Automated Mixed Resolution Acyclic Tiling in Reinforcement Learning

PETER DANIEL SCOPES

Doctor of Philosophy
University of York
Computer Science

September 2015

Abstract

This thesis presents novel work on how to automatically alter a Tile Coding whilst simultaneously learning to improve both the quality of an agent's policy and its speed of learning. It also identifies the detrimental effects of transition cycles in an MDP to Reinforcement Learning and Tile Coding.

Reinforcement Learning (RL) (Sutton and Barto 1998) is a popular and widely-studied machine learning technique, where an agent learns a policy through continual interactions with an environment, based on performing actions and observing their rewards. In the basic RL formulation, in order to guarantee learning an optimal policy, an agent needs to visit each state in the environment at least once (and often repeatedly). For this reason the speed of learning does not scale well to complex environments with large state spaces.

Tile Coding (TC) (Albus 1981) is a popular value function approximation method that is able to reduce the size of a state space through approximation. In this approach, values from one or more state features are grouped into exhaustive partitions called tiles. However, as the state space becomes more granular, there is an increase of potential reduction in the precision and quality of the policy the agent is learning. As a rule of thumb, the larger the tiles are in a tiling, the faster the agent arrives at its final policy but the lower its quality; the smaller the tiles are in a tiling, the slower the agent arrives at its final policy but the higher its quality. Furthermore, using multiple, offset tilings can improve performance without the need for smaller tiles.

The guarantees that surround common RL algorithms revolve around being able to visit every state in the environment at least once. However, many implementations of these algorithms use episode roll outs and can find themselves looping through a cycle of state-action pairs. This thesis theoretically and empirically shows that if the reward of each state-action pair in this transition cycle is identical then it is possible for the agent to temporarily diverge from learning the optimal policy. These detrimental effects of transition cycles can occur at any point of learning and, therefore, RL algorithms must heed them or risk sudden, temporary lacklustre performance. Furthermore, we consider the use of TC in conjunction with RL and find that it aggravates the detrimental effects of transition cycles to learning. This is caused by tiles inducing transition cycles.

Tile Coding is still an effective and efficient method of approximation when the detrimental impacts of transition cycles are avoided. This motivates us to create a novel strategy for manual tile placement called Mixed Resolution Acyclic Tiling (MRAT). MRAT is based on heuristics derived from theoretical work and empirical studies conducted in this thesis. MRAT is empirically demonstrated to be a very effective way of improving the speed and quality of learning by using a non-uniform tile placement. MRAT is then automated and is empirically shown to outperform the state-of-the-art competitors and fixed TC. Automated MRAT (AMRAT) does not require parameter tuning and therefore has no hidden costs for its use unlike its competitors.

List of Contents

Abstract . . . 3
List of Contents . . . 4
List of Tables . . . 7
List of Figures . . . 8
Acknowledgements . . . 12
Declaration . . . 13

1 Introduction and Motivation **14**
1.1 Motivation . . . 14
1.2 Theoretical Properties of Transition Cycles . . . 15
1.3 Application to Tile Coding . . . 15
1.4 Automation of MRAT . . . 16
1.5 Thesis Structure . . . 16

2 Background and Field Review **18**
2.1 Artificial Intelligence . . . 18
2.1.1 Machine Learning . . . 19
2.1.2 Agent Paradigm . . . 19
2.1.3 Markov Decision Process . . . 23
2.1.3.1 Dynamic Programming . . . 23
2.1.4 Gradient Descent . . . 24
2.1.5 Reinforcement Learning . . . 25
2.1.6 Algorithms . . . 26
2.2 Generalisation . . . 30
2.2.1 Linear Methods . . . 31
2.2.2 Coarse Coding . . . 31
2.2.2.1 Tile Coding . . . 32
2.2.3 Algorithms . . . 34
2.2.4 Directions of Research . . . 37
2.3 Scope and Open Problems . . . 40
2.3.1 Credit Assignment Problem . . . 40
2.3.2 Scalability and Dimensionality . . . 40
2.3.3 Knowledge and Heuristics . . . 40

2.3.4 Design . . . 41
2.3.5 Function Approximation . . . 41
2.3.6 Scope of the thesis . . . 43

3 Theoretical Properties of Transition Cycles 44
3.1 Motivation . . . 44
3.2 Assumptions . . . 45
3.3 Theoretical Properties of SARSA . . . 46
3.3.1 Formal Proofs . . . 47
3.3.2 Theorems . . . 56
3.3.3 Empirical Demonstration . . . 58
3.3.3.1 Experiments & Results . . . 58
3.3.4 Conclusion . . . 60
3.4 Extension to Q-Learning . . . 61
3.4.1 Altered Assumptions . . . 61
3.4.2 Formal Proofs and Theorems . . . 62
3.4.3 Empirical Demonstration . . . 63
3.4.4 Conclusion . . . 64
3.5 Conclusions . . . 65

4 Application to Tile Coding 69
4.1 Application to Tile Coding . . . 69
4.2 Parameter Search . . . 75
4.2.1 Experiment set up . . . 75
4.2.2 Results . . . 76
4.2.3 Conclusion . . . 77
4.3 Impacts of Tile Shape . . . 78
4.3.1 Shape of the split tile . . . 79
4.3.2 Experiments & Results . . . 80
4.3.2.1 2d-Random Walk (2d-RW) . . . 80
4.3.2.2 Results of Tile Shape . . . 81
4.4 Heuristics . . . 83
4.4.1 Heuristics from theory . . . 83
4.5 Heuristic-based Tile Placement . . . 84
4.5.1 Mixed Resolution Acyclic Tiling . . . 85
4.5.2 Experiments & Results . . . 86
4.5.2.1 Mud World Environment . . . 86
4.5.2.2 Results . . . 86
4.6 Conclusions . . . 87

5 Automation of MRAT 96
5.1 Motivation . . . 96
5.2 Automation of MRAT . . . 97
5.2.1 Implementation Options . . . 98
5.2.2 Evaluation of the implementations . . . 101
5.2.3 Algorithmic Automation . . . 116
5.3 Experimentation & Results . . . 117
5.3.1 Comparative impacts of the unlearning phenomenon . . . 117
5.3.1.1 Experiment set up . . . 118
5.3.1.2 Results . . . 119

5.3.1.3 Conclusions 121
5.3.2 Performance Evaluation 121
5.3.2.1 Results of AMRAT 124
5.4 Conclusions 129
5.4.1 Discussion 129

6 Conclusion and Future Work **130**
6.1 Contributions and Conclusions 130
6.2 Limitations and Future Work 131

References **133**

List of Tables

4.1 Average rewards of Tile Shapes on a range of environment sizes 82

List of Figures

2.1 Agents perceive from and act upon environments. 20
2.2 A coarse coding over a two-dimensional, continuous state space where state X activates 3 groupings (Sutton 1988). 31
2.3 Multiple, overlapping grid tilings 32
2.4 Tile coding examples with a different resolution. Three tilings with tiles of three units in 2.4a and six units in 2.4b (Grzes and Kudenko 2009) 35

3.1 Examples of IRTCs in a sub-MDP 46
3.2 MDP environments to demonstrate theory 58
3.3 Results from an MDP consisting purely of a single IRTC of length 1. Transition Length = 1, $\alpha = 0.899628$, $\gamma = 0.63834$, $r = -47.018$ 59
3.4 Results from an MDP consisting purely of a single IRTC of length 1. Transition Length = 1, $\alpha = 0.83881$, $\gamma = 1$, $r = -44.6894$ 60
3.5 Results from an MDP consisting purely of a single IRTC of length 1. Transition Length = 1, $\alpha = 0.74085$, $\gamma = 1$, $r = 21.39147$ 61
3.6 Results from an MDP consisting purely of a single IRTC of length 2. Transition Length = 2, $\alpha = 0.756187$, $\gamma = 0.168648$, $r = -78.1331$ 62
3.7 Results from an MDP consisting purely of a single IRTC of length 2. Transition Length = 2, $\alpha = 0.094648$, $\gamma = 1$, $r = -25.0441$ 63
3.8 Results from an MDP consisting purely of a single IRTC of length 2. Transition Length = 2, $\alpha = 0.683996$, $\gamma = 1$, $r = 63.0188$ 64
3.9 Results from an MDP consisting purely of a single IRTC of length 7. Transition Length = 7, $\alpha = 0.250782$, $\gamma = 0.54597$, $r = 31.16617$ 65
3.10 Results from an MDP consisting purely of a single IRTC of length 7. Transition Length = 7, $\alpha = 0.886697$, $\gamma = 1$, $r = -26.6789$ 66
3.11 Results from an MDP consisting purely of a single IRTC of length 7. Transition Length = 7, $\alpha = 0.282216$, $\gamma = 1$, $r = 42.75248$ 66
3.12 SARSA results from the MDP in Figure 3.2a 67
3.13 SARSA results from the MDP in Figure 3.2b 67
3.14 Q-Learning results from the MDP in Figure 3.2a 67
3.15 Q-Learning results from the MDP in Figure 3.2b 68

4.1 Examples of IRTCs within in a tile 70
4.2 Random Walk Environments (Agents moves from the red circle to a green square) 71
4.3 Example of Q-Value convergence in 1d-RW (Tile size 4, Neutral step reward, No epsilon decay, Every update over a 500 update period) 73
4.4 Example of Q-Value convergence in 2d-RW (Tile size 4x4, Neutral step reward, No epsilon decay, Every update over a 500 update period) 74
4.5 Frequency of Unlearning Heat Map, k (Tile Size) x α (Learning Rate) k increases from left to right per graph, α increases from bottom to top per graph γ increases along the x-axis in each graph, ϵ increases along the y-axis in each graph 88
4.6 Frequency of Unlearning Heat Map, k (Tile Size) x ϵ (Exploration Factor) k increases from left to right per graph, ϵ increases from bottom to top per graph γ increases along the x-axis in each graph, α increases along the y-axis in each graph 89
4.7 Examples of different ways to split tiles 90
4.8 A 2d-Random Walk Environment (Agents moves from the red circle to a green square) 90
4.9 Examples of different experiment set-ups 91
4.10 An example Mud World Environment (Agents move from the red circle to the green square, brown squares are mud, grey squares are road) 92
4.11 Mud World with one optimal route. Q-Learning vs MRAT. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 92
4.12 Mud World with a few optimal routes. Q-Learning vs MRAT. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 92
4.13 Mud World with many optimal routes. Q-Learning vs MRAT. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 93
4.14 Mud World with a few optimal routes. MRAT with all optimal routes in high resolution vs 1 optimal route in high resolution. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 93
4.15 Mud World with many optimal routes. MRAT with all optimal routes in high resolution vs a few of the optimal routes in high resolution vs 1 optimal route in high resolution. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 94
4.16 Mud World with one optimal route. MRAT with high-medium-low resolutions vs MRAT with high-low resolutions. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 94
4.17 Mud World with a few optimal routes. MRAT with high-medium-low resolutions vs MRAT with high-low resolutions. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 95

