goal reward with a combination of both incorrect and incomplete knowledge

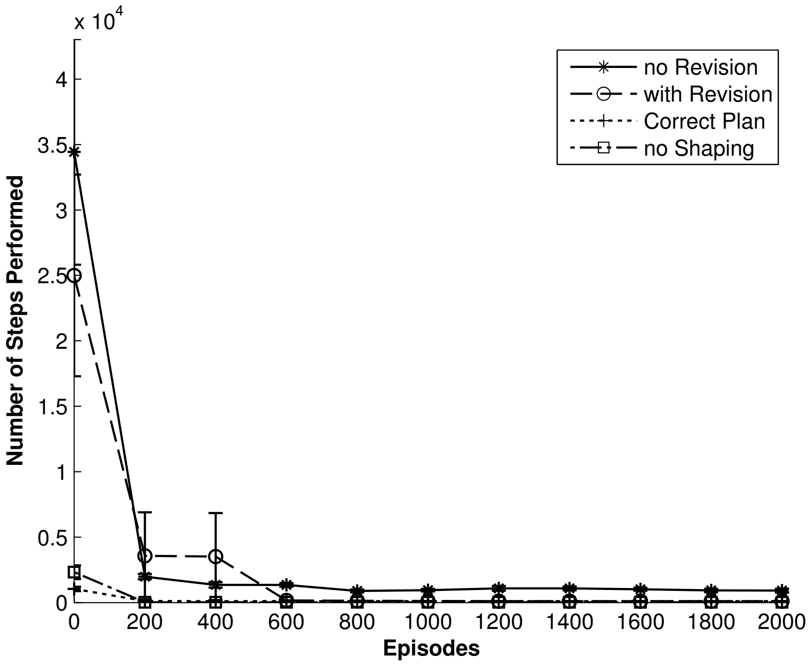


Figure 3.17: Average number of steps the agents' needed to perform in order to complete an episode with a combination of incorrect and incomplete knowledge

aged to outperform the agent without knowledge revision and the agent without reward shaping and achieve a similar performance to the agent which was provided with the correct plan.

3.3 Starcraft: Broodwar

The flag collection domain provides a nice environment in order to analyse and dissect the behaviour of novel algorithms such as these presented in this thesis. Having established that plan-based reward shaping with knowledge revision has the desired effects in the flag collection domain it's effectiveness is evaluated by scaling to a larger more complex environment, Starcraft: Broodwar.

StarCraft is a very popular commercial RTS game which has been used in AI research extensively with numerous competitions running in parallel to major game conferences. This was made possible with the development of the Brood War API (BWAPI) framework which allows integration with the SC:BW game engine.

As with every RTS game, the gameplay revolves around resource management, and building construction. Resource management is very important in every RTS game since it is what allows the players to build advanced units. In SC:BW the two major resources are *vespene gas* and *minerals*. Worker units are tasked with collecting those resources.

Once enough resources have been gathered, the player can decide how to expand. Different build orders allow the production of different units and result in different strategies in order to beat the opponent which can either be another player, or the built in game AI. Many different strategies have been developed throughout the years by players, some focusing on exploration and large armies, while others on speed and efficiency.

3.3.1 StarCraft Scenario

RL has been succesfully used in many games like Backgammon (Tesauro 1994), Tetris (Szita and Lörincz 2006), Unreal Tournament (Smith et al. 2007) and more. However its performance has not been studied extensively in StarCraft. Of particular interest in the context of this research is the application of RL in a small-scale combat scenario in the SC:BW (Wender and Watson 2012). In (Wender and Watson 2012) the scenario involved an overpowered unit (the RL agent), fighting against a group of enemy units spread around the agent. The agent benefited from superior firepower, speed and range when compared to a single enemy unit. The results showed that the agent quickly learnt how to utilize the “hit and run” strategy, and managed to destroy all enemy units in most of the experiments. These results show the use of RL in SC:BW and the authors aim at creating a hybrid agent that would manage the complexity of the entire game.

Despite the promising results, the state-space of the chosen environment is very small and is not representative of the complexity of the game in its entirety. When trying to scale to larger problem domains, a more efficient design is needed, in order to tackle the state-space explosion, than that of straightforward RL. One method that can achieve this is plan-based reward shaping (Grześ and Kudenko 2008).

In order to demonstrate the use of plan-based reward shaping and plan-based reward shaping with knowledge revision, we have chosen to evaluate the agent by scaling to a more complex and qualitatively different scenario. The chosen scenario involves the creation of a RL build-order manager that aims at learning part of a player strategy often used in SC:BW, the *Terran Battlecruiser Rush* strategy. The source of the high level strategy can be expert knowledge provided by players, or could come from a high level planning system. The Battlecruiser rush strategy dictates the clever use of resources in order to build the *Battlecruiser* unit as fast as possible while using worker units as support. The goal of this strategy is to march toward the enemy as soon as the Battlecruiser has been constructed so as to surprise the opponent who will most likely be poorly defended. Although this scenario is focused at the *Terran* race, the method can be adapted to any of the races in the game. A model of the agent's learning goal, state, and action space is presented in Section 3.3.2.

There have been other approaches on using RL in RTS games such as $CLASS_{Q-L}$ (Jaidee and Muñoz-Avila 2012), in which learning occurs for each class of units in the game in order to speed up the learning process, transfer learning (Sharma et al. 2007), which uses a hybrid case-based reasoning/RL approach in order to learn and reuse policies across multiple environments, and more. This study presents how the learning process can be sped up by the use of reward shaping and it is my belief that this method could be used on top of other RL methods like (Jaidee and Muñoz-Avila 2012) and (Sharma et al. 2007) to learn policies faster in large state-spaces.

3.3.2 Design

Start state

The agent is situated at a starting position controlling 1 Space Construction Vehicle (SCV), 1 Refinery (used by the Terran race to collect vespene gas), 1 Command Center (Terran base), 1 Resource Depot (used to increase the amount of minerals and vespene gas the agent can collect as well as the amount of units it is allowed to create) and 6 Mineral Fields spread around the starting area. This configuration creates eighteen slots which can be filled by either worker or combat units i.e. up to 18 units can be trained.

Learning objective

The task that the agent has to learn is that of a strategy often employed by players in SC:BW, the *Battlecruiser rush* strategy. More specifically, this strategy suggests creating one Battlecruiser unit as fast as possible along with several worker units that will act as support for the Battlecruiser, mostly by just being set to repair the damage caused by the opponent. The agent has to learn the optimal building order construction to achieve the creation of the Battlecruiser, while at the same time learn the optimal number of worker units needed in order to have a high enough rate of incoming supplies, to minimise the time to build the Battlecruiser unit. The correct build order, taking into account the constraints of required buildings is *Barracks→Factory→Science Facility→Physics Lab→Starport→Control Tower*. Having constructed these building allows the creation of the Battlecruiser unit.

State-space

Designing the state-space is one of the most important parts in the development of any RL agent. While one can use the raw information contained in the SC:BW environment, it is not practical as the amount of information that the agent will have to deal with is so massive, that it would eventually inhibit learning altogether. Therefore the raw information needs to be transformed into a useful abstraction relevant to the task the agent has to tackle. We have chosen the following abstraction:

- *vespene Gas:* the amount of vespene gas the agent has in its possession partitioned in buckets of 50 i.e. 0-50, 50-100... >400. We have chosen this design because most units and buildings in SC:BW have a cost which increases by 50 or 100 depending on their level. We have chosen to treat the region > 400 as constant since there are very few units that have a cost of 400 supplies to build. Table 3.1 shows the cost of the units that the agent will need to construct and the slots that each unit fills.
- *Minerals:* the amount of minerals the agent has in its possession partitioned in the same manner as vespene gas which was presented previously.
- *Units Built:* the types of units, including buildings, that the agent has under control. These types include the *Factory, Barracks, Starport, Control Tower, Science Facility, Physics Lab* and *Battlecruiser* units. These types are imperative to the creation of the Battlecruiser unit which is the agent's learning task. As a result it is very important information that needs to be included to state-space abstraction.
- *Workers*: the number of worker units the agent has under control. Including this information allows the agent to reason about the optimal number of worker units needed in order to have a supply flow high enough, to allow for the fastest creation of the Battlecruiser.

Action-space

The agent can perform a total of ten actions which can be divided into two categories, resource collection and unit construction. Before moving on to the description of those actions, it is worth noting that any chosen action may take a varying amount of time to complete in the game e.g. collecting supplies might take less time than building a specific unit. As a result we define a RL time-step to be the time it takes for an action to be completed.

- *Collect:* the agent can choose to collect either *minerals*, or *vespene gas*. Choosing any of those actions will result in all the worker units the agent controls to move to the closest

Table 3.1: Unit cost relevant to agent's learning objective.

Unit Type	*Minerals*	*vespene Gas*	*Slots Taken*
Factory	200	100	-
Barracks	150	-	-
Starport	150	100	-
Control Tower	50	50	-
Science Facility	100	150	-
Physics Lab	50	50	-
SCV	50	-	1
Battlecruiser	400	300	6

resource and start collecting supplies. Given that actions to collect gas or minerals are designed to not have a clear end in the game engine i.e. a collect action will continue until a new action is chosen, we set the action as completed after 500 frames. This amounts in a total of two actions for resource collection.

- *Construct:* the agent can choose to build a specific unit. This can be either a building, or a combat/worker unit. The possible units that the agent can choose to construct are *Factory, Barracks, Starport, Control Tower, Science Facility, Physics Lab, SCV worker* and *Battlecruiser*. This amounts in a total of eight actions for construction.

While there are many more actions that an agent can perform when playing the entire game, they are irrelevant to the task at hand and therefore do not need to be included as an option for the agent. Moreover, a different approach could be designed to include parallelism of actions by using their cross-product, however it is a trade off between action-space size and action efficiency.

Reward function

The agent is given a numeric reward of 1000 every time it manages to create the Battlecruiser unit and 0 otherwise. Once the Battlecruiser unit has been built, the episode ends and the agent returns at the starting state.

3.3.3 Plan-based reward shaping design

As discussed previously, there are numerous occasions when abstract knowledge regarding a task can be provided to an agent in terms of a STRIPS plan to help guide exploration. In the case of this scenario, the knowledge can be that of the build order that the agent has to learn. While it is impossible to be exact regarding the number of worker units that need to be built, in order to optimise resource income and speed up the process of building the Battlecruiser, expert players know the correct build order that needs to be followed. This knowledge of the build order can be used as a STRIPS plan to efficiently guide the agent to the optimal policy.

```
BUILD(Barracks)
BUILD(Factory)
BUILD(Science Facility)
BUILD(Physics Lab)
BUILD(Starport)
BUILD(Control Tower)
BUILD(Battlecruiser)
```

Figure 3.18: Starcraft STRIPS Plan.

```
1      have(Barracks)
2      have(Factory, Barracks)
3      have(Science Facility, Factory,
       Barracks)
4      have(Science Facility, Physics Lab,
       Factory, Barracks)
5      have(Starport,Science Facility,
       Physics Lab, Factory,
       Barracks)
6      have(Starport, Control Tower,
       Science Facility, Physics Lab,
       Factory, Barracks)
7      have(Battlecruiser, Starport,
       Control Tower, Science Facility,
       Physics Lab, Factory,
       Barracks)
```

Figure 3.19: Starcraft State-Based Plan.

Given this domain, the expected action-based STRIPS plan is given in Figure 3.18 and the full transformed state-based plan used for shaping is given in Figure 3.19 with the $CurrentStepInPlan$ used by Equation 2.16 noted in the left hand column.

3.3.4 Evaluation

This section presents the performance of an agent receiving erroneous plan-based knowledge in the SC:BW domain. However, as reward shaping methods have not been previously deployed in SC:BW, it is initially shown that those methods can be beneficial to agents, given that they are provided with a correct heuristic and then move on to show how the knowledge revision algorithms presented in Section 3.2 can help overcome the cases of erroneous knowledge.

All agents implemented SARSA with $\epsilon-$greedy action selection. For all experiments, the agents' parameters were set such that $\alpha = 0.1$, $\gamma = 0.99$, and $\epsilon = 0.3$ linearly decreasing during an experiment. Decreasing ϵ results in the agents being very explorative at the start of the experiment, and slowly shifting to a more greedy action selection near the end. Each experiment lasted for 500 episodes i.e. games, and was repeated a total of 30 times.

In all experiments, the scaling factor of Equation 2.16 was set to:

$$\omega = MaxReward/NumStepsInPlan \tag{3.3}$$

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.

Although the maximum discounted reward the agent can achieve cannot be directly computed, an estimate can be given to serve as a measure of performance. If we assume that no extra worker units are being built, then the total amount of minerals and gas the agent will need to collect are 1100 and 750 respectively as shown in Table 3.1. When the agent performs a resource collection action, it gathers on average $\simeq 30$ resources. This amounts in a total of, including the construction actions, 69 actions to reach the goal. As a result, given that $\gamma = 0.99$ and the goal reward $= 1000$, then the expected discounted goal reward is $\simeq 500$.

Correct Knowledge

Before evaluating the effects of erroneous knowledge and knowledge revision in the SC:BW scenario it needs to be established that plan-based reward shaping does benefit an agent acting in this complex setting. Therefore the performance of an agent using plan-based reward shaping is presented, compared to a baseline RL agent that receives no guidance. The agents are compared against the total discounted, by γ, reward they achieve. Note that the agents do not reach a reward of 1000, which is the reward for reaching the goal, as the reward they achieve is discounted by γ to account for the number of steps they performed, in order to reach the goal and finish the episode. The graphs include error bars which show the standard error.

Figures 3.20 and 3.21 show the performance of the agents comparing the discounted goal reward they achieve and the steps taken to complete an episode. The results are averaged over the 30 repetitions of the experiments. It is apparent that the agent using plan-based reward shaping manages to very early on in the experiment outperform the baseline agent without reward shaping, and learn the optimal policy in the number of episodes that the experiment lasts.

The agent without reward shaping receives no guidance regarding states that seem promising and as a result spends most of its actions exploring in random locations of the state-space, until a good enough estimate is calculated. It is worth noting that if the agents were left to run an infinite amount of time, they would eventually reach the same policy as the RL theory suggests. However learning must be achieved within a practical time limit. Considering this scenario, running more than 500 episodes per experiments, so as to have the baseline agent learn the optimal policy, would result in the experiments taking months to complete since as stated previously, an experiment of 500 episodes was completed within $14 \sim 16$ hours even though the game settings were being set at the highest speed. This is not at all practical especially if RL needs to be incorporated into commercial games when we consider the tight deadlines the game industry works with.