5.2 Puddle World. Different parameter settings for Unlimited Consecutive. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 103
5.3 Mountain Car. Different parameter settings for Unlimited Consecutive. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 103
5.4 Puddle World. Different parameter settings for Unlimited Plateaued. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 104
5.5 Mountain Car. Different parameter settings for Unlimited Plateaued. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 104
5.6 Puddle World. Different parameter settings for Fixed Consecutive. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 105
5.7 Mountain Car. Different parameter settings for Fixed Consecutive. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 106
5.8 Puddle World. Different parameter settings for Fixed Plateaued. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 106

5.9 Mountain Car. Different parameter settings for Fixed Plateaued. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 107
5.10 Puddle World. Comparison of the best IRTC detection implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 107
5.11 Mountain Car. Comparison of the best IRTC detection implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 108
5.12 Puddle World. Results of Episode History combination. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 109
5.13 Mountain Car. Results of Episode History combination. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ 109
5.14 Puddle World - Backward. Results of Greedy Plateaued combination. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 110
5.15 Mountain Car - Backward. Results of Greedy Plateaued combination. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 110
5.16 Puddle World - Forward. Results of Greedy Plateaued combination. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 111
5.17 Mountain Car - Forward. Results of Greedy Plateaued combination. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 111
5.18 Puddle World. Current greedy policy with plateaued learning to decide when to split tiles. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 112
5.19 Mountain Car. Current greedy policy with plateaued learning to decide when to split tiles. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 113
5.20 Puddle World. Comparison of the best IRTC detection implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 113
5.21 Mountain Car. Comparison of the best IRTC detection implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 114
5.22 Puddle World. Comparison of the two combination implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 114
5.23 Mountain Car. Comparison of the two combination implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 115
5.24 Cart Pole Environment. Objective: balance the pole by moving the cart left and right. Image from: `http://library.rl-community.org/wiki/CartPole_(Java)` . . . 119
5.25 Performance of AMRAT and TC in the Puddle World environment. Where m is the number of tilings and n is the number of tiles per feature. . . . 120
5.26 Performance of AMRAT and TC in the Cart Pole environment. Where m is the number of tilings and n is the number of tiles per feature. . . . 120
5.27 Puddle World. Parameter tuning for fixed Tile Coding. Where $m = 1, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 122
5.28 Puddle World. Parameter tuning for fixed Tile Coding. Where $m = 5, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 122
5.29 Puddle World. Parameter tuning for fixed Tile Coding. Where $m = 10, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 123
5.30 Puddle World. Parameter tuning for fixed Tile Coding. Comparison of the best. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 123
5.31 Mountain Car. Parameter tuning for fixed Tile Coding. Where $m = 1, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 124
5.32 Mountain Car. Parameter tuning for fixed Tile Coding. Where $m = 5, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 124

5.33 Mountain Car. Parameter tuning for fixed Tile Coding. Where $m = 10, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 125
5.34 Mountain Car. Parameter tuning for fixed Tile Coding. Comparison of the best. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 125
5.35 Puddle World. Parameter tuning for Adaptive Tile Coding. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 126
5.36 Mountain Car. Parameter tuning for Adaptive Tile Coding. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 126
5.37 Puddle World. AMRAT state-of-the-art comparison. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 127
5.38 Puddle World. Vs Tile Coding state-of-the-art comparison. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 127
5.39 Mountain Car. AMRAT state-of-the-art comparison. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 128
5.40 Mountain Car. Vs Tile Coding state-of-the-art comparison. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$ . . . 128

Acknowledgements

Daniel, thank you for all the advise, suggestions, ideas, enthusiasm, patience, and guidance over the years. Thank you for the many hours of your time you given to me and all the proof-reading that you have done. Thank you for the freedom you allowed me to explore my ideas. It has been a pleasure to be supervised by you.

Grace, thank you for your patience when my mind was consumed by this thesis. Thank you for supporting me through the highs and lows, and for all the encouragements you gave despite the set backs. Thank you for your help, your kindness, and your love. And thank you for helping me look up and see the bigger picture when I was lost in a some intricate detail.

To my examiners James Cussens and Chris Watkins. Firstly to James, thank you for your advise and wisdom throughout my PhD. Thank you for your incessant desire to see British English at British universities, I am sorry for my inability to grammar. And thank you for always being a friendly face. Secondly to Chris, thank you for the enthusiasm you showed toward my thesis at my viva. Thank you for being a captive audience whilst I defended my work. And thank you for the constructive feedback which lead me to an improved thesis.

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. Peter Scopes and Daniel Kudenko. Theoretical Properties and Heuristics for Tile Coding In *Proceedings of ALA Workshop, at the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014. ACM Press.

2. Peter Scopes and Daniel Kudenko. Automated Mixed Resolution Tiling In *Proceedings of ALA Workshop, at the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2015. ACM Press.

CHAPTER 1

Introduction and Motivation

This chapter introduces the thesis by motivating the contributions, stating the contributions and summarising the structure of the thesis.

This chapter begins by providing motivation for the thesis. Next, the contributions to the theoretical knowledge from chapter 3 are stated. Then this chapter states the contributions of the empirical study from chapter 4. After this the contributions to the state-of-the-art from chapter 5 are stated. Finally, this chapter summarises the structure of the thesis.

1.1 Motivation

Reinforcement Learning (RL) is a popular and widely-studied machine learning technique, where an agent learns a policy through continual interactions with the environment, based on performing actions and observing their rewards. In the basic RL formulation, in order to learn an optimal policy, an agent may need to visit each environment state and perform all possible actions in that state at least once (and often repeatedly). For this reason the speed of learning does not scale well to complex environments with large state spaces.

In order to deal with large state spaces, a popular technique employed in RL is Tile Coding (TC) (Albus 1981). In this approach, values from one or more state features are grouped into exhaustive partitions, called tiles. This reduces the size of the state space, and thus TC enables an agent to learn more quickly. However, with the reduction of state space granularity also comes a potential reduction in the precision and quality of the learnt policy.

TC has been shown to be an effective and efficient method of approximation. However, the effect of its use in conjunction with RL is yet to be fully understand theoretically. It is known

that the guarantees of learning the optimal policy that techniques like Q-Learning have, do not extend in the general case to include TC. Moreover, there are nuances of TC performance that are difficult to explain. TC can require a large amount of trial-and-error to find a parameter setting that learns quickly with high performance. This provides motivation to study TC theoretically and create algorithms that do not require parameter tuning.

1.2 Theoretical Properties of Transition Cycles

Techniques like SARSA and Q-Learning are well known and well studied both theoretically and empirically. However, there are still instances where these algorithms behave unexpectedly or hard-to-predict ways. This motivates further study both empirically, and then underpinned, theoretically as to the causes of unexpected behaviour. Moreover, the deeper the understanding we have of how RL algorithms interact with MDPs the better we can apply them and avoid negative impacts that may be inherent in certain MDPs.

In chapter 3 of this thesis we will uncover new theoretical properties in general Reinforcement Learning. They apply to all MDPs that contain transition cycles where the transition reward at every point in the cycle is identical. The properties are first applied to learning agents that use the SARSA update rule and then these are extended to include agents using the Q-Learning update rule. The MDPs these theoretical properties apply to are more restricted in the extension to Q-Learning. Theorem 1 correlates the probability that an agent will perform consecutive loops through an identical reward transition cycle and the probability the agent's policy will temporarily diverge away from the optimal policy. This temporary divergence from learning the optimal policy can occur at any point in learning where an agent finds itself performing consecutive loops through an identical reward transition cycle. Theorem 2 explicitly states the two possible means of decreasing the probability that interactions with identical reward transition cycles will have negative effects on learning. Specifically, decreasing the rate at which Q-values converge or decreasing the probability an agent will perform consecutive loops through an identical reward transition cycle.

1.3 Application to Tile Coding

There is currently a lack of theoretical understanding of how Tile Coding interacts with standard RL learning algorithms such as SARSA or Q-Learning. For example, it is common to see an agent's optimal performance wavering in results where TC is used; in contrast, results where an RL algorithm is used its performance remains constant once the agent reaches optimal policy. This could be a inherent property of environments with stochastic noise but there could also be more to understand.

In chapter 4 of this thesis we apply the theoretical properties described in the previous chapter to the function approximation method, Tile Coding. We then conduct an empirical study into the impact of tile shape on learning. From the applied theory and empirical study we derive heuristics

of learning with TC. Finally, we present a tile placement strategy derived from those heuristics called Mixed Resolution Acyclic Tiling (MRAT). The application of the theory regarding transition cycles to TC shows that tiles can induce identical reward transition cycles in MDPs where previously only identical reward paths existed. Therefore, reducing the number of states contained within a tile may improve the reliability of learning on that tile. The empirical study of tiles shapes showed that biasing more significant state features when constructing tile placements can speed up learning. However, the opposite is also true and therefore it is ill-advised to bias state features without high confidence that the significant state features have been correctly identified. Finally, the heuristics derived state that tiles should not be too large and that on or near the optimal path(s) through the MDP tiles should be small. These heuristics were empirically confirmed to be beneficial to the speed of learning.

1.4 Automation of MRAT

Due to the popularity of TC and because of the big impact tile design has on learning performance, the ability to design good tilings that speed up an agent's learning while preserving the quality of the learned behaviour is highly important. However, tile design is a hard and understudied problem that often requires good knowledge and understanding of the agent's environment. In cases where such a deep understanding is not available, a designer is left with trying different configurations of tilings until a satisfactory performance is reached, which is a lengthy and sometimes infeasible procedure, which could be made obsolete by an automated method for good tile design. A first approach, Adaptive Tile Coding (ATC) (Whiteson et al. 2007), addressed this problem by starting with a few large tiles and intelligently splitting the tiles based on various criteria. However, the implementation of ATC requires the transition function to be known which is not assumed in the general case of RL. It nevertheless does provide further evidence that intelligent splitting of tiles over time during learning improves performance (Sherstov and Stone 2005).

In chapter 5 of this thesis we create an automated version of the manual tile placement strategy MRAT. Automated MRAT (AMRAT) begins with a uniform coarse tile coding and over time moves toward an MRAT tile coding. This chapter examined different methods of implementing these two detection mechanisms. These different implementations were trialled and compared against each other. Then the best implementation was compared to state-of-the-art methods competitors. The best implementation of AMRAT proved to at least match if not better current state-of-the-art algorithms. Furthermore, since finding the best tiling using fixed TC takes many trials, AMRAT is beneficial since it does not require trials.

1.5 Thesis Structure

The thesis begins with a thorough background and field review of the current literature concluding with the open problems and scope of the thesis. It then presents new theoretical properties of

transition cycles within MDPs that can ill-effect reinforcement learning. Specifically looking at two popular RL algorithms: SARSA and Q-Learning. Next it applies these theoretical properties to a function approximation method called Tile Coding. Then, it presents a new algorithm called Automated Mixed Resolution Acyclic Tiling (AMRAT) that uses heuristics derived from the applied theory presented in this thesis. Finally, this thesis concludes its contributions and presents possible future work.

CHAPTER 2

Background and Field Review

This chapter presents definitions, history, and reviews the literature of the topics covered in this thesis. Furthermore the state-of-the-art algorithms are presented and discussed along with open problems and the scope of this thesis.

This chapter starts by motivating the need for computer intelligence. Next to be presented are definitions of the agent paradigm, dynamic programming, Markov decision processes, and reinforcement learning. Followed by the presentation and discussion of state-of-the-art reinforcement learning algorithms. This chapter proceeds to motivate generalisation and define value function approximation; specifically, gradient descent, coarse and tile coding, radial basis functions, and Kanerva coding. The definitions are followed by the presentation and discussion of the state-of-the-art tile coding algorithms. Finally, this chapter closes with the presentation and discussion of open problems and the scope of this thesis.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is the field of study concerned with the capacity of computers or other machines to exhibit or simulate intelligent behaviour (OED-Online 2014). Or the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and transition between languages (Oxford-Dictionaries 2014). AI is a broad and varied subject covering a multitude of disciplines.

In Computer Science, AI usually comprises of one of the two following ideologies:

- the study and creation of human-like intelligence; and,
- the study and creation of practical algorithms for solving problems.

The former ideology assumes that human intelligence is the product of a complex biological machine. Therefore, such a machine can be studied and replicated. The main field of research that accompanies this ideology is Neural Networks, an artificial replication of the network of neurons that exist in the biological brain. This is outside the scope of this PhD thesis and shall not be discussed further.

The latter ideology assumes that there is a process by which humans use to decide on an action or choice. Therefore, algorithms can be created and studied for solving problems. These problems can be specific to a single problem domain or general and applicable to many problems. Where the problem being solved is difficult, such as problems with large state spaces, it is often infeasible, due to time or money constraints, to create an algorithm that has a predefined action for every eventuality. One popular solution is Machine Learning.

2.1.1 Machine Learning

Machine learning is the field of construction and analysis of algorithms that can learn from data. Such algorithms operate by storing and updating internal knowledge based on the inputs it receives and using those to make predictions or decisions.

There are three broad categories that Machine learning algorithms are classified into depending on the way that it tackles the problem:

- *Supervised Learning*: the algorithm is presented with inputs and their desired outputs by a "teacher". The algorithm's aim is to learn the general rule that maps inputs to outputs.
- *Unsupervised Learning*: the algorithm is given no labels about the input it receives, rather it aims to learn on its own accord (discovering hidden patterns in data).
- *Reinforcement Learning*: the algorithm interacts with a dynamic environment in which it aims to maximise its long-term, discounted reward. After performing an action, the algorithm receives a reward and uses this signal to update its knowledge.

This PhD thesis is interested in the study and creation of practical algorithms for solving problems. Furthermore, it is interested in solving problems where there is no teacher available and the agent must interact with its environment to find a solution. Therefore, this PhD thesis is interested in Reinforcement Learning. Supervised and Unsupervised Learning fall outside of the scope of this thesis and shall not be discussed further.

2.1.2 Agent Paradigm

The agent paradigm is a particular way of viewing problem solving and is popular in the field of Reinforcement Learning (see Section 2.1.5). This paradigm supposes there is an Environment and an Agent. The environment represents the problem and the agent represents the algorithm used to solve the problem. Figure 2.1 visualises the agent paradigm where an agent perceives the

environment using sensors, uses that information in a decision process, and then acts upon the environment potentially altering the state of the environment.

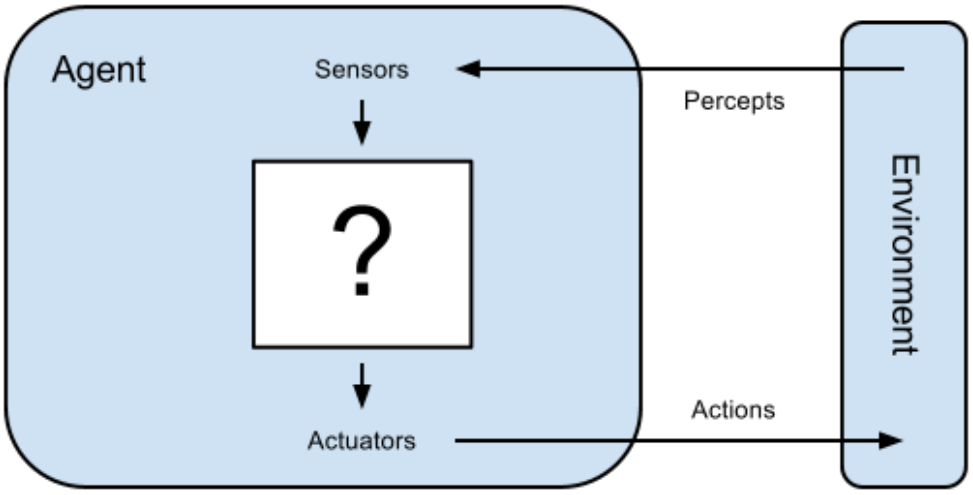


Figure 2.1: Agents perceive from and act upon environments.

Formally, at a minimum, an environment consists of a 3-tuple $\langle S, A, T \rangle$ where S is the set of states $S = \{s_0, s_1, \ldots, s_n\}$, A is the set of actions $A = \{a_0, a_1, \ldots, a_m\}$, and $T(s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s, a, s') \rightarrow Pr(s'|s, a, s_0, a_0, \ldots, s_{t-1}, a_{t-1})$ is the transition probability function of arriving in state s' from s when action a is performed with the history of previous states and actions $s_0, a_0, \ldots, s_{t-1}, a_{t-1}$. A state is an k-tuple consisting of k features that uniquely describe the environment in that state. A state feature is a member of a finite or infinite set, for example the day of the week or an integer. An agent follows a policy $\pi \in \Pi$ where Π is the set of all possible policies. A policy $\pi(s, s_0, a_0, \ldots, s_{t-1}, a_{t-1}) \rightarrow A, \forall s \in S$ that determines which action a should be performed in state s with the history of states and actions $s_0, a_0, \ldots, s_{t-1}, a_{t-1}$. The agent's policy may also incorporate further information that is stored and updated internally, such as beliefs or knowledge. There is a subset of policies $\Pi^* \subseteq \Pi$ which solve the problem. Therefore, an agent is said to have solved the problem if it follows a policy in $\pi^* \in \Pi^*$. The policies in Π^* may comprise of any mixture of similar and differing behaviours resulting in the agent achieving similar or differing tasks. Since multiple policies may be able to solve a problem, some of these policies may be more desirable than others. The environment can, therefore, also provide utility functions $U(\pi) \rightarrow \Re$ that allow the policies to be ordered. These utility functions would be used to measure a policy's performance. Utility functions may be: correlated, meaning that maximising one is mutually beneficial to another; conflicting, meaning that maximising one would be detrimental to another; independent, meaning maximising one has no affect on anther; or some mixture of these.

There is no limit to the number of different ways an agent's policy can determine its next action in an environment. Usefully, Russell and Norvig (2003) define four classes of agents which principally underlie almost all intelligent systems:

Simple reflex agents

A simple reflex agent is the simplest kind of agent; it uses only the current percept in the selection of its next action to take, ignoring anything that has happened previously. Most of the staple reinforcement learning algorithms, Q-Learning and SARSA, are learning reflex agents.

Model-based reflex agents

A model-based reflex agent which remembers, to some degree, what it has perceived; in this way it has a model of the environment which encapsulates more information about the environment than can be seen at any one time. In a similar way to the reflex agent it would, after updating its model with the current perception, make a decision on its next action based on the model.

Goal-based agents

Goal-based agents have some notion of a goal to achieve and so determines its next action in regard to achieving its goal(s). The key difference from the reflex agent is that goal-based agents are able to pre-empt and start to do something instead of just reacting.

Utility-based agents

Utility-based agents further this ability of achieving goals by considering a goal as something that can be partially completed; this differs from goal-based agents which see goals in a binary manner as completed or not. For example in chess there are goals such as checkmating your opponent, protecting your own pieces, taking your opponents pieces, etc.; these can all be decidedly completed or not. Whereas in poker, goals to maximise your profit and minimising your opponents dominance are harder to define as completed.

There are also different classifications of environments based on their characteristics:

Fully-/Partially- Observable Environments

Fully-observable environments allow an agent's sensors to capture the complete state of the environment. Partially-observable environments limit the scope of the agent's sensors to a local part of the environment.

Deterministic/Stochastic Environment

Deterministic environments have state transitions that are purely determined by the action(s) performed and the current state. Stochastic environments have probabilistic state transitions. Note that partially-observable environments may appear to be stochastic since

there may not be enough information in the current observation or the environment may be too complex to accurately calculate the state transition.