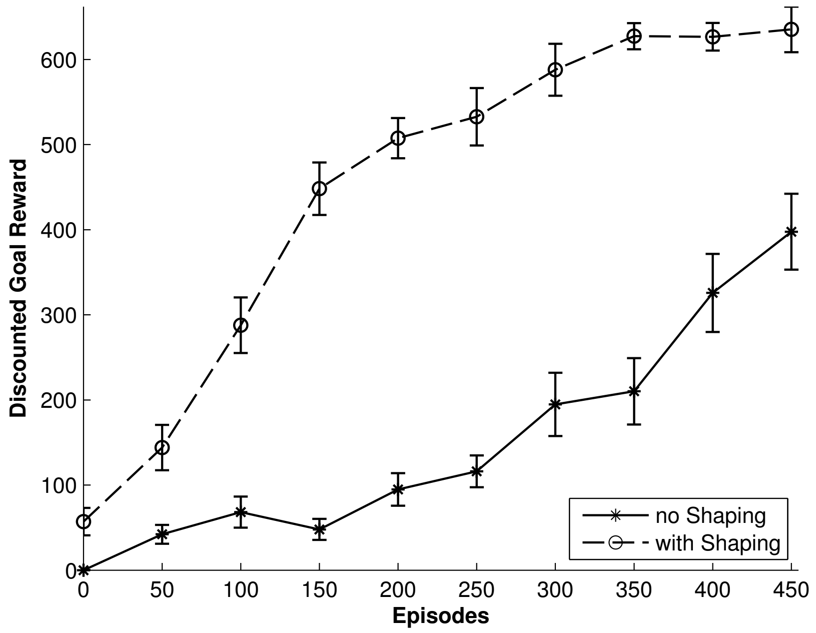


Figure 3.20: Discounted reward with correct knowledge in SC:BW

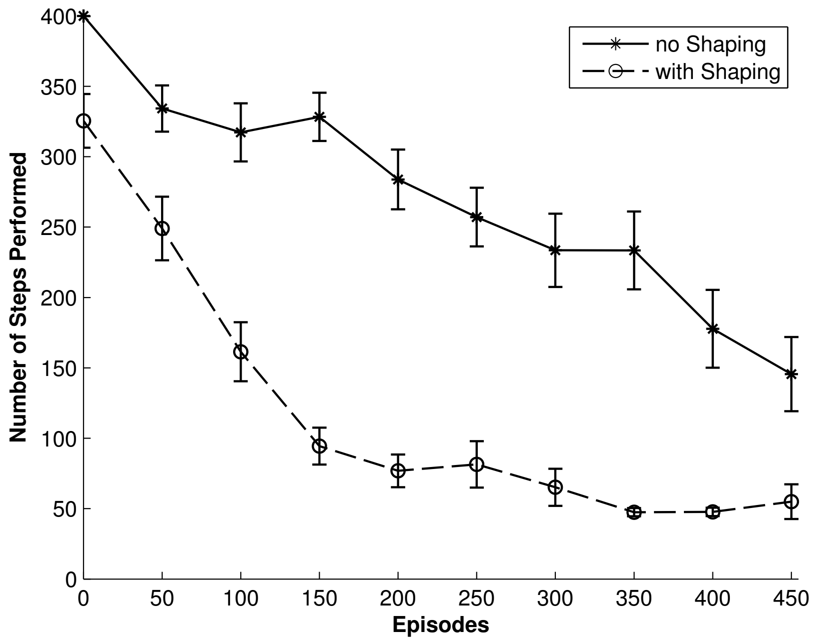


Figure 3.21: Average number of steps taken with correct knowledge in SC:BW

The agent using plan-based reward shaping is not at all affected by the complex, large environment. Since it receives guidance regarding promising states, it takes less time to calculate which states are “good” and which are not. Even though the experiments last for only 500 episodes, it still manages to find the optimal policy. The agent reaches a performance in the range of $550 \sim 600$ which is higher than the expected discounted reward specified earlier. The reason is that the expected discounted reward was calculated under the assumption that no extra worker units are being built to facilitate the resource collection process. This is of course not optimal but only an estimate as a higher flow of incoming resources, means that the agent needs to spend less actions for resource collection. However, the optimal number of workers needed to built the Battlecruiser, and by extension the maximum discounted goal reward the agent should reach, cannot be known in advance and it is up to the agent to learn and optimise the resource collection process.

To better understand why the agent using plan-based reward shaping performs significantly better than the agent without shaping we have included a graph showing the average number of steps each agent performed, in order to find the goal and complete an episode, shown in Figure 3.21.

It is clear that the agent without reward shaping spends a lot of its actions exploring. Many actions that are irrelevant or unnecessary are performed resulting in the agent exploring states that do not provide any benefit to the learning process and are not a part of the optimal policy.

The agent using plan-based reward shaping manages to reduce the number of unnecessary steps very early on in the experiment. Guided towards promising states by the provided abstract knowledge, it only performs $\approx$ 50 steps within 200 episodes while the agent without reward shaping spends $\approx$ 200 steps at the end of the experiments.

Having established that potential based reward shaping can be successfully applied in a SC:BW it is now possible to evaluate the effects or erroneous knowledge and the benefit of using the knowledge revision algorithms presented in Section 3.2.

Incorrect Knowledge

Similar to the flag collection domain, in this setting the agent is provided with an incorrect plan that includes additional information not present in the environment e.g. the agent is given a plan which states that it also needs to build Medic Units and Marine Units which do not feature in the environment. As a result the agent will not be able to satisfy those steps that contain the erroneous knowledge. A sample incorrect plan is given in Figure 3.22 and its corresponding state-based transformation in Figure 3.23.

Figures 3.24 and 3.25 show the performance of the agents when given incorrect knowledge using various levels of wrong knowledge i.e. the incorrect plan can contain from 2 up to 5 additional units or buildings that the agent should build. An agent being provided with the correct knowledge is also included to serve as an upper bound as well as an agent receiving no shaping.

```
BUILD( Barracks )
BUILD( Factory )
BUILD( Medic )
BUILD( Science Facility )
BUILD( Marine )
BUILD( Physics Lab )
BUILD( Starport )
BUILD( Control Tower )
BUILD( Battlecruiser )
```

Figure 3.22: Sample Starcraft Incorrect STRIPS Plan.

```
1       have( Barracks )
2       have( Factory , Barracks )
3       have( Medic , Factory , Barracks )
4       have( Science Facility , Medic , Factory ,
        Barracks )
5       have( Marine , Science Facility , Medic , Factory ,
        Barracks )
6       have( Science Facility , Marine , Physics Lab ,
        Medic , Factory , Barracks )
7       have( Starport ,Science Facility , Marine
        Physics Lab , Factory , Medic
        Barracks )
8       have( Starport , Control Tower , Marine
        Science Facility , Physics Lab , Medic
        Factory , Barracks )
9       have( Battlecruiser , Starport , Marine
        Control Tower , Science Facility , Medic
        Physics Lab , Factory ,
        Barracks )
```

Figure 3.23: Sample Starcraft Incorrect State-Based Plan.

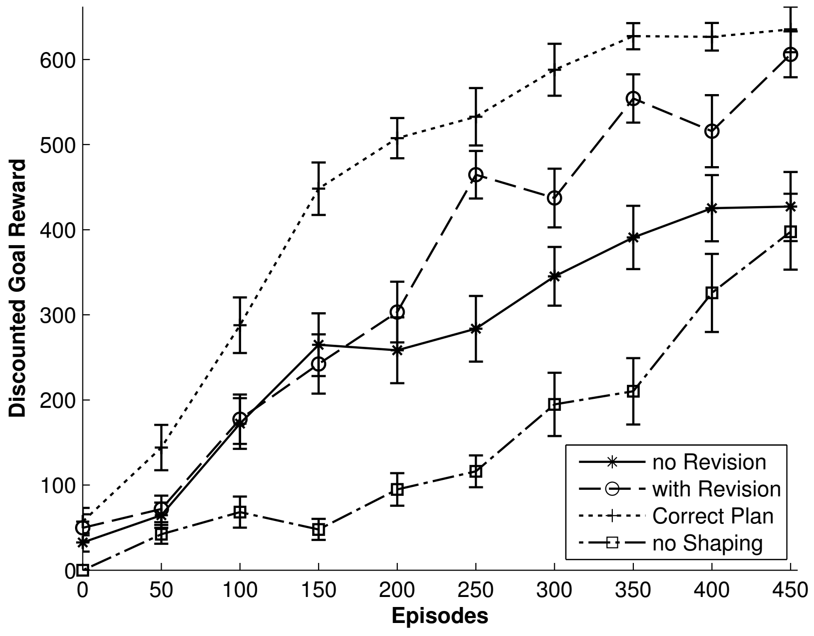


Figure 3.24: Discounted reward with incorrect knowledge in SC:BW

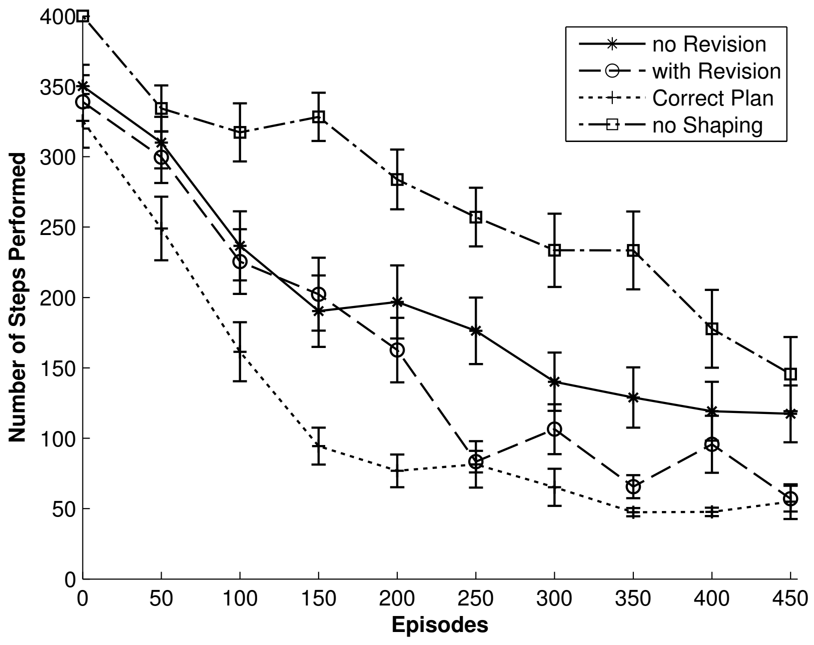


Figure 3.25: Average number of steps taken with incorrect knowledge in SC:BW

```
BUILD(Factory)
BUILD(Science Facility)
BUILD(Physics Lab)
BUILD(Control Tower)
BUILD(Battlecruiser)
```

Figure 3.26: Sample Starcraft Incomplete STRIPS Plan.

```
1        have(Factory)
2        have(Science Facility, Factory)
3        have(Physics Lab, Science Facility, Factory)
4        have(Control Tower, Science Facility, Physics Lab,
         Factory)
5        have(Battlecruiser, Control Tower, Science Facility,
         Physics Lab, Factory)
```

Figure 3.27: Sample Starcraft Incomplete State-Based Plan.

As shown in Figures 3.24 and 3.25 the same conclusions can be drawn as in the flag collection domain. The agent without knowledge revision is not able to overcome the problems of wrong knowledge and is being guided by an incorrect plan throughout the duration of the experiment. Since certain steps cannot be satisfied within the plan, the agent is essentially acting without any guidance.

In contrast, the agent utilising knowledge revision manages to rectify its knowledge and achieve a performance similar to the agent that is provided with the correct plan. Figure 3.25 shows the averaged number of steps the agents took to complete an episode. As in the flag-collection domain, the planner output was save in order to be able to have a clear understanding of the revisions that are taking place. In the case of incorrect knowledge, examining the planner output showed that the agent started revising its knowledge around the 50_{th} episode and had build up a correct knowledge base by the 350_{th} episode. The agent without knowledge revision spends most of its action in irrelevant parts of the state-space since it is guided by wrong knowledge while the agent with knowledge revision, after having revised its knowledge, prefers those parts of the state-space that lead to the goal quicker.

Incomplete Knowledge

In this setting the agent is provided with incomplete knowledge in the form of a plan that is missing important knowledge that would be beneficial e.g. the agent is given a plan that does not state that the Barracks and the Starport need to be built.

A sample incomplete plan and its state-based transformation is shown in Figures 3.26 and 3.27. Figures 3.28 and 3.29 show the performance of the agents in terms of discounted goal reward and average number of steps they took to complete an episode.

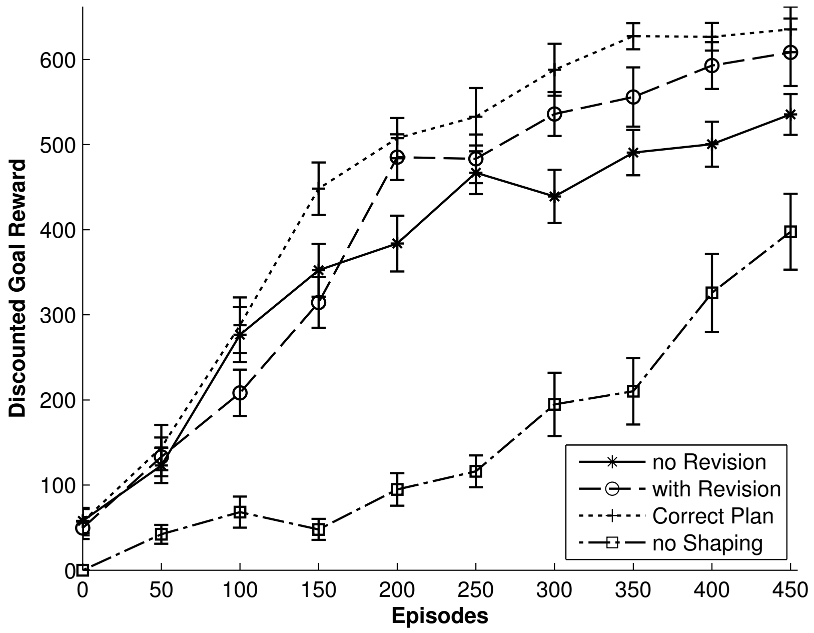


Figure 3.28: Discounted reward with incomplete knowledge in SC:BW

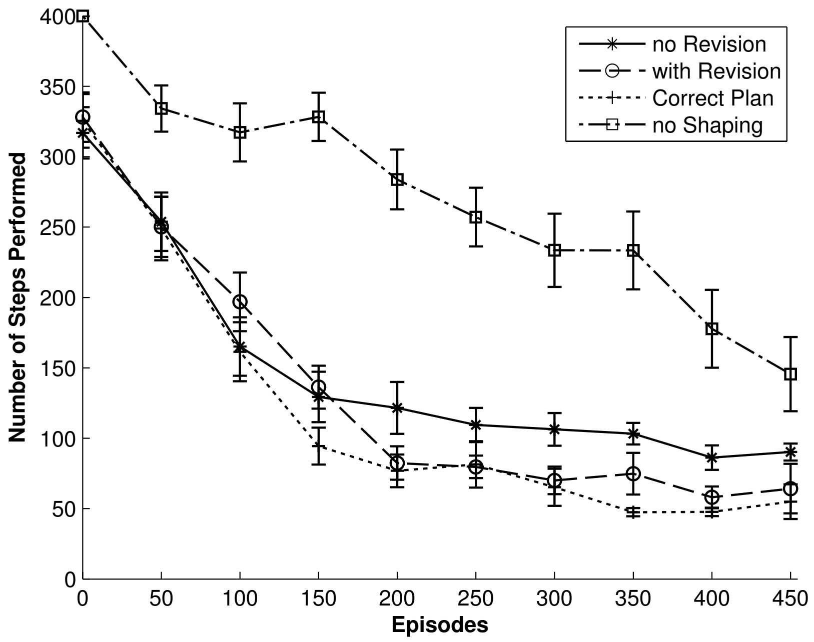


Figure 3.29: Average number of steps taken with incomplete knowledge in SC:BW

```
BUILD(Factory)
BUILD(Medic)
BUILD(Science Facility)
BUILD(Marine)
BUILD(Physics Lab)
BUILD(Control Tower)
BUILD(Battlecruiser)
```

Figure 3.30: Sample Starcraft Combination of Wrong Knowledge STRIPS Plan.

```
1       have(Factory)
2       have(Medic, Factory)
3       have(Science Facility, Medic, Factory)
4       have(Marine, Science Facility, Medic, Factory)
5       have(Science Facility, Marine, Physics Lab,
        Medic, Factory)
6       have(Control Tower, Marine
        Science Facility, Physics Lab, Medic
        Factory)
7       have(Battlecruiser, Marine
        Control Tower, Science Facility, Medic
        Physics Lab, Factory)
```

Figure 3.31: Sample Starcraft Combination of Wrong Knowledge State-Based Plan.

As in the case of incorrect knowledge, the agent using knowledge revision manages to identify those parts that are missing from the provided knowledge, revise them, and thus benefit from more accurate shaping as shown in Figure 3.28. Examination of the planner output during the re-planning periods of the agent showed that the agent started revising its knowledge to include new information at the 30_{th} episode and had build up a correct knowledge base by the 200_{th} episode. The agent without knowledge revision receives partial guidance and cannot achieve a good enough performance within the time frame of the experiments and spends most of its steps exploring randomly as shown in Figure 3.29.

Combination

The agents are now provided with both incorrect as well as incomplete knowledge. The agents not only have to deal with additional information that is not present in the environment but also with missing knowledge. The amount of errors in the knowledge is variable i.e. the plan can be missing up to 4 units that the agent will need to build (incomplete knolwedge) and it can contain up to 8 additional units or building to build (incorrect knowledge).

A sample plan that uses the examples given previously on incorrect and incomplete knowledge is given in Figure 3.30 and its state-based translation in Figure 3.31. In this example the

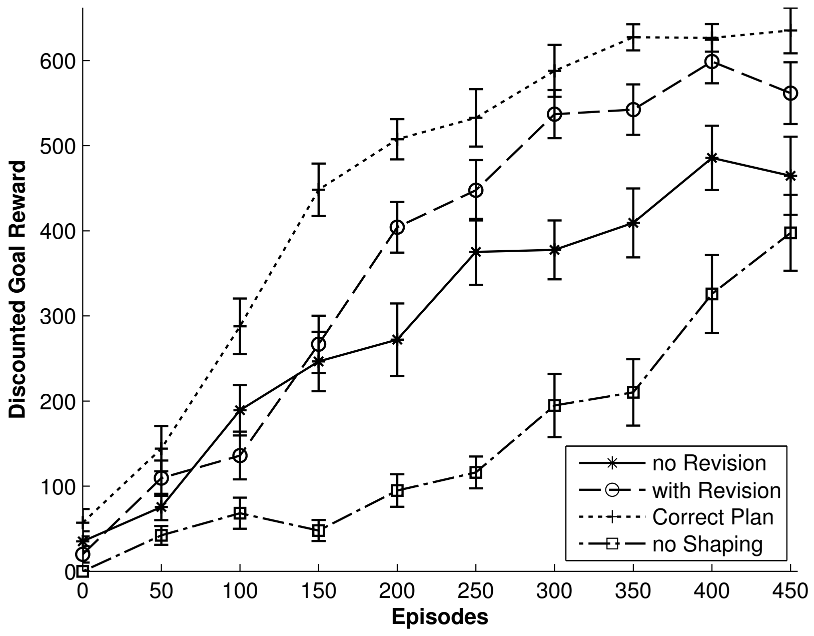


Figure 3.32: Discounted reward with incorrect and incomplete knowledge in SC:BW

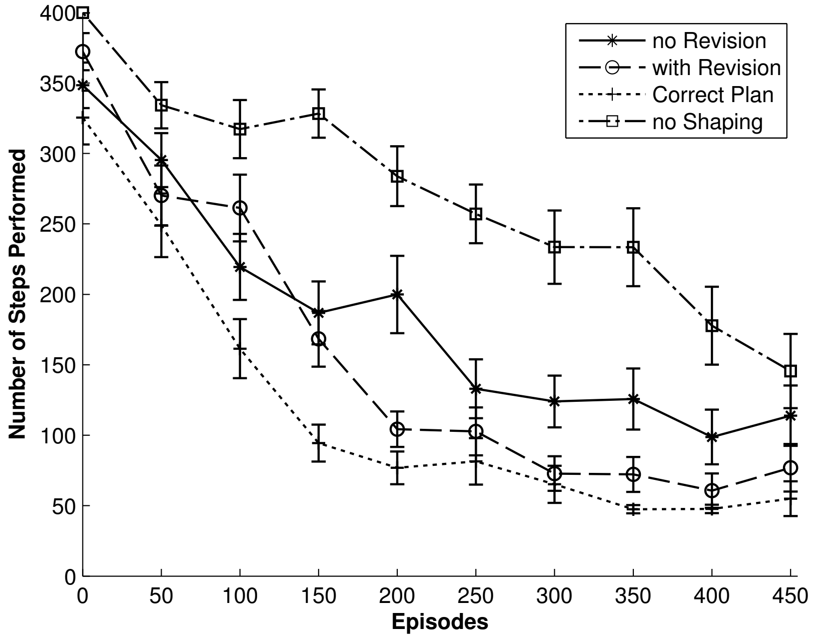


Figure 3.33: Average number of steps taken with incorrect and incomplete knowledge in SC:BW

agent is given a plan that does not state that the Barracks and the Starport need to be built and that the agent needs to build Medic Units and Marine Units which do not feature in the environment.

Figures 3.32 and 3.33 show the performance of the agents in terms of discounted goal reward and average number of steps taken to complete an episode.

Even in this very challenging setting of erroneous knowledge the agent utilising knowledge revision can achieve a performance similar to that of the agent provided with a correct plan. The revision algorithms provide the agent with the necessary tools to rectify its knowledge and benefit from more accurate plans.

Figure 3.33 shows that the agent without knowledge revision is misguided to irrelevant parts of the environment and is left exploring randomly while the agent with knowledge revision overcomes the problems caused by the erroneous knowledge and explore more beneficial parts of the state-space.

3.4 Conclusion

In conclusion, this chapter demonstrated how knowledge revision algorithms can be designed to be used in conjunction with plan-based reward shaping. Assuming that domain knowledge is always perfect is often not a realistic assumption and this chapter documented the adverse effects it can have when it is not dealt with efficiently.

It was demonstrated empirically that using knowledge revision can help an agent revise the erroneous parts of the provided domain knowledge and benefit from more accurate shaping in two different environments; a flag collection domain and Starcraft: Broodwar. In both environments, agents using knowledge revision algorithms managed to perform similarly to agent provided with the correct knowledge while the agents without knowledge revision were exploring random parts of the state-space which resulted in poor behaviour.

Despite the contributions this research is not without limitations. Firstly, it can be quite challenging to design an efficient knowledge verification algorithm. While DFS can be efficient in small domains it quickly becomes impractical when scaling to larger domains and perhaps it might be better to remove knowledge verification completely in those cases. Secondly, while the algorithms can capture erroneous knowledge in terms of missing or extra steps, they cannot deal with the cases where the plan might be correct, but the ordering of the steps is wrong. Lastly, as mentioned in the assumptions this research focused on deterministic environments. Applying this method in non-deterministic environments is not as straightforward and other approaches need to be designed.

The next chapter will focus on using knowledge revision methods with abstract MDP reward shaping to tackle non-determinism and also relax some of the assumptions presented in this chapter.

CHAPTER 4

Abstract MDP Reward Shaping with Knowledge Revision

The previous chapter on plan-based reward shaping with knowledge revision showed the adverse effects of erroneous domain knowledge and a new method utilising knowledge revision principles to help alleviate that problem. The results were very promising with the agent quickly revising the wrong parts of the knowledge base and thus benefiting from more accurate shaping. However, that method included exhaustively searching in the environment to verify the parts of the knowledge base that were deemed erroneous which can impact performance in larger domains. Moreover, in order for the agent to be able to search the environment it was allowed to 'teleport' between states. This required the agent to run only in simulation as backtracking or jumping among states cannot take place in a real world problem. In addition there is no mechanism in place in plan-based reward shaping with knowledge revision that handles non-deterministic environments and is therefore limited to only deterministic domains.

This chapter presents an alternative method to revise knowledge by the use of abstract MDPs coupled with a revision algorithm which relaxes some of the assumptions presented in the revision method for plan-based reward shaping. Marthi (2007) proposed the use of abstract MDPs as a source of reward shaping in which an abstract high-level MDP of the environment is defined and solved using dynamic programming, e.g. value iteration. The resulting value function can then be used in order to shape the agent.

The method of using abstract MDPs is evaluated in two environments. A comparison of the revision capabilities of abstract MDP reward shaping in the flag collection domain presented earlier is initially presented and it is demonstrated empirically that the agent can reach a similar performance to the agents using plan-based reward shaping with knowledge revision when the agents are provided with wrong knowledge. Lastly the abstract MDP agent is evaluated in a non-deterministic environment, a Micro Unmanned Air Vehicle (UAV) domain, and it is shown that the agent can overcome the problems posed by erroneous knowledge via revision despite the stochastic nature of the domain.

4.1 Abstract MDP Reward Shaping Revisited

As discussed in Section 2.2 reward shaping through abstract MDPs is an automatic method for imparting domain knowledge to a RL agent. The shaping algorithm obtains the potential function by firstly sampling the environment in order to learn dynamics for options (i.e. actions at the abstract level) and secondly solving an abstract MDP. Options can be defined as policies of low level actions (Sutton et al. 1999). Once the agent spends a number of episodes sampling the environment, it uses the resulting value function as a source for reward shaping.

We are interested in cases where domain knowledge comes from domain experts and therefore we have modified the abstract MDP reward shaping method to incorporate prior knowledge. We assume options to be primitive deterministic actions at an abstract level and therefore computation of their dynamics can be omitted.

In addition, by providing an abstraction of the low-level states of the environment to high-level abstract states the abstract MDP can be solved using dynamic programming before the main learning process begins and the obtained value function is used directly as the potential function:

$$\Phi(s) = V(z) * \omega, \tag{4.1}$$

where $V(z)$ is the value function over the abstract state space Z and it represents a solution to the corresponding MDP-based planning problem, and ω is an optional scaling factor. Modifying the method in this way, results in the agent not having to spend time randomly exploring the environment to estimate its dynamics and receives guidance the moment it starts acting.

The abstract MDP task can be solved using the following formula which is a special case of value iteration:

$$V_{k+1}(z) = \max_{z'} Pr_{zz'}[R_{zz'} + \gamma V_k(z')], \tag{4.2}$$

with $Pr_{zz'}$ being the probability of transitioning to the abstract state z' from the abstract state z, $R_{zz'}$ the reward received when transitioning to z' from z, γ the discount factor and $V_k(z)$ the value of state z at time k. Algorithm 4 shows the process of generating the abstract states to be used for solving the abstract MDP of any given environment.

The input to this algorithm is a state and an action abstraction which for brevity we will call SA and AA respectively. For all the the states and actions provided, each expected outcome is then computed to form a set of abstract states called Z. Using this set Z, for each abstract state z all of the reachable states are computed, and and MDP is solved using Formula 4.2.

Algorithm 4 Solving the Abstract MDP.

```
Input: state abstraction $SA$
  action abstraction $AA$
  for all $states$ in $SA$ and $actions$ in $AA$ do
    generate abstract states $Z$

  /* Solve MDP */
  Initialise $V(z)$
  while MDP not solved do
    get state $z$, set $max = 0$
    for all reachable states $z'$ from $z$ do
      $value = Pr_{zz'}[R_{zz'} + \gamma V(z')]$
      if $value > max$ then
        $max = value$
    $V(z) = max$
Output: solved MDP
```

The output of this algorithm is a solved MDP which contains all of the abstract states z and their value $V(z)$.

Instantiating to the flag collection domain to further illustrate this process, the states abstraction that can be used in this domain includes the rooms and halls that the agent can navigate to i.e. `hallA, hallB, roomA, roomB` and so on as well as the location of flags e.g. `flagA_in_roomA`. The actions abstraction can be the moves that the agent can perform from each position and specifically, which rooms or halls it can access from its current location e.g. `roomA_to_hallA` or `roomC_to_roomE`. The agent can also perform the action of picking a flag e.g. `taken_flagA`. By generating all the possible states and their respective probabilities an abstract MDP of the flag collection domain can be solved to be used as guidance by the agent. An example output of the solution used for shaping is shown in Figure 4.1 with with $V(z)$ used in Equation 4.1 shown in the right hand column.

robot_in(hallA)	96
robot_in(roomA)	98
robot_in(roomA) taken(flagA)	100

Figure 4.1: Example Partial Value Function

4.2 The Revision Process

This section presents the revision process for abstract MDP reward shaping. If an agent is provided with erroneous domain knowledge it is misguided through the course of an experiment and it can have a detrimental effect in its learning process. As mentioned previously using abstract MDPs with knowledge revision completely eliminates some of the assumptions present

Algorithm 5 Revision Algorithm.

```
Input: Abstract MDP
  Solve provided MDP to form $V(z)$ which will be used for shaping
  for $episode = 0$ to $max\_number\_of\_episodes$ do
    initialise transitions table $T$
    for $step = 0$ to $max\_number\_of\_steps$ do
      main learning process
      add $transition$ to $T$
      if $goal\ position$ then
        end episode
    /* add new transitions */
    for all $transition$ in $T$ do
      if $transition$ not in abstract MDP then
        add $transition$ to abstract MDP
        $Pr(transition) = 1$
    /* update the probabilities */
    for all $transition$ in abstract MDP do
      if $transition$ in $T$ then
        $Pr(transition) += \alpha[1 - Pr(transition)]$
      else
        $Pr(transition) += \alpha[0 - Pr(transition)]$
    Solve abstract MDP
    /* continue to next episode */
Output: updated Abstract MDP
```

in revision with plan-based reward shaping. Specifically, the only assumption that remains is the following:

- To implement abstract MDP reward shaping with knowledge revision we must assume an abstract high level knowledge and a direct translation of the low level states to the abstract high level MDP states i.e. the agent will always know, correctly, where it is located within the abstract MDP.