Episodic/Sequential Environment

Episodic environments have independent episodes where actions performed in one episode have no bearing on future episodes. Sequential environments have no independent actions; any previous action can impact the current state.

Static/Dynamic Environment

Static environments only have their states altered when actions are performed. Dynamic environments can alter their state at any point thus the state of the environment an agent observed may not reflect the current state of the environment even if no actions have been performed.

Discrete-/Continuous-time Environment

Discrete-time environments experience time in a turn- or move-based manner. Time increments when an agent has its turn. Continuous-time environments experience time continuously. The longer an agent takes to decide on a next action the more time will have passed in the environment.

Single-/Multi-agent Environment

Single-agent environments contain a single agent. Multi-agent environments (or Multi-Agent System) contain at least two agents; however, agents may not be aware of the presence of other agents.

If an agent is provided with a policy that solves the environment it has no need to learn; especially if that policy maximises the utility function(s) provided by the environment. However, we are interested in agents that can solve problems. That is an agent that can alter a policy that initially does not solve the problem to one that does. Furthermore, the final policy should aim to maximise the utility function(s) provided by the environment. A Learning Agent is any agent that is able to use observations to alter its beliefs or knowledge. These beliefs or knowledge would be then used in the agent's decision process. All the classes of agents defined above can, in general, be converted into learning agents.

Environments where the next state is only dependent on the current state and not a sequence of previous states is called Markovian. This means that an agent does not need to know its history in order to correctly determine the best next action to take. Environments which are not Markovian can be converted into them. This is done by adding extra information into the state to account for the artefacts of history needed. Furthermore, Markovian environments are desirable since they are well studied theoretically and empirically.

2.1.3 Markov Decision Process

> A Markov Decision Process (MDP) is a model for decision making in an uncertain, dynamic world. The (single) agent starts out in some state, takes an action, and receives some immediate rewards. The state then transitions probabilistically to some other state and the process is repeated. (Shoham and Leyton-Brown 2009)

A Markov Decision Process (MDP) is a tuple $\langle S, A, T, R \rangle$, where S is the state space, A is the action space, $T(s, a, s') = Pr(s'|s, a)$ is the probability that action a in state s will lead to state s', and $R(s, a, s')$ is the immediate reward r received when action a taken in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. The Markov Decision Process (MDP) extends the Markov Chain (a mathematical system that undergoes transitions between a countable number of states).

An MDP can be seen as a sub-class of Environments as described in Section 2.1.2; an MDP always provides a utility function in the form of the sum of all rewards from the reward function $\sum_{t=1}^{\infty} R(s_{t-1}, a_{t-1}, s_t)$. However, the policy and transition functions do not take into account the history of previous states and actions $s_0, a_0, \ldots, s_{t-1}, a_{t-1}$ due to the Markovian nature of the MDP.

2.1.3.1 Dynamic Programming

The policy that maximises the reward function of an MDP can be constructed using Dynamic Programming. This is because dynamic programming can solve any task where the transition probabilities and reward function are available (Bertsekas 2007). A learner consists of an algorithm and a set of updated knowledge; Dynamic Programming is the foundation that the majority of Reinforcement Learning algorithms (see Section 2.1.6) are built on. Specifically Dynamic Programming (DP), first introduced by Bellman (1957b,a), uses the notion that some complex problems are made up of simpler sub problems; in this way DP solves the complex problems by breaking down into the sub problems and solving those. The Bellman Equations are then a means of assigning values to the sub problems. Given some state s of a problem; a set of valid actions available in that state $a \in A(s)$; a reward function, $R(s, a)$, to calculate the pay-off to performing an action a in state s; a transition function $T(s, a)$ to calculate the new state arrived in; and a future discount factor γ for future state values the Bellman Equation can be defined recursively as:

$$V(s) = \max_{a \in A(s)} \Big(R(s, a) + \gamma V\big(T(s, a)\big) \Big) \tag{2.1}$$

where the action a is chosen to maximise the pay-off and the discounted value of the next state. DP is widely believed to be the only feasible way of solving general stochastic optimal control problems.

The transition probabilities and reward function of an environment are not always available. In such circumstances DP cannot be used to construct the policy directly. One solution would be to learn the transition probabilities and whenever they changed use DP to construct the new policy. However, repeatedly using DP can be costly. Another option is to combine an iterative approximation approach like Gradient Descent (see 2.1.4) with the Bellman Equation (see 2.1.5).

2.1.4 Gradient Descent

Gradient Descent is a method of iteratively stepping closer to a local minimum of a function. The function must be defined and differentiable. Since the function is differentiable the gradient can be calculated. Then a proportion of the negative of the gradient is taken toward the local minimum; hence the name *gradient descent*. As gradient descent moves its approximation a proportion of the gradient each iteration, if the approximation moved closer to the local minimum the gradient will be smaller and therefore will move a smaller amount in the next step[1]. In this way gradient descent is able to asymptomatically converge toward the local minimum.

For example, Algorithm 1 defines a function that given an initial value x will return a value within the given precision p of the local minimum. Formally it takes an initial value x, a single-variable, differentiable function F, a step size ε, and a precision p. Each iteration of the while loop in Algorithm 1 will cause the approximation x_{aprx} to move by $\varepsilon \Delta F(x_{aprx})$. In this example the step size is fixed and does not change over time; this can lead to the gradient descent to fail by diverging from the minimum.

Algorithm 1 Gradient-Descent(x, F, ε, p)

$x_{prev} \leftarrow 0$
$x_{aprx} \leftarrow x$
$\Delta F \leftarrow$ the derivative of F
while $abs(x_{aprx} - x_{prev}) > p$ **do**
  $x_{prev} \leftarrow x_{aprx}$
  $x_{aprx} \leftarrow x_{prev} - \varepsilon \Delta F(x_{prev})$
return x_{aprx}

Gradient descent can easily be modified to iteratively step closer to a local maximum of a function. Instead of using the negative of the gradient, the unaltered gradient is used. This is called Gradient ascent.

In general, gradient descent can be applied to any multi-variable, differentiable function. Furthermore, once the derivative has been calculated the process is computationally easy. However, gradient descent can be slow close to the minimum as its asymptotic rate of convergence is inferior to many other methods (Seiler and Seiler 1989; Strutz 2010). Moreover, for non-differentiable functions, gradient descent is ill-defined. Interestingly, since the gradient descent

[1]This is not true in all cases, for example if there is a plateau between the current approximate value and the local minimum.

method only requires a differentiable function to calculate the gradient, if the gradient can be procured by another method the function would not be required. Methods such as Temporal Difference, Q-Learning, and SARSA use the concept of gradient descent in their update rules.

2.1.5 Reinforcement Learning

Sutton (1988) eloquently defines Reinforcement Learning (RL) capturing the essence of the subject as:

> ... learning what to do–how to map situations to actions–so as to maximise a numerical reward signal. The learner is not told which actions to take... but instead must discover which actions yield the most reward by trying them. (Sutton 1988)

When RL is framed within the Agent Paradigm it can also be defined as:

> An RL agent ... is given a reward from the environment based on the state it was in, the action performed, and the proceeding state. The aim of the agent is to maximise its long-term reward over time. (Sutton and Barto 1998)

Strictly speaking this is a definition of a single, on-line RL agent, since agents can use RL to learn in an off-line manner without being able to act upon the environment but still able to observe it (i.e., watching another agent acting in an environment). Also agents can learn in a multi-agent setting in which the other agents can affect the reward of the individual agent or the agents can receive a joint reward. This adds further challenges such as specifying a good goal, the non-stationarity of the learning, and coordination between the agents. However, multi-agent systems are outside the scope of this thesis and shall not be discussed further.

RL is based on Trial and Error learning. Trial and Error Learning began in psychology with studying animals and humans alike. The animal would be observed repeatedly trialling new ways to perform a task and learning from the mistakes made. In particular the concept of how reinforcement is used in teaching an animal to perform some task. This can be seen clearly in the way one would go about training a domestic animal such as a dog; when it correctly performs the task it is given a treat corresponding to a positive reward. Vice versa, it may be given a punishment (i.e., a negative reward) if it deliberately disobeys. Possibly the most clear definition was first published by Thorndike (1911). In his paper he defines the *Law of Effect*; the effect of repeatedly reinforcing an action in a particular scenario. He states that there are two key parts of this type of learning: *Selectional learning* that tries and compares alternatives and *Associative learning* that remembers what has been tried and the situations they were tried in for future reference. It is from these two types of learning that RL's exploration policies and value functions have been derived from respectively.

Turing (1950) was the first person to speculate whether computers could learn. Minsky (1954) and Farley and Clark (1954) began discussions about computational models of RL thus bringing the idea of trial and error learning to computers; both of these initial works used the concept of

neural-networks (a network of simulated neurons aimed to replicate the way a brain computes). It was Minsky (1961) who laid out a plan of action in discussing "Steps toward Artificial Intelligence" noting such key issues as the *credit assignment problem*; how does one distribute the credit for success among the decisions made? Partly due to a confusion between Supervised Learning (SL) and RL, RL would have to wait a couple of decades until the 1980s before dedicated research restarted.

There were, of course, exceptions in this confusion such as Andreae (1963) who developed a system called STeLLA which learned by trial and error with its environment; STeLLA also introduced the idea of an internal monologue. Klopf (1972, 1975, 1982) was the individual who was most responsible for the revival of trial and error concept of learning. He noted that the concept of an agent learning from its environment, the hedonistic aspects of learning, were being lost, since the focus of other researchers was on SL, and sought to correct this. He, amongst others, was key to the work later done by the likes of Sutton and Barto. Lastly it would be Sutton and Barto who set up the playing field and are seen as founding fathers of modern RL.

Temporal Difference (TD) is a key technique in RL. TD combines Monte Carlo ideas with dynamic programming (Witten 1977; Sutton 978a,b,c). Monte Carlo problems constitute a class of algorithms that use randomised sampling to predict an answer, the larger the number of samples the more accurate the result. However, Monte Carlo methods must wait for an episode to conclude before it updates its knowledge; in contrast TD methods only need to wait a single time step. TD methods achieve this by learning an estimated value from an estimated value, this is called *bootstrapping*. Since TD methods bootstrap they do not require a model of the environment, reward function, or transition function. This sets TD, and consequentially RL, apart from DP methods.

In summery, RL is a type of machine learning. RL is based conceptually on using the agent paradigm. RL is a method where an agent learns by receiving rewards or punishments through continual interactions with the environment (Sutton and Barto 1998). The agent receives a numeric feedback relative to its actions and in time learns how to optimise its action choices. Typically reinforcement learning uses a Markov Decision Process as a mathematical model (Puterman 1994).

2.1.6 Algorithms

As already stated, when the environment dynamics are not available, as with most real problem domains, dynamic programming cannot be used. However, the concept of an iterative approach remains the backbone of the majority of reinforcement learning algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states, $V(s)$, or state-action pairs, $Q(s, a)$. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value.

RL algorithms come in various different forms. They can be: 1) on-/off-policy meaning

they either use their exploration policy to estimate future reward or another policy; 2) on-/off-line meaning they either they never stop learning or stop learning after an initial training phase; and, 3) modelled/unmodelled meaning they either learn the transition probabilities directly or indirectly. Two key unmodelled, on-line RL algorithms are Q-Learning, an off-policy algorithm and SARSA, an on-policy algorithm. These two algorithms are fundamental cornerstones to many of the state-of-the-art RL algorithms. For comparison we will also present a modelled RL algorithm called R-MAX.

Q-Learning

The Q-Learning algorithm is a temporal-difference based reinforcement learning algorithm (Watkins 1989). After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \arg\max_{a'} Q(s', a') - Q(s, a)) \tag{2.2}$$

where α is the rate of learning and γ is the discount factor. It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and if the action a' pertaining to the currently held greatest value in the new state s' was chosen. This means that Q-Learning is an off-policy learning algorithm where the greedy policy is not used.

Q-Learning is guaranteed to converge to the true Q-values provided the exploration method visits every state-action pair an infinite number of times, that the learning rate is $0 < \alpha \leq 1$, and the discount factor is $0 \leq \gamma < 1$. Practically in simple, episodic games convergence can happen within a few hundred steps. There are many different types of exploration strategies, which many extensions of Q-learning specify, but most common are ε-greedy and Softmax. ε-greedy denotes that with ε probability you randomly choose an action otherwise you follow the greedy, the highest valued, action. Softmax denotes that every action is given a probability; the probability of each action is proportional to its value and the current temperature τ. When the temperature is high, $\tau \rightarrow 1$, the probabilities are more uniform, when the temperature is low, $\tau \rightarrow 0$, the probabilities favour high valued actions. In both methods ε and τ can be decreased over time as to settle on a fixed, greedy policy.

The Q-learning algorithm does not define the way the state space should be explored or how to choose an action; rather it is concerned with creating an accurate table of state-action pairs with an associated value, a Q-table where state-action pairs have Q-values.

An example implementation of Q-Learning in pseudo code can be seen in Algorithm 2. In this example the agent is learning in an episodic environment where the agent is repeatedly placed in the initial state and explores the environment through transitions based on the actions chosen by the exploration policy π_e.

Algorithm 2 Q-Learning($S, A, \pi_e, \alpha, \gamma$)

Initialise $Q(s,a), \forall s \in S, a \in A$ arbitrarily
repeat
  Initialise s
  repeat
    $a \leftarrow \pi_e(s)$
    Take action a, observe r and s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \arg\max_{a'} Q(s',a') - Q(s,a))$
    $s \leftarrow s'$
  until s is terminal **or** time expires
until time expires

SARSA

The SARSA, or State-Action-Reward-State Action algorithm is also a temporal-difference method which is based on Q-learning (Rummery and Niranjan 1994; Sutton and Barto 1998). The update rule is altered as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{2.3}$$

where $max_{a'} Q(s',a')$ is replaced with just $Q(s',a')$; instead of using the action pertaining to the currently held greatest value, the next action based on the exploration policy used. This means SARSA is an on-policy learning algorithm. Empirical results often show SARSA converging on its final policy faster than Q-Learning and as such is often used in lieu of Q-Learning.

An example implementation of SARSA in pseudo code can be seen in Algorithm 3. In this example the agent is learning in an episodic environment where the agent is repeatedly placed in the initial state and explores the environment through transitions based on the actions chosen by the exploration policy π_e.

Algorithm 3 SARSA($S, A, \pi_e, \alpha, \gamma$)

Initialise $Q(s,a), \forall s \in S, a \in A$ arbitrarily
repeat
  Initialise s
  $a \leftarrow \pi_e(s)$
  repeat
    Take action a, observe r and s'
    $a' \leftarrow \pi_e(s')$
    $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal **or** time expires
until time expires

R-MAX

R-MAX is a very simple model-based RL algorithm which can attain near-optimal average reward in polynomial time. It achieves this by maintaining a complete, but possibly inaccurate model of its environment and acts on the optimal policy derived from this model. The model is initialised optimistically assuming all actions in all states yield the maximum reward. This model is adjusted throughout learning based on the observations of the agent (Brafman and Tennenholtz 2003).

Strehl et al. (2009) provide an efficient implementation that can be seen in Algorithm 4. In their implementation ε_1 is a real-valued parameter that specifies the desired closeness to optimality of the policy produced by value iteration, m is an integer parameter that specifies the number of samples to use in the model (the first m samples), and U is a function that returns the upper bound for a Q-value, $Q^*(s,a) \leq U(s,a)$. Furthermore, $n(s,a)$ and $n(s,a,s')$ are the number of time steps the agent has taken a in s and the number of times that state-action pair transitioned to state s' respectively; $r(s,a)$ is the sum of rewards given for performing action a in state s; $\bar{s}$ and $\bar{a}$ are strictly not equal to s and a respectively; $\hat{R}(s,a)$ is the mean reward for perform action a in state s; and, $\hat{T}(s'|s,a)$ is the empirical frequency of arriving in state s' when action a is performed in state s.

Algorithm 4 R-MAX($S, A, \gamma, m, \varepsilon_1, U$)

for all $(s,a) \in S, A$ **do**
  $Q(s,a) \leftarrow U(s,a)$
  $r(s,a) \leftarrow 0$
  $n(s,a) \leftarrow 0$
  for all $s' \in S$ **do**
    $n(s,a,s') \leftarrow 0$
for $t = 1, 2, 3, \ldots$ **do**
  Let s denote the state at time t
  $a \leftarrow \arg\max_{a' \in A} Q(s,a')$
  Take action a, observe r and s'
  if $n(s,a) < m$ **then**
    $n(s,a) \leftarrow n(s,a) + 1$
    $r(s,a) \leftarrow r(s,a) + r$
    $n(s,a,s') \leftarrow n(s,a,s') + 1$
    if $n(s,a) = m$ **then**
      for $i = 1, 2, 3, \ldots, \lceil \frac{\ln(1/(\varepsilon_1(1-\gamma)))}{1-\gamma} \rceil$ **do**
        for all $(\bar{s}, \bar{a})$ **do**
          if $n(\bar{s}, \bar{a}) \geq m$ **then**
            $Q(\bar{s}, \bar{a}) \leftarrow \hat{R}(\bar{s}, \bar{a}) + \gamma \sum_{s'} \hat{T}(s'|\bar{s}, \bar{a}) max_{a'} Q(s', a')$

2.2 Generalisation

All the algorithms that have been introduced so far assume that every state-action pair has a uniquely represented value estimate in a value table. This limits such algorithms to environments with few states and actions for a number of reasons: 1) the amount of memory required to store the value table; 2) the amount of time needed to fill the value table; and, 3) the need to visit a state multiple times. Although modern computers have access to extremely large amounts of memory, any environment with at least one continuous state feature has an infinite number of states to store values for. Practically speaking, the number of states will be limited by the number representation in the programming language used. These environments then require an infinite amount of time to learn even a subset of all possible states. Furthermore, learning algorithms that optimistically initialise values require at least $|A|$ visits to any state to learn, performing each action, a, at least once. Learning algorithms that pessimistically initialise values require, in the best case, 1 visit to a state assuming the optimal action is performed.

This is a severe problem for RL agents but it can be resolved with generalisation (Sutton 1988). Generalisation assumes that if in multiple states an agent's policy will be the same, such states can be grouped together. This allows for knowledge learnt in one state to be applied to all states in that group. The problem of grouping states effectively is still open. Unfortunately generalisation does not solve the problem perfectly. Boyan and Moore (1995) state that the use of function approximation in a continuous search space does not guarantee convergence to the optimal policy as the requirement of visiting all state-action pairs infinitely often cannot be realised. Moreover, methods of generalisation often assume that neighbouring states in a state space have similar values of the value function which may not always be the case. Furthermore, environments that require precision in particular areas of the state space and allow for small margins of error run the risk of high values being lost to the average of low values surrounding them. However, there still can be convergence guarantees in some restricted cases and there may be some particular formulation of function approximation that can be proven to converge in the general case, for example see Melo et al. (2008).

Value Function approximation is one solution that has been proposed to handle this problem; value function approximation "*simply means using any sort of representation for the function other than a table*" (Russell and Norvig 2003:pp777). It is called approximation because these other forms of representation may not encompass the true value function or Q-function. Approximation allows for massive reduction in the state space but the true principle behind it is the ability to generalise knowledge, "*the compression achieved by a function approximator allows the learning agent to generalise from states it has visited to states it has not visited*" (Russell and Norvig 2003:pp777). Since the state space is reduced the learning rate increases, albeit the quality of their solution can decrease.

2.2.1 Linear Methods

Gradient Descent methods are well suited to RL and as such among the most widely used of all function approximation methods. Gradient Descent requires that the function used to approximate value of the state, or state-action pair, is differentiable and assumes that the local minimum is the best approximation of that value. Every time the approximate value is to be updated the gradient is calculated at the current approximated value and one proportional step is taken to the negative of the gradient. One particular special case of gradient descent is when the approximate function is linear in regard to the parameter vector, i.e., for every state there are corresponding parameters in the approximation function for each state feature. Function approximation methods that use this linearity are known as linear methods.

2.2.2 Coarse Coding

Coarse coding is a linear method which groups states into value ranges of each feature; for example in a two-dimensional, continuous state set each grouping could be seen as a circle, possibly overlapping, covering the entire state space. In this way a state could fall into the realm of more than one grouping. For example, see Figure 2.2 where state X falls into the realm of 3 groupings. Each grouping the state lies within would be activated by that state. The agent would then learn about what to do for the different combinations of activated groupings. In general these groupings do not need to be uniform in size, shape, or placement.