At each time step t the agent performs a low level action a and traverses to a different state s' this time shaped by a high level abstract MDP. Since the agent is performing low level actions it can gather information about the environment and as a result discover potential errors in the provided knowledge. Through the experiences of the agent in the low level environment the probabilities of the abstract MDP can be constantly updated in order to capture the dynamics of the environment and thus revise potential errors in order to benefit from more accurate shaping. It is worth noting that since now the agent is only dealing with transitions it does not care what form the erroneous knowledge can take; incorrect knowledge is just a transition that will be given a low probability and incomplete knowledge just a transition that is added in the current MDP. This method does not require the agent to verify knowledge as the revision takes place alongside the agent's exploration of the low level states whilst learning.

In order to identify erroneous knowledge, the agent constantly updates the abstract MDP probabilities according to its experiences in the low level environment using the following formula:

$$Pr_{zz'} = \begin{cases} Pr_{zz'} + \alpha(1 - Pr_{zz'}) & \text{if } z, z' \text{ experienced} \\ Pr_{zz'} + \alpha(0 - Pr_{zz'}) & \text{otherwise} \end{cases} \tag{4.3}$$

If the agent experiences a state transition which is not present in the abstract MDP, it is added with $Pr_{zz'} = 1$. The MDP is then solved and the new value function is used for shaping. This results in states which are not experienced, either because of wrong domain knowledge or because of the environment dynamics, to hold a low probability value. Consequently the states with low probability will have a very low potential and thus will not have an impact in the agent's action decision-making process. Algorithm 5 outlines the process of knowledge revision for agents using abstract MDP reward shaping.

As the potential function will change whilst the agent is learning, this is an instance of dynamic potential-based reward shaping (Devlin and Kudenko 2012a) and the theoretical guarantees of policy invariance hold.

In order to demonstrate further how abstract MDPs with knowledge revision tackle erroneous knowledge consider the following examples which are instantiated to the flag collection domain and deal with incorrect and incomplete knowledge.

Incorrect Knowledge Example

Consider an agent acting in the flag collection domain that is given incorrect knowledge. As mentioned earlier, incorrect knowledge is when agents are provided with domain knowledge that contains additional information not present in the environment e.g. an extra flag or room. In this example an agent is given an abstract MDP that contains an additional flag; `flagG_in_roomA`. A partial example of the value function used for shaping in this case is shown in Figure 4.2.

```
robot_in(hallA) 96
robot_in(roomA) 98
robot_in(roomA) taken(flagA) 100
robot_in(roomA) taken(flagA) taken(flagG) 111
robot_in(hallA) taken(flagA) taken(flagG) 123
robot_in(hallA) taken(flagA) 105
robot_in(roomD) taken(flagA) taken(flagG) 140
robot_in(roomD) taken(flagA) 129
```

Figure 4.2: Example Partial Incorrect Value Function

While the agent is acting in the environment the probabilities of the abstract MDP can be updated according to its low-level experiences. As mentioned earlier, each state within the provided abstraction is associated with a probability which can be updated to capture the dynamics of the environment using the formula shown in Equation 4.3.

At the start of the experiment all the state probabilities have the same value and are set to 1. Setting the probabilities to 1 means that the agent fully trusts the domain knowledge that has been provided and expects to be able to successfully transition to all the states in its knowledge. In this example, all those states that contain `flagG` however, will never be experienced and as a result their probability will be constantly decreasing while the rest of the probabilities will remain unchanged, provided they are always experienced. If Figure 4.2 is considered a snapshot of the value function used for shaping at the first episode, then by constantly decreasing the probabilities after 1000 episodes the value function will be similar to that shown in Figure 4.3.

This example shows how abstract MDPs are able to decrease the probability of those states that correspond to erroneous knowledge and as a result lower their value which in turn results in the agent being guided by more accurate shaping.

```
robot_in(hallA) 96
robot_in(roomA) 98
robot_in(roomA) taken(flagA) 100
robot_in(roomA) taken(flagA) taken(flagG) 20
robot_in(hallA) taken(flagA) taken(flagG) 31
robot_in(hallA) taken(flagA) 105
robot_in(roomD) taken(flagA) taken(flagG) 34
robot_in(roomD) taken(flagA) 129
```

Figure 4.3: Example Partial Incorrect Value Function after 1000 Episodes

Incomplete Knowledge Example

In this example, consider an agent acting in the flag collection domain and receives incomplete knowledge. Incomplete knowledge is when an agent is provided with knowledge that is missing important information that the agent would find beneficial e.g. missing flags or rooms. In this example the agent provided with an abstract MDP that does not contain any information regarding `flagA`. A partial example of the value function used for shaping is shown in Figure 4.4.

```
robot_in(hallA) 96
robot_in(roomA) 85
robot_in(roomD) taken(flagD) 104
```

Figure 4.4: Example Partial Incomplete Value Function

While the agent is acting in the low-level environment it can identify states which are not present in its provided knowledge. These new states can be added to the abstract MDP as new transitions. If Figure 4.4 is a snapshot of the value function used for shaping at the start of the experiment, then by adding new transitions the value function will look similar to Figure 4.5.

```
robot_in(hallA) 96
robot_in(roomA) 104
robot_in(roomA) taken(flagA) 110
robot_in(hallA) taken(flagA) 119
robot_in(roomD) taken(flagA) 125
robot_in(roomD) taken(flagA) taken(flagD) 133
```

Figure 4.5: Example Partial Incomplete Value Function after Revision

This example describes how agents using abstract MDP reward shaping with knowledge revision, can tackle incomplete knowledge by adding new transitions to the abstract MDP and thus benefit from better shaping.

4.3 Parameter Evaluation

During the initial experimentation with abstract MDP reward shaping it was apparent that this method requires parameter tuning. The original paper on this method (Marthi 2007) did not mention any parameters used to solve the abstract MDP and therefore some of the findings are presented in this section.

The parameter in question is the γ parameter which is used to offset future versus immediate rewards. Initially setting this parameter at random showed that it can make or break an agent's shaping. Setting the reward function also showed to have an effect in the agent's learning process but that effect was minimal. The α parameter did not have a big impact in the agent behaviour and is not worth reporting.

Figure 4.6 shows the parameter evaluation in the flag collection domain. The comparison shows the results of agents using a γ value of 0.99, 0.9, 0.8 and 0.1 to cover the extremes. The graph shows that the agent using a γ value of 0.9 significantly outperforms all other setting. These findings show that there is a clear correlation between the reward of the environment and the potential difference of abstract states. Since abstract MDP shaping can have a value associated with every low level state the agent is in, contrary to plan-based reward shaping where only a single path to the goal is provided, if the potential difference across high level states is minimal then as the experiments show, the effect of reward shaping is also minimal to the point of receiving no meaningful shaping at all as it happens with the agent with γ set to 0.1. On the other hand if the potential differences are very large then the agent struggles to separate the environment reward from the additional rewards at the start of the experiment as is the case with the agent with γ set to 0.99.

These findings are very important in RL and reward shaping as a whole. This thesis is however concerned with knowledge revision in RL with reward shaping and therefore further examination is not within the scope of this topic. For now, these findings are used to correctly set the parameters of abstract MDP reward shaping and leave further examination as future work.

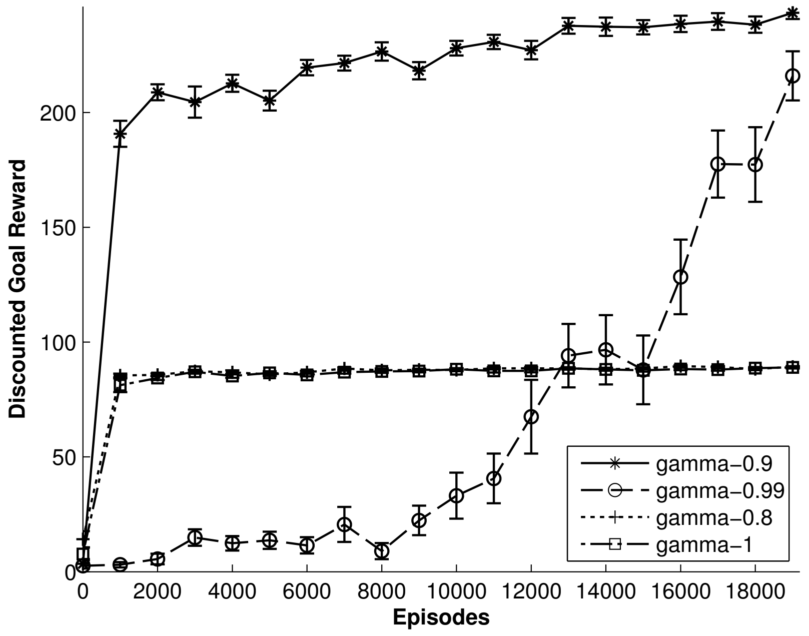


Figure 4.6: γ comparison in the flag collection domain.

4.4 Evaluation in the Flag Collection Domain

This section evaluates abstract MDP reward shaping with knowledge revision in the flag collection domain. This is the same domain used in plan-based reward shaping and the purpose of this examination is to show that abstract MDP shaping can reach a similar performance with plan-based reward shaping when both methods are provided with erroneous knowledge. The erroneous knowledge can take the form of incorrect and incomplete knowledge. In addition this section presents the results of the abstract MDP agent when it is provided with an extreme version of erroneous knowledge(incorrect, incomplete, wrong connections, misplaced flags etc.) which is something the plan-based agent cannot deal with.

As discussed previously the flag collection domain is an extended version of the navigation problem. The agent needs to navigate through the maze, collect flags and drop them off at a designated location. At each time step it can decide to move up, down, left or right and will deterministically complete its move provided it does not collide with a wall. Regardless of the collected flags the episode ends once the agent reaches the goal position and the given reward is relative to the number of collected flags.

The abstract MDP agent is compared against an agent with knowledge revision using plan-based reward shaping. The comparison is based on the performance in terms of discounted goal reward.

All agents implemented SARSA with $\epsilon-$greedy action selection and eligibility traces (Sutton and Barto 1998). For all experiments, the agents' parameters were set such that $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon = 0.3$ and $\lambda = 0.4$. The experiments were run for 30 iterations each lasting $50,000$ episodes.

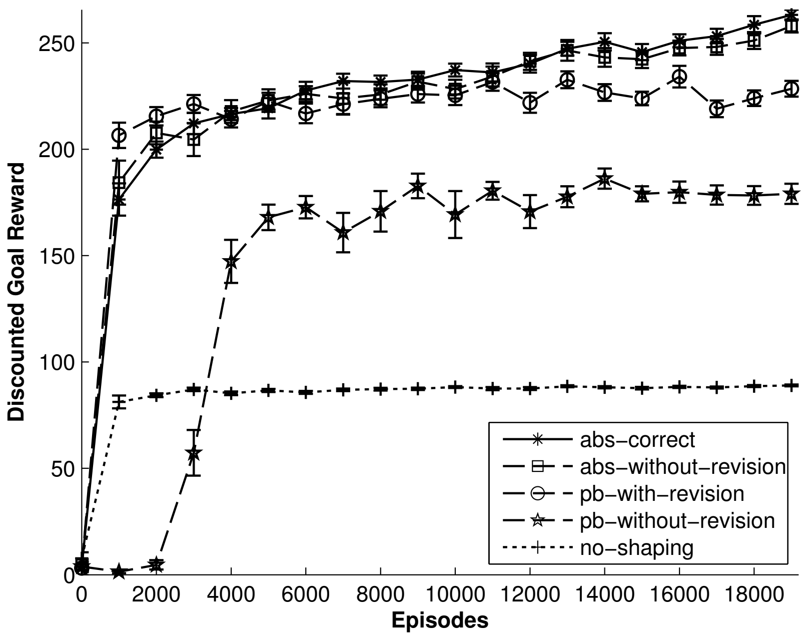


Figure 4.7: Incorrect knowledge comparison.

In the case of incorrect knowledge, the agents are provided with knowledge which contains extra information which is not present in the environment. This means that the knowledge contains those facts that are correct in the environment as well as more irrelevant information e.g. the knowledge contains two extra flags which are not present in the simulation but also includes those that are in the correct position. The instances of incorrect knowledge are varying per experiment i.e. different additional flags are placed at different locations. As an example, experiment 1 might contain the additional flag `flagG_in_roomA` while experiment 2 might contain the additional flag `flagH_in_roomE`.

In the case of incomplete knowledge, the agents are provided with knowledge which is missing important goals e.g. three flags are missing from the provided knowledge and thus the shaping does not provide an incentive to collect them. As in the incorrect case, the instances of incomplete knowledge vary per experiment .i.e. different knowledge is absent from the abstract MDP on each experiment.

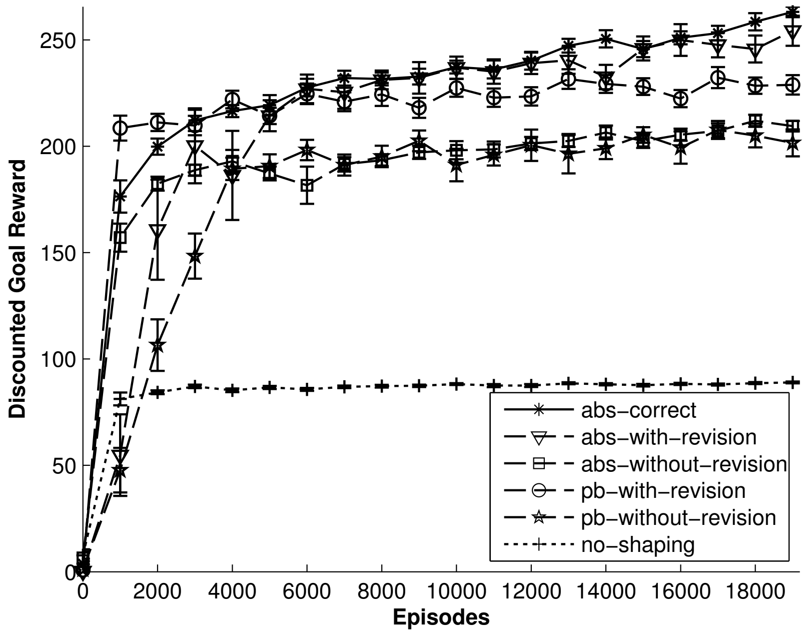


Figure 4.8: Incomplete knowledge comparison.

In the extreme case, the provided knowledge not only contains incorrect and incomplete knowledge in terms of flags as described previously, but also wrong connections between rooms and misplaced flags. Therefore there exist cases where the provided knowledge can be completely wrong. These results are only presented for the abstract MDP shaping with revision as the plan-based method does not handle these types of wrong knowledge efficiently and its performance does not serve for comparison. The results for the incorrect case are shown in Figure 4.7, the incomplete in Figure 4.8 and the extreme case in Figure 4.9.

The results are very interesting when it comes to the incorrect case of wrong knowledge in abstract MDPs. While the plan-based agent has an impact in behaviour and a need for revision is apparent, this is not the case for the abstract MDP agent. It appears that the agent's performance is not impacted at all. Due to this behaviour, the abstract MDP agent with knowledge revision is not included in Figure 4.7 since it achieves a similar performance to the agent without revision. Taking a closer look at how the abstract MDP provides extra rewards, it becomes clear why this happens. Since a value function is used as a reward shaping source, every state the agent finds itself in will have a potential that will lead to the goal. These multiple paths to the goal mean that the agent will never be left without guidance. Paths which are not encountered by the agent because they do not exist do not feature at all when receiving rewards. Therefore there is no need to revise incorrect knowledge when using abstract MDP shaping.