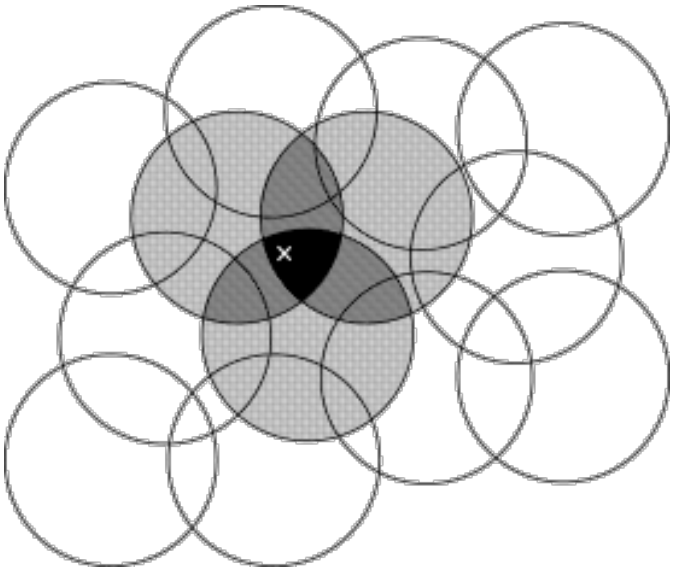


Figure 2.2: A coarse coding over a two-dimensional, continuous state space where state X activates 3 groupings (Sutton 1988).

2.2.2.1 Tile Coding

Tile Coding (TC) is a form of coarse coding. In TC one or more features of the state space is exhaustively partitioned, called a *tiling*. Each partition of the tiling is called a *tile*. Only one tile from each tiling can be activated by any one state. Tilings can be regular, irregular, log stripes, diagonal stripes, etc., and more than one tiling can be overlaid onto the state space (Albus 1981). Typically each tiling contains the same number of tiles and each tiling is offset as to overlap but not align.

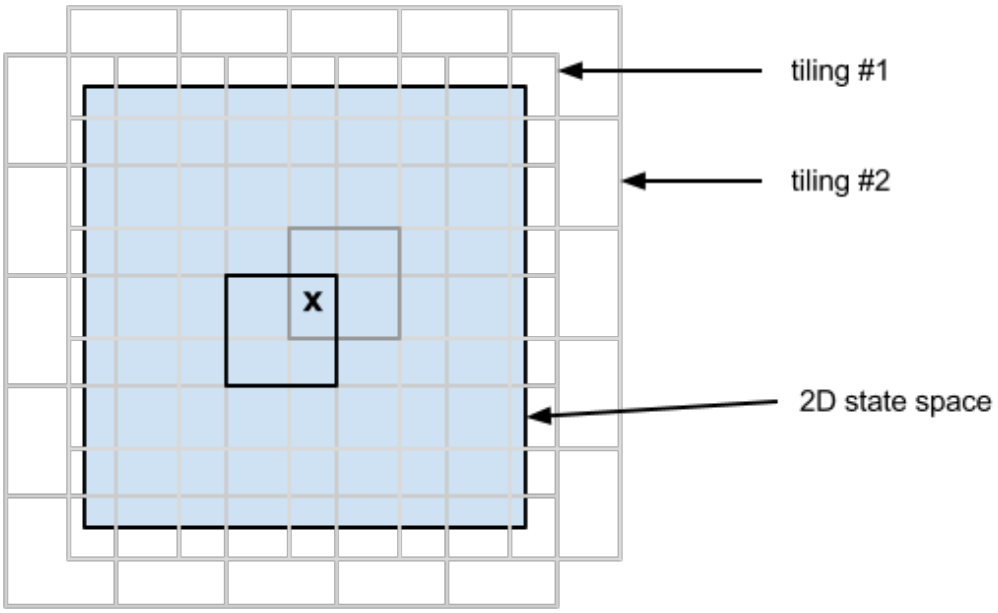


Figure 2.3: Multiple, overlapping grid tilings

Figure 2.3 is an example of two tilings covering a 2-dimensional state space. Each dimension corresponds to a numerical feature and each point in the space corresponds to a state. As can be seen the two tiles activated in each tiling have been highlighted.

Formally, TC defines a weights vector $\vec{w}$, with n components where n is the total number of tiles, and a function $TC(\vec{\phi_s}) \rightarrow \vec{\theta_s}$, which takes a vector of the state features $\vec{\phi_s}$ and returns a binary vector $\vec{\theta_s}$ with n components. Each component in the returned binary vector corresponds to whether the i^{th} tile was activated, i.e., $\theta_s(i) = \{0, 1\}$. This allows us to define the value function of a state as:

$$V(s) = \vec{\theta_s}\vec{w} = \sum_{i=1}^{n} \theta_s(i)w(i)$$

Since in RL the transition function is assumed to be not known the Q-value is more useful to learn. To allow for this we create a new feature in the state space to account for the action taken.

This then requires $|A| \cdot n$ tiles instead of n tiles.

Another, and arguably simpler, way of approaching the formulation of TC is to consider it as a discretisation of the state space. This is equivalent since there is always exactly one tile activated in each tiling for any one particular state. Furthermore, for any two equal states the TC function should activate the same set of tiles. Therefore, each tiling can be considered its on discretisation of the state-space.

In the discretisation approach a state s maps to m activated tiles $TC(s) \rightarrow (t_{s_1}, t_{s_2}, \ldots, t_{s_m})$ where t_i is the tile activated in the i^{th} tiling. The Q-value of s then becomes the sum of Q-values of the $(t_{s_1}, t_{s_2}, \ldots, t_{s_m})$:

$$Q(s,a) = Q(t_s, a) = \sum_{i=1}^{m} Q(t_{s_i}, a)$$

The Q-value for each activated tile would then need to be updated. For example, the update rule for Q-Learning would be altered as follows:

$$Q(t_{s_i}, a) \leftarrow Q(t_{s_i}, a) + \frac{\alpha}{m}(r + \gamma \arg\max_{a'} Q(s', a') - Q(s, a))$$

where t_{s_i} is the i^{th} activated tile in the tile coding. TC simplifies the learning for an agent by allowing the agent to learn the tile-space rather than the state-space, which is usually much smaller in cardinality.

A basic formulation of the TC algorithm can be seen in Algorithm 5. The algorithm starts by initialising the tilings and the tiles within. Then, while there is still time to learn, it repeatedly resets the environment and begins an episode. During an episode the agent perceives the current state, selects and performs the action according to an exploration/exploitation policy, receives a reward, and for each tiling updates the activated tile. S is the set states in the environment, A is the set of actions, T is the tile coding function, R is the reward function, m is the number of tilings to use, n is the total number of tiles to use, π is the action selection policy, α is the learning rate, and γ is the future discount factor.

Algorithm 5 Tile-Coding($S, A, T, R, m, n, \alpha, \gamma, \pi$)

Initialise $Q(t, a), \forall t \in TC, a \in A$ arbitrarily
repeat
  Initialise s
  repeat
    $a \leftarrow \pi(s)$
    Take action a, observe r and s'
    $Q' \leftarrow \sum_{i=0}^{m} \arg\max_{a' \in Q} Q(t_{s'_i}, a')$
    for $i \leftarrow 1$ **to** m **do**
      $\Delta \leftarrow r + \gamma Q' - Q(t_{s_i}, a)$
      $Q(t_{s_i}, a) \leftarrow Q(t_{s_i}, a) + \frac{\alpha}{m}\Delta$
  until s is terminal **or** time expires
until time expires

Tile Coding, as any value function approximation method, impacts the agents learning. This can be positive or negative; for example, TC can reduce the time required to learn a policy, or in the worst case TC can hinder learning entirely.

2.2.3 Algorithms

There are many other methods for function approximation beyond tile coding. However, tile coding is "*particularly well suited for use on sequential digital computers and efficient on-line learning*" (Sutton 1988). Tile coding is conceptually easy to understand, in part thanks to the ease of which environments with two state features can be visualised. The following are adaptations and improvements to the tile coding method.

Radial Basis Functions

Radial Basis Functions (RBFs) generalise coarse coding whereby instead of having binary features (either a grouping is activated or not) each grouping can be partially activated depending on the radial "distance" of the centre of the particular grouping. Specifically, an RBF feature, i, has a Gaussian (bell-shaped) response $\phi_s(i)$ dependant only on the distance between the state, s, and the feature's prototypical or centre state, c_i, and relative to the feature's width, σ_i:

$$\phi_s(i) = exp(-\frac{||s - c_i||^2}{2\sigma_i^2}) \tag{2.4}$$

The norm or distance metric can be chosen in whatever way seems most appropriate to the states and task at hand.

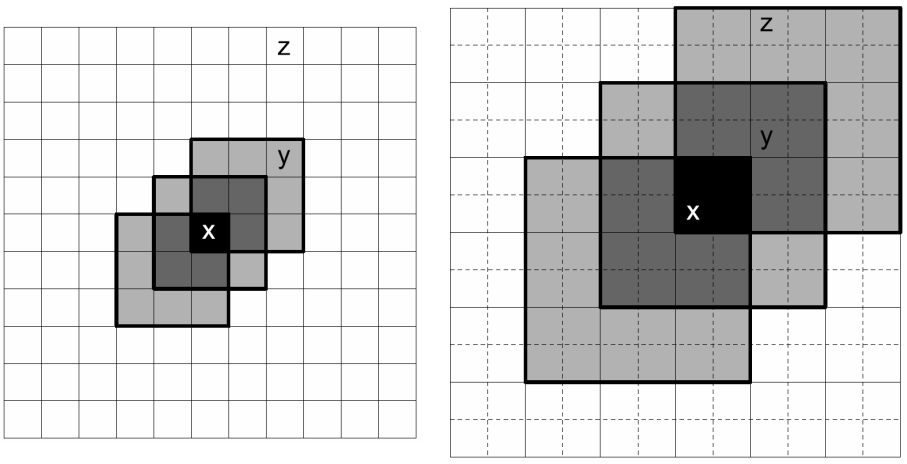
An *RBF network* is a linear function approximator using RBFs for its features. The primary advantage of RBFs over binary features is that they produce approximate functions that vary smoothly and are differentiable. In addition, some learning methods for RBF networks change the centres and widths of the features as well. Such non-linear methods may be able to fit the target function much more precisely. The downside to RBF networks, and to non-linear RBF networks especially, is greater computational complexity and, often, more manual tuning before learning is robust and efficient.