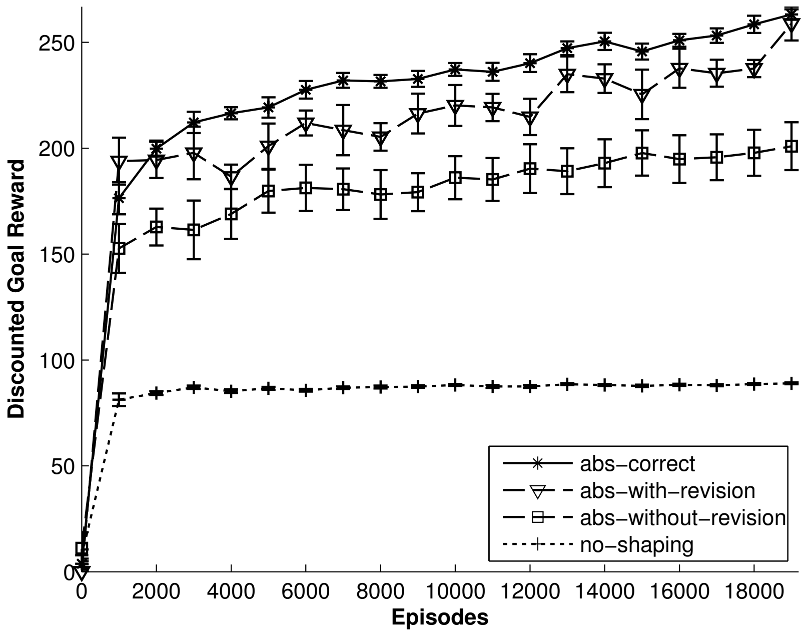


Figure 4.9: Abstract MDP agent with extreme version of wrong knowledge.

```
0  robot_in(hallA)
1  robot_in(roomA)
2  robot_in(roomA)  taken(flagA)
3  robot_in(roomA)  taken(flagA)  taken(flagG)
4  robot_in(hallA)  taken(flagA)  taken(flagG)
5  robot_in(roomD)  taken(flagA)  taken(flagG)
6  robot_in(roomD)  taken(flagA)  taken(flagG)  taken(flagD)
```

Figure 4.10: State-Based Plan in the Flag Collection Domain

The plan-based agent on the other hand receives only a single path to the goal. Therefore if the agent cannot achieve a step in the provided plan because the path does not exist, it does not receive any further guidance after that point and is effectively left to act without reward shaping.

It is better to illustrate the case of incorrect knowledge with an example comparing the two methods, abstract MDPs and plan-based reward shaping, in the flag collection domain. Consider that the agents are provided with knowledge that includes additional information regarding `flagG_in_roomA` and that the domain only contains flags `flagA` and `flagD`. Then a full plan is shown in Figure 4.10 and a partial value function in Figure 4.11.

The agent using plan-based reward shaping receives a single path to the goal. This effectively means that since the agent cannot ever be in the state of having taken `flagG`, since it is not in

```
robot_in(hallA) 30
robot_in(hallA) taken(flagA) 43
robot_in(hallA) taken(flagA) taken(flagG) 52
robot_in(hallA) taken(flagA) taken(flagG) taken(flagD) 60
robot_in(roomA) 34
robot_in(roomA) taken(flagA) 38
robot_in(roomA) taken(flagA) taken(flagG) 48
robot_in(roomA) taken(flagA) taken(flagG) taken(flagD) 44
robot_in(roomD) 32
robot_in(roomD) taken(flagA) 46
robot_in(roomD) taken(flagA) taken(flagG) 55
robot_in(roomD) taken(flagA) taken(flagD) 55
robot_in(roomD) taken(flagA) taken(flagG) taken(flagD) 60
```

Figure 4.11: Partial Value Function in the Flag Collection Domain

the environment, the agent will never receive any additional rewards after step 2 in the plan. As a result it is left to find the optimal solution without receiving reward shaping after that point.

On the other hand, the agent using abstract MDP reward shaping receives multiple paths to the goal. Despite not being able to receive the additional reward of picking up `flagG` the agent is still encouraged to continue and gather the rest of the flags. Specifically, after the agent picks up `flagA`, it receives additional reward for going back to `hallA` and then to `roomD` to pick up `flagD`. Since the agent never experiences any of the states involving `flagG` it receives the additional rewards for picking up the rest of the flags and as a result is not impacted at all by the incorrect knowledge in the abstract MDP.

In the incomplete case, the agent can encounter states which are not in its provided knowledge. The agent manages to quickly identify the parts which are missing from the provided MDP. By adding these new transitions it encounters in the low-level to the abstract MDP, it manages to solve a more accurate MDP and thus benefit from better shaping and reach a similar performance to the agent using plan-based reward shaping.

In the extreme case, the provided knowledge not only contains incorrect and incomplete knowledge in terms of flags as described previously, but also wrong connections between rooms and misplaced flags. Therefore there exist cases where the provided knowledge is completely wrong. These results are only presented for the abstract MDP shaping with revision as the plan-based method does not handle these types of wrong knowledge.

As mentioned previously, imparting knowledge to an agent can still prove beneficial compared to no shaping, even in the cases where the domain knowledge is partially wrong. This section showed that the abstract MDP agent with knowledge revision can achieve the same performance as the agent using plan-based reward shaping with knowledge revision when both agent are provided with erroneous knowledge in a deterministic environment. The next section evaluates the abstract MDP agent in a non-deterministic environment.

4.5 Evaluation in the Micro Unmanned Air Vehicle Problem

While both these methods work perfectly well in a deterministic domain and both reach a similar performance, in order for them to be widely applicable they need to be able to tackle the complexity of non-determinism. This section evaluates the abstract MDP reward shaping method with knowledge revision in a non-deterministic domain, the Micro UAV problem which was developed by our industrial collaborators at QinetiQ and represents a real world scenario they are interested in. Plan-based reward shaping with knowledge revision cannot tackle such a domain as there is no mechanism in place to account for non-deterministic environments and is not included it in this examination.

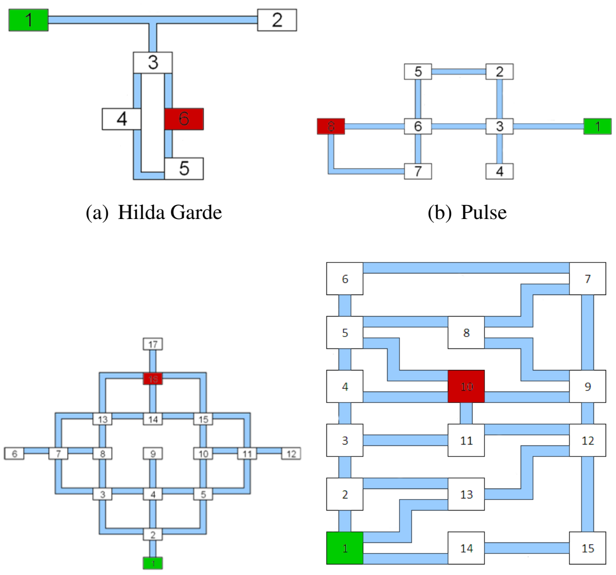

(a) Hilda Garde (b) Pulse (c) Tomb of the Forgotten King (d) Pharos Ascend

Figure 4.12: Micro UAV Problem.

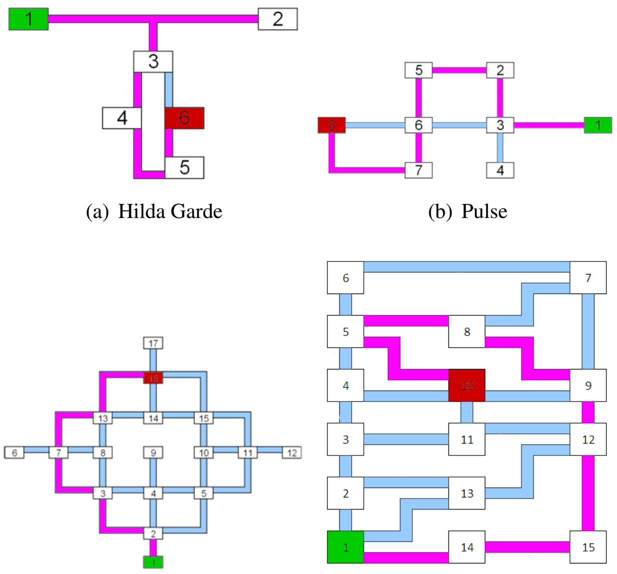
The Micro UAV problem is one where a micro UAV is intended to go through a building and locate a villain. This is realised as a grid world where the agent is provided with a high level map of the building it needs to search and serves as the domain knowledge that the agent is provided with. The building is comprised of rooms and corridors each associated with a probability that the agent gets caught by the enemy. The domain knowledge that is provided is similar to that in the flag collection domain with rooms and halls but this time the knowledge also includes the probabilities that the agent might get caught at certain parts in the maze. Specifically, the state abstraction includes rooms `1-17` and the probability of the agent being detected in those areas, halls `A-V` along with the probability of detecting the agent and the location of the villain within the building. A partial value function for this domain is shown in Figure 4.13 where the

```
robot_in(hallA)  30
robot_in(room1)  35
robot_in(hallC)  39
robot_in(room3)  10
robot_in(hallE)  8
robot_in(room5)  40
robot_in(hallB)  34
robot_in(hallD)  4
```

Figure 4.13: Partial Value Function in the Micro UAV Domain

low values of certain states correspond to areas of high detection in the building and should be avoided.

Within the grid the agent can choose to move up, down, left or right to one of its neighbouring squares. If the agent is caught while searching the building then the episode ends and the agent receives a small negative reward. If the agent manages to successfully locate the villain, it is given a high reward and the episode is reset.



(a) Hilda Garde (b) Pulse

(c) Tomb of the Forgotten King (d) Pharos Ascend

Figure 4.14: Micro UAV Problem. Safe routes.

It is worth noting that a clear path to the villain always exists and remains constant throughout the duration of an experiment. The rooms that are not part of this safe route have a $0.4 - 0.6$ probability of detecting the agent while the corridors a $0.7 - 0.9$ probability. This ensures that an enemy can always be found, depending on the probabilities the agent might have to follow a much longer route if the shortest path has a high probability of getting caught, therefore the agent will not get stuck trying to search for an enemy endlessly. Figure 4.12 shows the map configuration used for evaluating this method in the Micro UAV problem. The room the agent is located is shown in green and the villain's position within the building is shown in red. Figure 4.14 shows the safe route for each of the building noted in magenta.

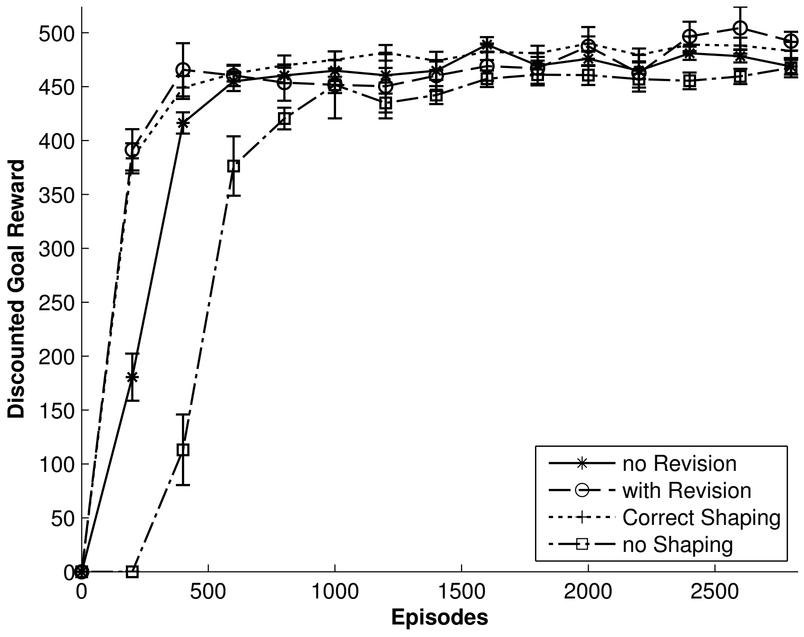


Figure 4.15: Hilda Garde.

The same parameters are used as those in the flag collection domain to conduct these experiments. For clarity the graphs only show up to 3000 episodes since there was no change in performance after that point. The graphs also include error bars showing the standard error from the mean.

The erroneous knowledge the agents are given is both incorrect and incomplete with varying degrees of 'errorness' per experiment i.e. the agent receives a combination of erroneous knowledge which can vary for each experiment in the number of incorrect or incomplete parts. For example, in the Hilda Garde building shown in Figure 4.12 the agent can receive domain knowledge which can be missing `room3`, have additional information of a corridor, `hallT`, connecting `room5` to `6` and also have the probability that the agent gets detected in the corridors set to 0 for all corridors.

A sample value function for this example is shown in Figure 4.19. The agent will need to utilise its knowledge revision capabilities and overcome the problems posed by this erroneous knowledge. Figures 4.15, 4.16, 4.17 and 4.18 show the agents' performance.

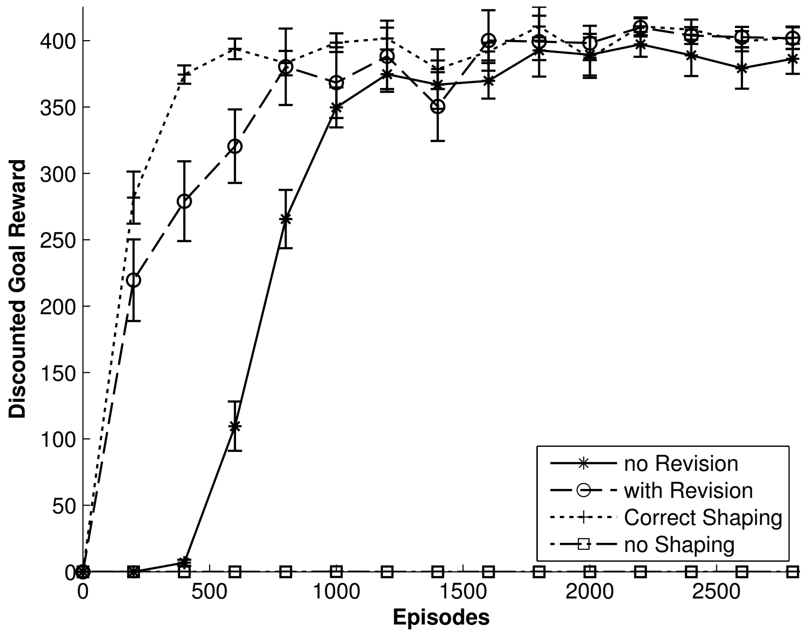


Figure 4.16: Pulse.

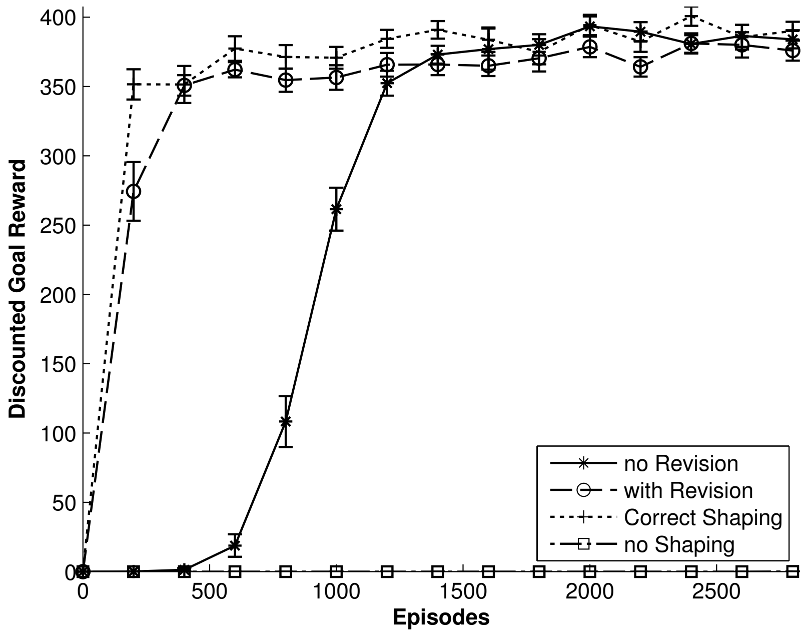


Figure 4.17: Tomb of the Forgotten King.

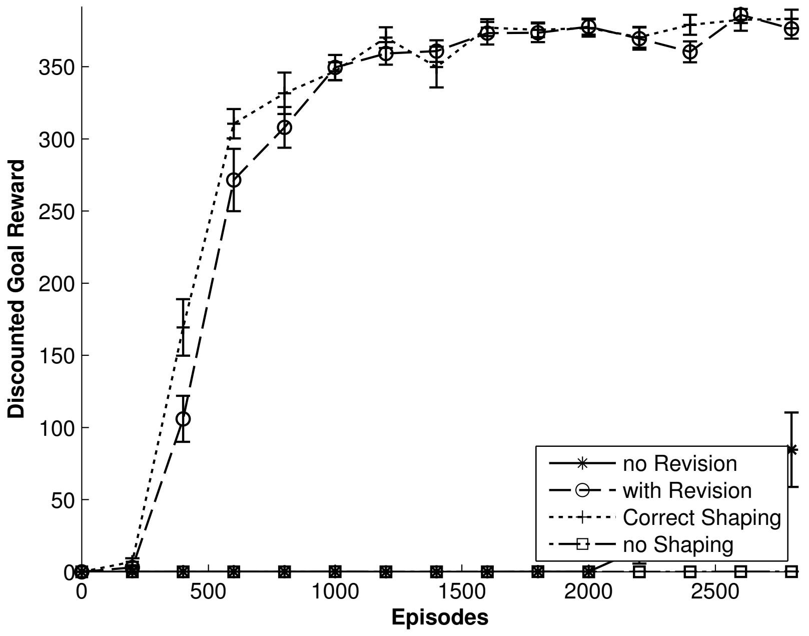


Figure 4.18: Pharos Ascend.

robot_in(room1)	0
robot_in(hallA)	0
robot_in(room2)	0
robot_in(hallB)	46
robot_in(room4)	55
robot_in(hallC)	68
robot_in(room5)	75
robot_in(hallD)	85
robot_in(hallT)	85
robot_in(room6)	100
robot_in(hallE)	81

Figure 4.19: Sample Erroneous Value Function in the Hilda Garde Domain

Similar to the flag collection domain the abstract MDP agent can efficiently revise erroneous knowledge even when dealing with a non-deterministic environment. On all configurations the agent manages to quickly overcome the problems posed by erroneous knowledge and reach a performance similar to the agent receiving correct knowledge. The agent without knowledge revision takes longer to find the optimal policy, around 1000 episodes on most cases apart from Pharos Ascend where it takes more than 4000 episodes to start improving, while the agent receiving no shaping is not able to overcome the problem of non-determinism and cannot reach a good enough policy within the time frame shown in the graphs. As the theory suggests all agents are able to reach the same performance after episode 15000 but the benefits of using reward shaping and knowledge revision are visible especially at the start of the experiments.

While it is very difficult to report the revisions that are taking place when using abstract MDP reward shaping, mainly because there is constant updating of beliefs until the end of an experiment, it was still possible to extract the final MDP after each experiment and compare it against the correct MDP for each scenario. In both the flag-collection domain and the Micro UAV problem the agent managed to revise the values in the MDP and end up with similar values as those in the correct MDP. However, there were cases where certain values differed but not so much so as to cause the performance to drop. It is also worth noting that in contrast to plan-based reward shaping, where it is not possible for the agent to revise correct information, in the abstract MDP case the agent revises everything according to its experiences which are based on exploration. This means that the agent could potentially revise out beliefs that are correct and could prove beneficial to its performance. Nevertheless, this was not witnessed in any of the experiments that were performed in this thesis and the agent still managed to revise the values of the MDP close to correct values.

4.6 Non-Grid World Domains

This chapter demonstrated the use of knowledge revision when using abstract MDP reward shaping. This was demonstrated in the flag-collection domain and a Micro UAV problem. Both these domains however are grid world domains and while they do pose quite a few challenges it would be useful to evaluate this algorithm to a domain which is not a grid world. This is left as future work but this section will present how we could move to different environments.

It is our belief that knowledge revision with abstract MDP reward shaping can be successfully applied to a wide variety of domains. In particular, any domain that can be expressed as an MDP should be a suitable application for this method as long as the algorithm's assumption is maintained i.e. to implement abstract MDP reward shaping with knowledge revision we must assume an abstract high level knowledge and a direct translation of the low level states to the abstract high level MDP states.

The steps to implement abstract MDP reward shaping in non-grid world domains follow the same order that is presented in this thesis. Once a suitable state and action abstraction has been devised, they can be used to solve an abstract MDP with the value function used to shape the agent's decisions.

As an example consider the Starcraft scenario that is presented in Section 3.3 which is a qualitatively different domain compared to the flag-collection and Micro UAV domains. The same state and action abstraction can be used in order to form a MDP which later is solved and its resulting value function used for shaping. Given the Starcraft domain, a sample shaping function that we would expect to see is shown in Figure 4.20 with the value of each state shown in the right.

```
have(Barracks) 30
have(Factory, Barracks) 44
have(Starport ,Science Facility) 23
have(Science Facility, Physics Lab) 15
```

Figure 4.20: Sample Starcraft Abstract MDP

Following these steps should enable the application of abstract MDP reward shaping to other classes of domains which are not grid worlds. Hopefully the Starcraft example provided in this section has helped the reader understand and implement abstract MDP reward shaping to any domain that can be expressed as an MDP while maintaining the assumptions set in this chapter.

4.7 Conclusion

In conclusion, it was demonstrated that knowledge revision algorithms can be designed to be used in conjunction with abstract MDP reward shaping. Assuming that domain knowledge is always perfect is often unrealistic and this chapter documented the adverse effects it can have when it is not dealt with efficiently even in non-deterministic environments.

It was demonstrated empirically that using knowledge revision can help an agent revise the erroneous parts of the provided domain knowledge and benefit from more accurate shaping in two different environments; the flag collection domain and a Micro UAV problem. It was shown that the agent using abstract MDP reward shaping reached the same performance as the agent using plan-based reward shaping with knowledge revision when both agents are provided with erroneous knowledge.

In addition it was demonstrated that the abstract MDP agent can efficiently deal with non-deterministic environment and is able to overcome the problem of erroneous domain knowledge via revision and thus benefit from more accurate shaping.

Moreover many of the assumptions that make plan-based reward shaping with knowledge revision feasible have been eliminated in knowledge revision with abstract MDPs.

Despite the contributions this research is not without limitations. The main disadvantage of using abstract MDPs is the parameter configuration. There is no research yet that dictates how abstract MDPs should be set and is thus left to find the correct settings through experimentation. This is something which is not present in plan-based reward shaping as it is much easier to design a reward function using that method; all it takes is a start state, actions and effects, and a goal state.

In addition, the size of the MDP can grow very large in size as the environments get larger. Some of the MDPs contained more than 100000 states and took more than 5 minutes to be solved, with the upper bound being 17 minutes in the experiments reported in this thesis, and this can potentially make it difficult to use when scaling to more complex domains. Automatically generating all the possible states can reach a complexity of n^3 or more depending on the implementation and the environment but this limitation is perhaps more interesting from an algorithmic complexity point of view.

Despite these drawbacks abstract MDPs provide the means to use reward shaping with knowledge revision efficiently in stochastic environments.

CHAPTER 5

Abstract MDP Reward Shaping in Multi-Agent RL

Chapters 3 and 4 presented how reward shaping and knowledge revision can be used in a single agent environment when provided with erroneous knowledge. This chapter presents some initial results in a multi-agent scenario and how reward shaping can be used for conflict resolution.

Previous research demonstrated the use of plan-based reward shaping for multi-agent reinforcement learning (Devlin and Kudenko 2012b). This method uses STRIPS planning to generate a potential function in order to shape the agents. The generation of multi-agent plans can occur within one centralised agent or spread amongst a number of agents (Rosenschein 1982; Ziparo 2005). The centralised approach benefits from full observation making it able to, where possible, satisfy all agents' goals without conflict. However, this approach requires sharing of information, such as goals and abilities, that agents in a multi-agent system often will not want to share.

The alternative approach, allowing each agent to make their own plans, will tend to generate conflicting plans. This can have an impact in behaviour with the agents not being able to co-ordinate efficiently (Devlin and Kudenko 2012b) and fail to reach a good enough policy similar to that of receiving joint plans. Many methods of co-ordination have been attempted including, amongst others, social laws (Shoham and Tennenholtz 1995), negotiation (Ziparo 2005) and contingency planning (Peot and Smith 1992) but still this remains an ongoing area of active research.

I am interested in those settings where information sharing is not allowed and the agents are agnostic to other learning entities acting in the environment.

This chapter proposes the use of a modified version of abstract MDPs for reward shaping in Multi-Agent RL (MARL). It demonstrates empirically that the agents using abstract MDP reward shaping despite receiving decentralised shaping which contains conflicting goals, manage to efficiently overcome them and coordinate to learn a much better policy compared to the agents using plan-based reward shaping, which fail to do so. It is shown that abstract MDPs as a reward shaping function can be used in decentralised planning for decentralised agents as a means of coordination.

5.1 Multi-Agent RL

Multi-Agent RL, as the name suggests, is the deployment of multiple RL agents in the same environment. As with single agent RL the agents can learn to improve their performance through repeated interactions with the environment (Sutton and Barto 1998). In addition multi-agent RL agents can potentially share experiences amongst them which can benefit weaker agents that can mimic an expert (Tan 1993).

One of the problems in single-agent RL is also one of the more prominent problems in multi-agent RL; the state-space explosion. With the addition of multiple agents this problem becomes worse in multi-agent RL as each agent adds its own variables to the joint state-action space. Each time an agent is added there is an exponential increase in the observable features.

One implication of MA learning is that the environment is no longer static as is mostly the case in single-agent RL. In addition, as the transition probability function is now dependent on the joint action, if an agent can observe only its own action then the Markov property does not hold.

Given these difficulties it would be better for agents to co-ordinate, but achieve co-ordination while following their independent goals can be challenging.

In RL multi-agent systems are often modelled as a generalisation of MDPs, stochastic games(SG). A SG of n agents is a $2n + 2 - tuple < S, A_1, \ldots, A_n, T, R_1, \ldots, R_n >$ where (Busoniu et al. 2008):

- **S** is the state-space.
  It defines the set of possible states;
- **A**$_i$ is the action-space of agent i.
  It defines the set of possible actions;
- **T** is the transition model: $T(s, a, s') = Pr(s'|s, a)$.
  It defines the probability of reaching state s' when in s, after performing joint-action a;
- **R**$_i$ is the reward function of agent i: $R(s, a, s') = \mathbb{R}$.
  The numerical feedback provided by the environment when the agent transitions to state s' after performing joint-action a in state s.

The nature of a SG can be co-operative, competitive or a mixture of both. In co-operative games the reward function for all agents is the same. In games which are competitive, the sum of rewards received for each pair of states and joint actions is zero. SGs with a mixture of both elements are known as general sum games.

Unlike MDPs where there is a clear, single optimal policy, SGs do not follow the same principle. There can be multiple policies as some trade offs between the agents' goals need to occur. The typical solution preferred by the community is to converge to a Nash equilibrium and specifically the Pareto optimal Nash equilibrium.

A Nash equilibrium is a joint-strategy in which no agent has an incentive to change their own strategy assuming all other agents will stick to their current action selection mechanism (Nash 1951). It can be pure i.e. each agent will always play the same action, or it can be mixed where each action is chosen with a certain probability.

Formally a joint policy π^{NE} is a Nash equilibrium if:

$$\forall i \in 1 \ldots n, \pi_i \in \Pi_i \mid R_i(\pi_i^{NE} \cup \pi_{-i}^{NE}) \geqslant R_i(\pi_i \cup \pi_{-i}^{NE}) \tag{5.1}$$

where n is the number of agents, Π_i is the set of all possible policies of agent i, R_i is the reward function of agent i, π_i^{NE} is a specific policy of agent i and π_{-i}^{NE} is the joint policy of all agents except agent i. If this inequality holds for all agents, the joint policy π^{NE} of each agent following its own policy π_i^{NE} is a Nash equilibrium.

5.2 Multi-Agent Potential-Based Reward Shaping (MA-PBRS)

Reward shaping, as mentioned earlier, is a method for imparting domain knowledge to RL agents so as to improve learning. So far this thesis has discussed PBRS within the context of single-agent RL giving focus to the theoretical guarantees. To re-iterate, to avoid the problems of changing the optimal policy by providing additional rewards Ng et al. (1999) proposed the use of PBRS. PBRS defines the additional rewards that an agent receives as the difference of some potential function Φ defined over a source s and a destination state s'. More formally:

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \tag{5.2}$$

where γ must be the same discount factor as used in the agent's update rule. The potential function is a representation of the designer's, or domain expert's preference regarding specific states. If the provided knowledge is correct, then PBRS will encourage the agent to move towards the goal.

PBRS, defined according to Equation 5.2, has been proven to not alter the optimal policy of a single agent in both infinite- and finite- state MDPs (Ng et al. 1999).

Wiewiora et al. (2003) proved that PBRS is equivalent to Q-table initialisation i.e. an agent using PBRS and an agent with no shaping, but initialised with the same potential function of the

PBRS agent, will exhibit the same behaviour. Figure 2.5 shows the typical behaviour of an agent receiving PBRS, presuming a good heuristic.

Devlin and Kudenko (2011) proved that the theoretical guarantees of single-agent PBRS also hold in the multi-agent case. Specifically, MA-PBRS is equivalent to Q-table initialisation and also guarantees to maintain constant Nash equilibria.