Kanerva Coding

Environments whose states have extremely high dimensionality in regard to their features do not suit techniques like coarse coding since they use all the dimensions of the state. A simple approach which handles this problem is call Kanerva Coding. Kanerva coding is unaffected by dimensionality per se; it chooses binary features that correspond to prototype states. So instead of dividing up the state space into sections it reduces the size of the state space to a few prototype states with binary features. In this way is unaffected in its complexity when dimensions are added or removed.

Mixed Resolution Tile Coding

Mixed Resolution Tile Coding is an approach devised by Grzes and Kudenko (2009). Mixed Resolution Tile Coding treats both fine and coarse tilings as parts of the same function approximator. Figure 2.4 shows an example of two such tilings.



(a) (b)

Figure 2.4: Tile coding examples with a different resolution. Three tilings with tiles of three units in 2.4a and six units in 2.4b (Grzes and Kudenko 2009)

In the algorithm two different tile codings are used, one with a fine resolution of the environment and the other with a coarse resolution. The coarse resolution is less expressive and allows for a broader generalisation earlier on in learning. The fine resolution is more expressive and allows for refinement later on in learning. The value for any state $V(s)$ then simply becomes the sum of the value in the fine and coarse tile codings. This produces a natural coexistence of the two resolutions in the one function approximator.

Grzes and Kudenko (2009) proposed a second, more sophisticated algorithm where the reward shaping is used. In this second version they use a higher and lower resolution function approximation, both resolutions are updated as to learn about the environment but the lower resolution is used as an additional reward shaping signal for the higher resolution which then the agent uses to decide upon the actions to take. They showed that their extensions can be beneficial when "1) there are many actions in each state; 2) a high resolution of the policy is required with a wide range of values of state variables ...; 3) a high level guidance can be extracted from a subset of state variables" (Grzes and Kudenko 2009). A powerful part of this algorithm is that it doesn't require any domain specific knowledge and that, although applied to SARSA tile coding in this case, the theory should be transmutable between other forms of function approximation.

Adaptive Tile Coding

Munos and Moore (2002) investigated different methods of when to increase resolution of particular areas of the state space:

- criteria based on the value function, split on the feature that would most alter the value function;
- criterion based on the policy, split on the feature that would most alter the policy;
- notion of influence, split on the feature that has the largest influence on surrounding states, influence being a measure on the extent the state contributes to its neighbours;
- variance of a Markov chain, a measure of how dissimilar the cumulative future reward obtained along all possible trajectories; and,
- a global splitting criterion, which is a combination of influence and variance.

An interesting observation made was for 2d problems this approach worked well but it couldn't outperform uniform grids on more complex problems. These methods require knowledge of the state transition function and therefore are not strictly compatible with RL but Whiteson et al. (2007) use their work to create Adaptive Tile Coding.

Adaptive Tile Coding (ATC) is an approach devised by Whiteson et al. (2007). ATC uses a single, adaptive tile coding with 1 tiling. For each tile in the tile coding ATC stores $2k$ sub-tiles weights where there are k features; for each feature the tile is split in half to create a lower and upper sub-tile. ATC initially starts with n tiles but over time splits tiles in half with respect to one of the one of the state features. ATC decides when to split by keeping track of the number of updates that have occurred since a new lowest Bellman error was recorded on a tile u; if u reaches a parameter threshold p ATC splits the tile that maximises either the value or policy criterion. The value criterion states that tiles should be split so as to minimise error in V. It achieves this by selecting the tile and feature that has the greatest absolute difference between the value of the lower and upper sub-tile. This causes ATC to devote more resolution to areas of the state space where V changes rapidly and less resolution to areas where it is relatively constant. The policy criterion states that tiles should be split so as to improve the policy π. It achieves this by incrementing a counter for each feature on a tile if, for the current state, splitting the tile along that feature would cause a change in a purely greedy action selection in that state. Then the tile and feature with the highest valued counter is selected to be split. This causes ATC to split tiles in regions where more resolution will yield a refined policy.

Algorithm 6 shows the pseudo-code for ATC. Where k is the number of state features, n is the number of tiles that the tile coding initial will contain, and p is the threshold for splitting.

Algorithm 6 Adaptive-Tile-Coding($S, A, T, R, k, \alpha, \gamma, n, p$)

$u \leftarrow 0$
Initialise one tiling with n tiles
for $i \leftarrow 1$ to n **do**
  Initialise i^{th} tile and $2k$ sub-tile weights to zero
repeat
  $s \leftarrow$ random state from S
  $\Delta V(s) \leftarrow max_a[R(s,a) + \gamma V(T(s,a))] - V(s)$
  $w \leftarrow$ weight of tile activated by s
  $w \leftarrow w + \alpha \Delta V(s)$
  for $d \leftarrow 1$ to k **do**
    $w_d \leftarrow$ weight of sub-tile w.r.t split along d activated by s
    $\Delta w_d \leftarrow max_a[R(s,a) + \gamma V(T(s,a))] - w_d$
    $w_d \leftarrow w_d + \alpha \Delta w_d$
  if $|\Delta V| <$ lowest Bellman error on tile activated by s **then**
    $u \leftarrow 0$
  else
    $u \leftarrow u + 1$
  if $u > p$ **then**
    Perform split that maximises value or policy criterion
until time expires

2.2.4 Directions of Research

All the example algorithms described in the previous section, excluding Adaptive Tile Coding, follow a very straight path from the inception of Tile Coding to the present day. There are other paths of research that have veered from the straight and narrow.

The three main paths are: 1) Transfer Learning; 2) Two Steps Learning; and, 3) Adaptive Learning. Transfer learning aims to take knowledge from one environment, usually quite similar in nature, and transfer that knowledge to another environment so as to reduce the time required to learn a good policy in the new environment. Two Steps Learning aims to break learning down into two stages: pre-learning, where the agent tries to learn some meta information about the problem; and learning-proper, where the agent uses what they learnt in the first stage to improve the speed and/or accuracy in the second stage. Finally, Adaptive Learning, which is similar to Two Steps Learning, aims to unite the two steps of learning so that they occur side-by-side, influencing each other.

Transfer learning

Transfer learning is a technique used in many different machine learning disciplines. However, we will only consider transfer learning research that has been specifically tied to Tile Coding approximation. Tile Coding is a helpful tool for Transfer Learning for a number of reasons: 1) Tile Coding can be set up and used without much in-depth knowledge and understanding; 2) Tile Coding representations can often be easily visualised (especially in low dimension envir-

onments); and, 3) Mapping functions between Tile Coding representations of environments are easy to define. Transfer learning is a good example of the usage of Tile Coding outside of its direct line of research. Transfer learning, as stated above, aims to simply transfer learnt knowledge between environments. However, this is a non-trivial problem to solve. In general, it is not clear what aspects of a solution in one environment will be beneficial in another. This subsection is outside the scope of this thesis and will not be discussed further. If the reader is interested, a stepping stone into this subject would be (Taylor et al. 2005) and an up-to-date paper (Brys et al. 2015a).

Two Steps Reinforcement Learning

Two Steps RL is based on the notion that there are two distinct parts to adaptive approximation: 1) deciding on the tile coding; and 2) learning the optimal policy using that tile coding. Moreover, these two steps *must* happen sequentially. By contrast Adaptive Tile Coding is where there are still the two distinct parts but these can occur simultaneously. Environments with very large continuous state spaces must be generalised for reinforcement learning to be applied, as we explained at the beginning of this section.

Fernández and Borrajo (2008) introduce this approach whereby the agent undergoes two stages of learning: in the first, the agent acts as a supervised function approximator that outputs a state space discretisation of the environment; in the second, a tabular value function is learnt of the discretised space produced by the first stage.

Jin et al. (2009) continues this two step line of research but focuses on discovering critical states within the state space. They define Critical States to be states observed as having a high probability of being passed through (according to the learnt experience). These critical states are then used to partition the larger state space. Further to this Jin et al. (2009) prove that the optimal policy found in the partitioned state spaces is the equivalent policy to the optimal policy found in the original state space.

Adaptive Learning

Multi-Resolution Exploration is based around the fundamental idea that the essence of exploration is acting to try and decrease an agent's uncertainty about a given problem.

Kearns and Singh (2002) introduce an algorithm which aimed to have near-optimal reinforcement learning be possible in polynomial time. They are able to prove that their algorithm has polynomial bounds on the resources required to achieve near-optimal return in general MDPs. Explicit Explore or Exploit (E^3) uses either "balanced wandering" on unknown states or plans a *T-step* policy guaranteed to either exploit or explore. Kakade et al. (2003) are able to further this research by presenting "metric-E^3". The near-optimal policy found by metric-E^3 is dependent on the *covering* number of the state space rather than the size of the state space directly. This allows the user of the algorithm to decide the trade off between the accuracy of the policy and the speed at which the policy is learnt. Nouri and Littman (2009a) bring the idea of using vari-

able resolution to identify regions of the state space that would benefit from additional samples. Rather than assume the entire state space requires a uniform distribution of "knowness" (unlike Kakade et al. (2003) and Kearns and Singh (2002)) they introduce the Multi Resolution Exploration (MRE) algorithm that uses this assumption to great effect. Nouri and Littman (2010) extends their previous work by using it in conjunction with a dimension-reduction technique that provides a statistically more efficient way of learning a transition function in high dimensional environments. The use of their knowness function dramatically decreases the sample complexity required for stable learning. MRE is an interesting alternative approach to adaptive approximate learning. Instead of Tile Coding being the base approximation, MRE uses samples. However, like ATC, it states that the entire state space should not be covered by a single, uniform approximation. This reinforces the non-uniform approximation concept by showcasing it in a field closely related to Tile Coding approximation.