As most of the work discussed in this thesis modifies the reward shaping function overtime, it is worth noting that recent work on potential-based reward shaping, has removed the assumptions of a static potential function from the original proof with the existing guarantees maintained even with a dynamic potential function regarding the constant Nash Equilibria (Devlin and Kudenko 2012a). MA-PBRS however is no longer equivalent to Q-table initialisation when using a dynamic potential function since if the Q-table is initialised with the potential of states before the experiment starts, future changes in potentials are not accounted for.

5.3 Experimental Design

Evaluation Domain

Evaluation of the reward shaping algorithms takes place on an extended version of the navigation maze problem, the flag collection domain which has been presented previously. In this multi-agent case however there is not only a single agent acting in the environment. Figure 5.1 shows the configuration of the maze with one agent starting at $S1$ and agent two starting at $S2$. From this they 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 collected, the scenario ends when both agents reach the goal position. At this time both the agents receive the same rewards which is equal to one hundred times the number of flags which were collected.

Figure 5.1 shows the layout of a simple version of the domain in which rooms are labelled `RoomA-E` and `HallA-B`, flags are labelled `A-F`, `S1` and `S2` are the starting positions of the agents and `G` is the goal position.

5.3.1 Results

A series of experiments were conducted in order to assess the performance of the abstract MDP reward shaping method for MARL. The agent is compared against an agent using plan-based reward shaping. The comparison is based on the performance in terms of discounted goal reward.

In order to be fair when comparing the two approaches, the same state abstraction function and options/actions are used both in the plan-based and abstract MDP methods.

In all experiments, the scaling factor of the abstract MDP method was set to:

$$\omega = MaxReward/NumStatesInMDP \tag{5.3}$$

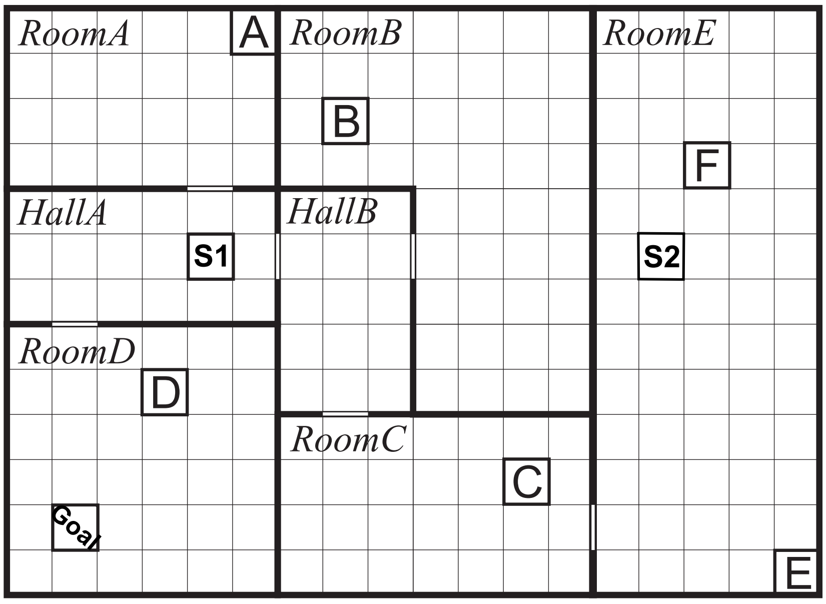


Figure 5.1: Multi-Agent Flag Collection Domain.

and the scaling factor of the plan-based method to:

$$\omega = MaxReward/NumStepsInPlan \tag{5.4}$$

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.

All agents implemented SARSA with $\epsilon-$greedy action selection and eligibility traces (Sutton and Barto 1998). For all experiments, the agents' parameters were set such that $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon = 0.3$ and $\lambda = 0.4$. The experiments were run for 30 iterations each lasting $50,000$ episodes.

For clarity all the graphs only display results up to 20000 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 of the mean.

The agents are tested in the flag collection domain described earlier and shown in Figure 5.1. In addition two scaled up versions of this maze are used; a maze with 12 flags and 7 rooms and a maze with 12 flags and 12 rooms. As mentioned earlier, I am interested in those settings where information sharing is not allowed and the agents are agnostic to other learning entities in the environment. To set an upper bound on performance however, a setting in which agents receive plan-based reward shaping from a joint-plan generated by a centralised agent is included.

Figure 5.2: Joint-Plan for Agent 1

```
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)
```

Figure 5.3: Joint-Plan for Agent 2

```
TAKE( flagF ,roomE)
TAKE( flagE ,roomE)
MOVE( roomE, roomC)
TAKE( flagC ,roomC)
MOVE( roomC, hallB )
MOVE( hallB , hallA )
MOVE( hallA ,roomD)
TAKE( flagD ,roomD)
```

Given this domain, the joint-plan of both agents that serves as an upper bound is shown in Figure 5.2 and 5.3 and its state-based transformation in Figure 5.4 and 5.5. The individual plans and value functions generated for the agents receiving decentralised shaping have been presented before in Chapters 3 and 4. Figures 5.6, 5.7 and 5.8 show the performance of the agents.

```
0 robot_in_corridorA
1 robot_in_roomA
2 robot_in_roomA taken_flagA
3 robot_in_corridorA taken_flagA
4 robot_in_corridorB taken_flagA
5 robot_in_roomB taken_flagA
6 robot_in_roomB taken_flagA taken_flagB
7 robot_in_corridorB taken_flagA taken_flagB
8 robot_in_corridorA taken_flagA taken_flagB
9 robot_in_roomD taken_flagA taken_flagB
```

Figure 5.4: Joint State Plan for Agent 1

```
0 robot_in_roomE
1 robot_in_roomE taken_flagF
2 robot_in_roomE taken_flagF taken_flagE
3 robot_in_roomC taken_flagF taken_flagE
4 robot_in_roomC taken_flagF taken_flagE taken_flagC
5 robot_in_corridorB taken_flagF taken_flagE taken_flagC
6 robot_in_corridorA taken_flagF taken_flagE taken_flagC
7 robot_in_roomD taken_flagF taken_flagE taken_flagC
8 robot_in_roomD taken_flagF taken_flagE taken_flagC taken_flagD
```

Figure 5.5: Joint State Plan for Agent 2

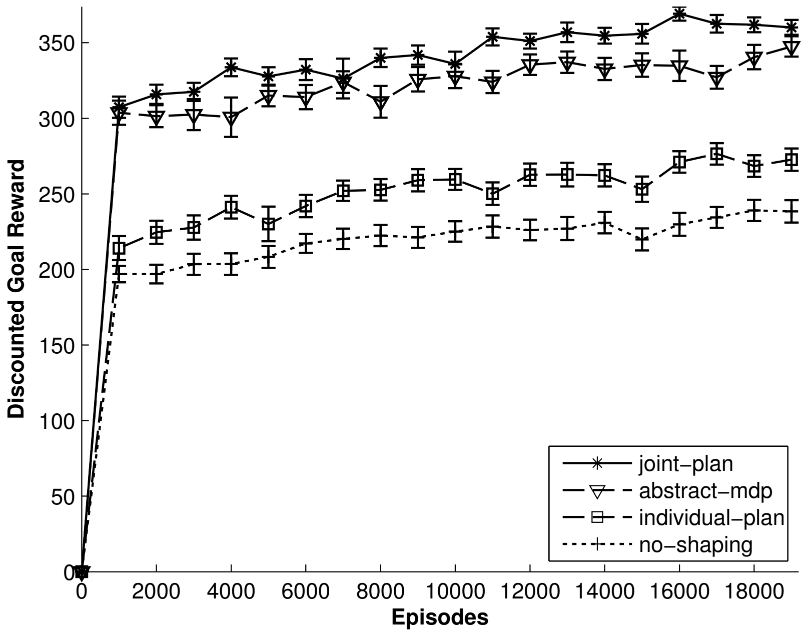


Figure 5.6: Multi-agent flag collection domain with 6 flags and 7 rooms.

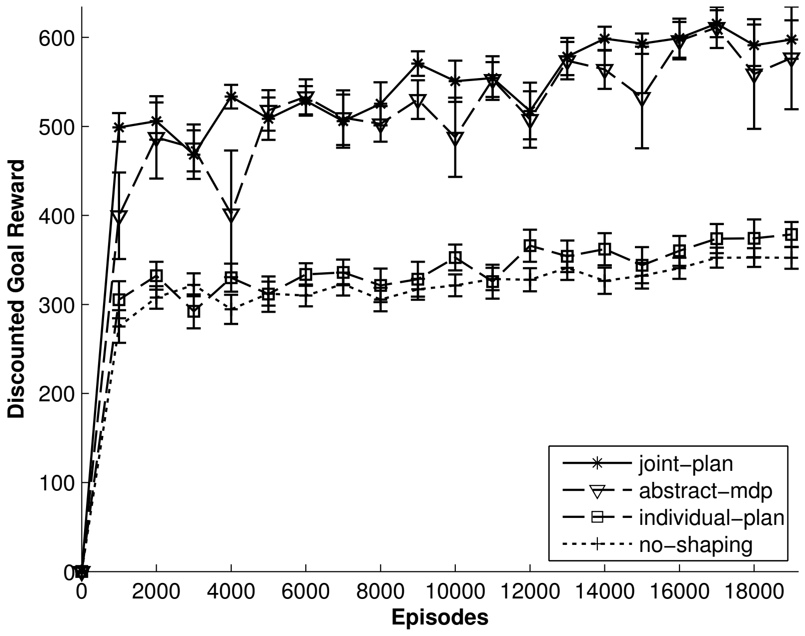


Figure 5.7: Multi-agent flag collection domain with 12 flags and 7 rooms.

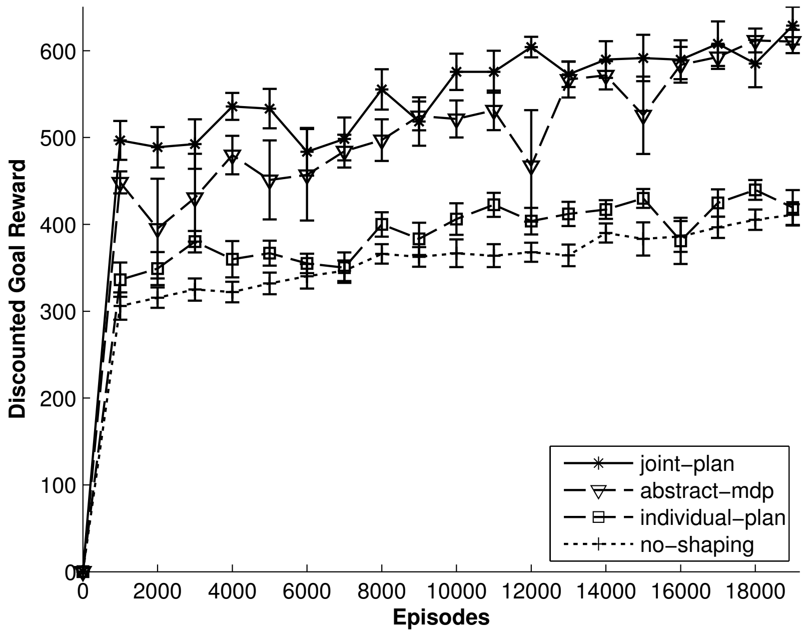


Figure 5.8: Multi-agent flag collection domain with 12 flags and 12 rooms.

The results show that the plan-based agent receiving individual shaping fails to reach a satisfying performance. Careful examination shows that the agents fail to coordinate and one of them opts out and heads to the goal location, while the other agent collects all the flags. This is due to the way plan-based shaping provides extra rewards. Certain goals in the plan cannot be satisfied and as a result only one agent is able to collect all the extra rewards and learn a better policy. A typical behaviour exhibited through the experiments is shown in Figure 5.9.

More specifically, when both agents receive individual plans, they are guided to collect all the flags that are present in the maze. The knowledge provided as a STRIPS plan contains a single path to the goal, i.e. a succession of high level states the agent will have to go through. As a result, if one of the steps cannot be satisfied by the agent, no other extra rewards can be given after that point. Consider the case where the second agent has picked up `flagC`. Any steps in the first agent's plan which contain `flagC` now cannot be satisfied and the agent is left without any reward shaping after that point.

The agents receiving abstract MDP shaping manage to achieve a performance similar to the agents receiving centralised shaping which contains joint-plans instead of individual and is provided by a centralised planner. Since the agent is agnostic to other learning entities in the environment, it follows that certain paths in the shaping function will not be encountered in simulation. For instance, consider that the second agent picks up `flagE` and `flagF` in the maze shown in Figure 5.1, this will have as a consequence the first agent never encountering those

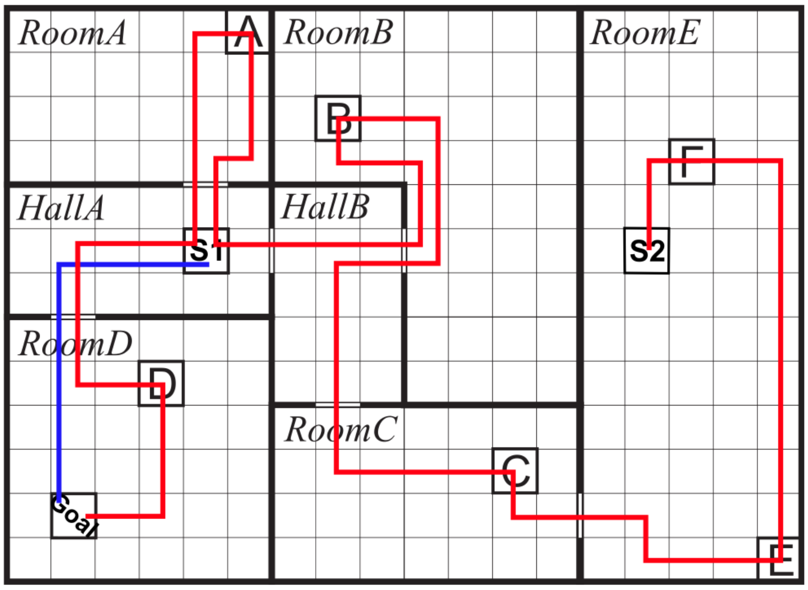


Figure 5.9: Typical behaviour of individual plan-based reward shaping.

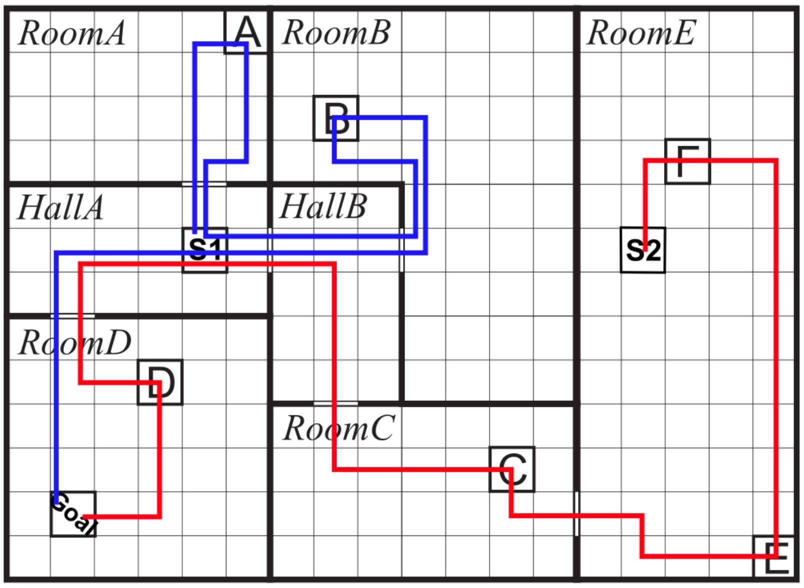


Figure 5.10: Typical behaviour of joint plan-based reward shaping and abstract MDP shaping.

states where it has collected those flags. This does not have any impact in the agent's performance since there still exist other paths in the shaping function which lead to the goal position. This is due to the fact that, contrary to the plan-based agent, the abstract MDP agent does not receive only a single path to the goal since the value function contains all the possible states the agent can be in along with their values. Therefore, both agents are able to learn a much better policy than the individual learners using plan-based shaping and the agents can cooperate efficiently without the need to use centralised shaping.

This behaviour of the abstract MDP agent is similar to that presented in the case of a single agent receiving incorrect knowledge where there was no impact in performance. A multi-agent scenario for individual shaping can be considered a special case of single agent learning where an agent is receiving incorrect domain knowledge. The difference here is not that certain elements are not present in the environment, but that a second agent is simultaneously acting in the environment thus changing its dynamics. A typical behaviour of both joint plan-based shaping and abstract MDP shaping can be seen in Figure 5.10.

5.4 Conclusion

This chapter presented the use of abstract MDP reward shaping in multi-agent RL and showed how it can be used for conflict resolution and co-ordination. The agent was compared to an agent receiving individual plan-based reward shaping and an agent receiving joint plan-based reward shaping.

It was demonstrated empirically that the abstract MDP agents can learn to co-operate efficiently and eliminate conflicting goals while the plan-based method cannot reach similar performance. This difference in performance is attributed to the type of knowledge provided by the two methods. While both are considered decentralised methods for reward shaping, abstract MDPs provide multiple paths to the goal when the plan-based method provides only a single path and has a direct impact in performance due to the conflicting goals.

In addition the agent was scaled to a maze with 12 flags and 7 rooms and a maze with 12 flags and 12 rooms and it was shown that the abstract MDP agent can still reach similar performance to the agent receiving centralised shaping even in larger environments.

Therefore, abstract MDP reward shaping can be used not only as a method to impart domain knowledge in MARL, but also as a means of conflict resolution and cooperation in decentralised reward shaping.

There is however a major drawback in this method and that is the MDP size. As mentioned previously, the size of the abstract MDP can grow very large in size while scaling to more complex domains and it can prove very difficult to use as it might take a long time to solve an MDP.

CHAPTER 6

Conclusion and Future Work

To conclude I recall the hypothesis of this research presented earlier:

> Adding knowledge revision capabilities to reinforcement learning agents utilising reward shaping can alleviate the adverse effects of erroneous domain knowledge by improving its quality and thus agents can reach a better overall performance in terms of convergence speed and learnt policy compared to agents without knowledge revision and agents that receive no shaping. Agents without knowledge revision receiving erroneous knowledge may still reach a better performance than agents without shaping.

In all the experiments presented in this thesis, it was demonstrated empirically that adding knowledge revision to agents utilising reward shaping can lead to better performance when compared to agents without knowledge revision. In order to show the adverse effects of erroneous knowledge and how they can be overcome the following domains were used for experimental evaluation: an extended version of a navigation domain, the flag-collection domain, a real-time strategy game, Starcraft: BroodWar, and a Micro UAV domain, developed by our industrial collaborators at QinetiQ which represents a real world scenario.

The research was extended by expanding to non-deterministic environments through the use of abstract MDP reward shaping with knowledge revision. It was demonstrated that even when an agent is acting in a non-deterministic environment, knowledge revision can still be efficiently used in order to overcome erroneous knowledge and thus the agent is guided by more accurate knowledge.

In a multi-agent scenario it was shown that abstract MDP reward shaping can be used not only to guide agents, but also for conflict resolution in the cases where agents are provided with decentralised shaping.

6.1 Summary of Contributions

The most significant contributions of this thesis can be summarised as follows:

Knowledge Revision in Plan-Based Reward Shaping

A series of experiments conducted in the flag-collection domain and in Starcraft:BroodWar showed that using algorithms to handle revision in plan-based reward shaping can result in agents being guided by more accurate reward shaping. As a result, agents with knowledge revision managed to reach better policies faster when compared to agents that do not revise knowledge. These results are documented in Chapter 3.

Knowledge Revision in Abstract MDP Reward Shaping

In Chapter 4 knowledge revision capabilities are incorporated to agents using abstract MDP reward shaping. In a series of experiments conducted in the flag-collection domain, it is shown that agents that constantly update the probabilities in the provided high level MDP can quickly rectify the erroneous parts of the provided knowledge and thus benefit from better shaping. The results show that the agents manage to reach a similar performance as agents using plan-based reward shaping. More interestingly, abstract MDP shaping does not require revision in the cases of incorrect knowledge. The multiple paths that reward the agent for moving closer to the goal make the agent immune to this type of erroneous knowledge.

Extending to Non-Deterministic Environments

In addition to the flag-collection domain, the abstract MDP agent is also evaluated in a non-deterministic environment to test its capabilities of dealing with a dynamic domain. In the Micro UAV problem, the set of experiments that were conducted showed that the agent can efficiently revise its knowledge even when challenged by the stochastic nature of the domain it is acting in. Extending to non-deterministic environments is a very important step in making this research applicable in a wide variety of domains. These results are documented in Chapter 4.

Conflict Resolution in Multi-Agent Reinforcement Learning

In Chapter 5 the agents using plan-based reward shaping and abstract MDP reward shaping are compared in a multi-agent environment. When both agents are provided with decentralised shaping, the plan-based method is hindered by conflicting goals and cannot reach a good enough joint policy. On the other hand, due to the multiple paths that lead the agents to the goal, the abstract

MDP method allows the agents to co-ordinate thus reaching a better joint policy, similar to agents that receive centralised shaping.

6.2 Limitations

Despite the contributions mentioned in the previous section, this research does have limitations, with some potential solutions discussed in Section 6.3, the most important of which I will list here:

Knowledge Verification in Plan-Based Reward Shaping

While knowledge verification can be very handy in order to determine whether a set of beliefs needs to be revised, it quickly becomes impractical in large domains. DFS works well in small domains but scaling to more complex ones will prove to be a daunting task. Even in environments where search can be based on sensors, an agent should be able to handle noise. Therefore in certain cases it might be better to skip knowledge verification altogether but risk revising correct knowledge and as a result take longer to learn the optimal policy. As in many cases documented in this thesis however erroneous knowledge can be better than no knowledge at all and the trade-offs are up to the designer of the system to decide.

Dealing with Erroneous Plans in Terms of Order

Plan-based reward shaping agents with knowledge revision can very efficiently handle incorrect and incomplete knowledge. However, when a plan is complete in terms of the sub-goals an agent needs to achieve, but the order of the plan is wrong, the current design of this method will not be able to detect it. The agent will start to learn how to complete the sub-goals set by the shaping function, but might take longer to complete an episode by following a sub-optimal path. This is not present in abstract MDP reward shaping as there is not a single path to the goal but all possible states are considered and assigned a value by solving the MDP.

Plan-Based Reward Shaping with Revision in Non-Deterministic Environments

While abstract MDPs are more straightforward to use in non-deterministic environments, it would be beneficial to design a method that would make plan-based reward shaping usable in those domains for comparison purposes.

Parameter Configuration in Abstract MDP Reward Shaping

In the original work on using abstract MDPs for reward shaping there was no mention on how to efficiently set the parameters of the high level MDP. In the experiments that were conducted it was shown that this method requires parameter tuning as how one sets the parameters can make or break an agent's learning. In all experiments it was found that setting γ to 0.9 resulted in better shaping. This is due to the fact that the domains used in this thesis share a similar reward function. Other domains might need further experimentation to find the sweet spot of the parameters and that is not always feasible, especially in complex domains.

Size of Abstract MDPs

Although abstract MDPs do provide the benefit of multiple paths, which as discussed result in agent not needing to revise in the case or incorrect knowledge and also being able to co-ordinate in a multi-agent setting, it is at the same time a curse. As the domains grow larger in size, so does the abstract MDP and it might take a significant amount of time to solve it which might make the method unusable in domains where time is critical.

More Agents in Conflict Resolution with Abstract MDPs

In the multi-agent setting the agents using abstract MDP reward shaping managed to co-ordinate and reach a similar behaviour to agents receiving centralised shaping. However the evaluation was conducted using only two agents. It would be beneficial to also evaluate the effects of multiple agents acting in the same environment.

6.3 Future Work

Finally I will discuss what I believe would be beneficial to experiment and further investigate in the work presented in this thesis.

Firstly, tackling non-deterministic environments using plan-based reward shaping would be very beneficial to investigate due to the simplicity of constructing a knowledge-base using STRIPS. STRIPS is a well understood and studied formalism and extending our approach to include probabilities over beliefs in the provided knowledge base and cleverly updating them according to the agent's experiences in the low level environment, could potentially provide the same benefit as in the abstract MDP reward shaping method while keeping the process of setting up relatively easy.

Secondly, it would be of great interest to explore how the abstract MDP method performs in a non-grid world domain. The next step towards that direction would be to evaluate this method in the SC:BW domain which is a qualitatively different domain compared to the flag-collection or the Micro UAV domains which are both grid worlds. Successful evaluation would further strengthen our view that the methods in this thesis are widely applicable.

Lastly, this research focused on two reward shaping methods to design knowledge revision capabilities. As shown revision methods can be added to reward shaping functions but they need to take into account the way that shaping operates e.g. literals in plan-based, transitions in abstract MDPs. Extending the design principles mentioned in this thesis to alternative reward shaping methods, not necessarily potential-based, is key for making it more accessible to researchers.

6.4 Closing Remarks

Imparting knowledge to RL agents can speed-up the learning process significantly. As research is moving away from tabula-rasa approaches to more informed agents the problem of erroneous domain knowledge needs to be addressed. Using knowledge revision principles to design agents that can rectify erroneous domain knowledge and thus improve its quality, can lead to better overall performance both in terms of convergence speed as well as learnt policy. I hope by now that the methods presented in this thesis are both easily understood and implemented by any reader and that the design principles discussed can serve as a starting point to further extend it to multiple reward shaping methods.

List of Symbols

α	Learning Rate
γ	Discount Factor
λ	Decaying Parameter
Φ	Potential Function
π	Policy
w_i	Feature Weight
$e_t(s)$	Eligibility Trace
$+$	Expansion
$\dot{+}$	Revision
$\dot{-}$	Contraction
$Q(s,a)$	Value Function
R_t	Cumulative Reward

List of Abbreviations

AI	Artificial Intelligence
AGM	Alchourrón, Gärdenfors and Makinson
MC	Monte Carlo
BWAPI	Brood War API
RL	Reinforcement Learning
MDP	Markov Decision Process
KBRL	Knowledge Based Reinforcement Learning
PBRS	Potential Based Reward Shaping
HRL	Hierarchical Reinforcement Learning
RRL	Relational Reinforcement Learning
KR	Knowledge Revision
SC:BW	StarCraft: Broodwar
SCV	Space Construction Unit
UAV	Unmanned Air Vehicle
MA	Multi-Agent
MARL	Multi-Agent Reinforcement Learn
SG	Stochastic Game
MA-PBRS	Multi-Agent Potential Based Reward Shaping

References

Alchourrón, C., Gärdenfors, P., and Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2):510–530.

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

Barto, A. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

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

Busoniu, L., Babuska, R., and De Schutter, B. (2008). A Comprehensive Survey of MultiAgent Reinforcement Learning. *IEEE Transactions on Systems Man & Cybernetics Part C Applications and Reviews*, 38(2):156.

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

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

Devlin, S. and Kudenko, D. (2012b). Plan-based reward shaping for multi-agent reinforcement learning. In *In Proceedings of the AAMAS Workshop on Adaptive and Learning Agents (ALA)*.

Dzeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52.

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

Galliers, J. (1992). Autonomous belief revision and communication. *Belief revision*, 29:220.

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

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

Grześ, M. and Kudenko, D. (2008). Plan-based reward shaping for reinforcement learning. In *International Conference on Intelligent Systems*, pages 22–29. IEEE.

Hunter, A. and Delgrande, J. (2011). Iterated belief change due to actions and observations. *Journal of Artificial Intelligence Research*, 40:269–304.

Jaidee, U. and Muñoz-Avila, H. (2012). Classq-l: A q-learning algorithm for adversarial real-time strategy games. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Kern-Isberner, G. (2001). *Conditionals in nonmonotonic reasoning and belief revision: considering conditionals as agents*.

Kunapuli, G., Maclin, R., and Shavlik, J. W. (2011). Advice refinement in knowledge-based svms. In *Advances in Neural Information Processing Systems*, pages 1728–1736.

Lau, Q. P., Lee, M. L., and Hsu, W. (2013). Distributed temporal difference learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1077–1084.

Leopold, T., Kern-Isberner, G., and Peters, G. (2008). Combining reinforcement learning and belief revision: A learning system for active vision. In *submitted to: 19th British Machine Vision Conference,(BMVC 2008)*. Citeseer.

Maclin, R., Wild, E., Shavlik, J., Torrey, L., and Walker, T. (2007). Refining rules incorporated into knowledge-based support vector learners via successive linear programming. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 22, page 584. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *International Conference on Machine learning*, page 608. ACM.

Mitchell, T. (1997). Machine learning. *Mac Graw Hill*.

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, 54(2):286–295.

Ng, A. Y., Harada, D., and 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*, pages 278–287.

Peot, M. and Smith, D. (1992). Conditional nonlinear planning. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, page 189. Morgan Kaufmann Pub.

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

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

Rosenschein, J. (1982). Synchronization of multi-agent plans. In *Proceedings of the National Conference on Artificial Intelligence*, pages 115–119.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering.

Russell, S. J. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall.

Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C., and Ram, A. (2007). Transfer learning in real-time strategy games using hybrid cbr/rl. In *Proceedings of the twentieth international joint conference on artificial intelligence*, number 1041-1046.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252.

Smith, M., Lee-Urban, S., and Muñoz-Avila, H. (2007). Retaliate: Learning winning policies in first-person shooter games. In *Proceedings of The National Conference on Artificia Intelligence*, volume 22, page 1801. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

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

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211.

Szita, I. and Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941.

Tan, M. (1993). Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*, volume 337.

Tesauro, G. J. (1994). TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.

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

Wender, S. and Watson, I. (2012). Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 402–408. IEEE.

Wiewiora, E., Cottrell, G., and Elkan, C. (2003). Principled methods for advising reinforcement learning agents. In *Proceedings of the Twentieth International Conference on Machine Learning*.

Ziparo, V. (2005). Multi-Agent Planning. Technical report, University of Rome.

Index