Adaptive-Resolution Reinforcement Learning is based on the idea that an agent is able to learn how to discretise a state space whilst simultaneously learning about that very same state space. As previously discussed, Whiteson et al. (2007) present their Adaptive Tile Coding algorithm which uses one of two possible criterion to decided when an area of the state space requires higher resolution.

Bernstein and Shimkin (2010) propose a model-based learning algorithm, which they call Adaptive-resolution Reinforcement Learning (ARL), and an incremental version called Incremental ARL (IARL). ARL uses adaptive approximation of the optimal value function using *kernel-based* averaging. IARL is a more practical version of ARL which has reduced computation complexity at each stage. Jung and Stone (2010) present a model-based on-line reinforcement learning algorithm similar to RMAX for continuous domains by combining GP-based (*Gaussian Process regression*) model learning and value iteration on a grid. This allows their approach to separate the problem function approximation in the model-learner from the problem function approximation/interpolation in the planner. They claim that if the transition function is easier to learn, then large savings in sample-complexity can be gained. However, they concede that the fundamental limitation of their approach is that it relies on solving Bellman equations globally over the state space. ARL, IARL, and the algorithm devised by Jung and Stone (2010) are explicitly model-based approaches to learning rather than keeping and maintaining a value table at every time step. This thesis is interested in RL approaches that are not model-based. For this reason these algorithms will be not considered as possible baselines.

Zhu et al. (2014) propose a new on-line RL algorithm MSEC (Multi-Samples in Each Cell) which aims to pursue a near-optimal policy in continuous state environments. The approach combines state aggregation technique and the efficient exploration principle. Specifically, this is achieved by applying a grid over the continuous state space and partitioning it into different cells. They concede that in order to achieve highly near-optimal policies the grid must have high resolution. Their method does not allow for varied resolutions throughout the continuous-state

space.

Another adaptive resolution learning algorithm comes from Taylor et al. (2011). Their work introduced HOLLER (Hinge loss Online Logdet LEarner for Relative distances), a distance metric learning algorithm, and combines it with an existing instance-based RL algorithm. They state strength of HOLLER is the small amount of data required to learn an appropriate state representation and thus on-line learning can still be achieved.

2.3 Scope and Open Problems

As the literature has been reviewed questions have been raised, some of which are answered by later works, others have been partially answered, and still more have not been answered. The state-of-the-art algorithms have progressed far and recent works such as Whiteson et al. (2007) work on Adaptive Tile Coding and Grzes and Kudenko (2009) work on Mixed Resolution Tile Coding. These results are promising and encouraging to the potential of Tile Coding. In the following subsections different open topics in RL are considered.

2.3.1 Credit Assignment Problem

The credit assignment problem, first addressed by Sutton (1984), is an ongoing open problem. It is the problem of deciding which actions that lead up to receiving some reward were responsible for obtaining said reward, and if so to what extent. For example, there may have been a series of actions which either took the agent further from the reward or left the agent stationary in terms of moving toward the reward. This issue is further complicated in MAS where coordination of actions may be required to achieve the reward, so that even if one agent performs the exact same set of actions in the same order different rewards may be achieved.

2.3.2 Scalability and Dimensionality

Most literature in RL provides empirical evidence of their algorithm in environments with low numbers of actions and relatively short paths from the initial state to the goal. Whilst these test bed environments are practical and allow researchers to compare performance there are no guarantees that the performance benefits shown in one environment will scale up to a more complex problem. Furthermore, when environments with high numbers of state features, large dimensionality, are presented techniques that previously performed well suffer computationally if the algorithm took into account state features. Still the problem of scalability is very much open.

2.3.3 Knowledge and Heuristics

One of the strengths of RL is that it does not need a model of the environment, nor does it need to know the state transition function since it learns through exploratory interaction with the environment receiving immediate rewards. This strength can be a double-edged sword in terms of practical use of RL algorithms in real world environments since the state space that the agent must learn is so vast, infinite in most cases, that without heuristic knowledge to guide searching

for good policies convergence, even if theoretically guaranteed, practically never appear. So the overarching question is then how to supply domain knowledge to an RL agent without losing the natural robustness of RL? This question is further complicated when you consider the possibility that the knowledge provided may not be entirely accurate or, because of the dynamic nature of the environment, is quickly outdated. So the question here is (how) can an agent overcome bad/incorrect/expired knowledge? Another issue is can such knowledge cause an agent to actually draw an agent away from an optimal policy, or alter the system so that the optimal changes? One answer to this issue is potential-based reward shaping which guarantees that the Nash equilibria won't be altered if used properly but it can alter, for better or worse, which equilibrium is arrived at.

2.3.4 Design

It is plain to see that the same thing can be designed in countless different ways, some terrible, others brilliant, and many very similar having slight improvements in particular areas. Take cars for example; the basic concept of a powered vehicle which can transport persons and goods has led to thousands of different realisations. Due to the commercial nature of auto-mobile industry most, if not all, the terrible designs are never seen. They also share their components for reason such as they are known to be good. In a similar vein environments, and RL algorithms, share components and the terrible ones are considered and ruled out. Since there is such thing as good and bad design, subjective as it is, can the design of an environment affect the ease of convergence? Or a better question would be, is there a way of general means constructing environments such that optimal performance is a trivial problem to solve? And is this question any easier than designing RL algorithms?

2.3.5 Function Approximation

The goal of Function Approximation (FA) is to reduce a problem so that it is possible to solve, whilst maintaining enough of the original problem so that the solution is viable. In general the larger the reduction, the greater the loss of performance. The first open problem in FA is how best to approximate state/action spaces. There are many parts to the open problem such as:

1) What method should be used to group states/actions together?

1) Should states and actions be combined in the approximation?

1) What method should be used to apply information learnt in one state to another?

1) How should it be determined which states should share information and which states should not?

Furthermore, there is the open problem of deciding what level of approximation should be used. Is the level of approximation required dependent on the specific environment, the environment class, or is there a general level of approximation that applies to all environments?

There are, of course, already methods of approximation, such as Tile Coding, that work and allow RL algorithms to solve continuous problems. Tile coding reduces the state/action space by exhaustively grouping continuous state spaces together by a finite number of partitions. However a lack of theoretical understanding leads to the open problem of how TC impacts learning algorithms such as SARSA or Q-Learning in the general case. Further problems arise in this methodology: 1. at boundary cases where a true generalisation of the state/action space may be split across one or more chosen groupings, and 2. when more than one conflicting state/action is within the same grouping.

The approximation can be progressively refined, increasing the resolution, as and when needed, which can solve problems more quickly. This is known as "*the general towards specific approach: an initial coarse grid is successively refined in areas of the state space by using a splitting process, until some desired approximation is reached*" (Munos and Moore 2002). This approach requires the agent to learn meta information about the problem as well as the solution to the problem. This can happen in distinct steps, such as Two Step RL, or side-by-side, such as Adaptive Learning. The learnt meta information is used to adapt the approximation of the problem. There are various types of meta information that can be learnt, for example: 1) Jin et al. (2009), in their Two Steps RL algorithm, use the concept of Critical States; 2) Nouri and Littman (2009a), in their Multi Resolution Exploration algorithm, use the concept of Knowness; 3) Whiteson et al. (2007), in their Adaptive Tile Coding algorithm, use the change in value function or policy. Deciding what type of meta information to learn about the environment is an open problem. The main open problem of Two Steps RL is deciding when to change from the first to the second step of learning. Changing too soon can result in an approximation that does not represent the problem well; changing too late results in wasted time. One possible solution to this open problem is to adapt the approximation whilst learning the solution. However, a potential flaw in adaptive learning is that learning whilst the approximation is still coarse could act like a bad initialisation and slow down, but not entirely hinder, learning. Other general open problems of these types of FA include:

- Understanding how the shape and size of tiles impacts learning;
- Understanding how altering the Tile Coding impacts learning;
- Can groupings be joined/moved during learning to better match the environment;
- Can an automated process devise the "right" tile coding whilst efficiently solving the problem.

This property of function approximation, the coarser the resolution of the approximation the faster the learning rate, was investigated in another way by Grzes and Kudenko (2009). Grzes and Kudenko (2009) proposed learning with mixed resolution function approximation in two

different ways: firstly where two resolutions learn side-by-side, and secondly a more sophisticated algorithm where the reward shaping is used. This paper raises the question of how many levels of reward shaping can be used and would such a hierarchy prove beneficial? Should there be a weighting applied to the differing resolutions so that as time progresses one can be more impactful than another?

2.3.6 Scope of the thesis

The scope of this thesis is to study Function Approximation; specifically to: 1) study the impacts that Tile Coding has on Reinforcement Learning algorithms; 2) apply the findings to create a new, state-of-the-art adaptive Tile Coding algorithm.

Currently, there is a lack of theoretical understanding of the impacts that Tile Coding has on widely used RL algorithms such as SARSA and Q-Learning. Without proper understanding, algorithms based on Tile Coding may suffer unknowingly or may be inherently flawed. This provides motivation for conducting research into these impacts.

From the grounding of theoretical understanding this thesis will then devise a tile placement algorithm that will aim to create tiles that are sized sufficiently large to be able to generalise but sufficiently small, where appropriate, to learn precisely. Furthermore, this thesis will automate the tile placement algorithm and compare it to other, similar algorithms.

CHAPTER 3

Theoretical Properties of Transition Cycles

This chapter introduces new theoretical properties of SARSA, specifically how transition cycles in the MDP that yield the same transition reward can cause an agent to temporarily diverge from learning the optimal policy on the states-action pairs that comprise that transition cycle. These theoretical properties are then extended to include Q-Learning by altering the assumptions, lemmas, and theorems to require that each state-action pair in the transition cycle is maximal for that state. The theory will be used in Chapter 4 to show this effect is especially prevalent in Tile Coding where states are grouped together into tiles. Furthermore, these theoretical properties will be the basis of new heuristics for tile placement in a Tile Coding. Finally, in Chapter 5 an implementation of a novel algorithm that uses the new heuristics for tile placement is derived, discussed, and analysed.

This chapter starts by motivating the need for theoretical understanding. It then proceeds to present and discuss the assumptions of the new theoretical properties of SARSA. These are followed by the Lemmas and proofs used by the theoretical properties. Next this chapter presents the theoretical properties as theorems. Afterwards, the theory is extended to include Q-Learning. Finally, the theory is empirically demonstrated and conclusions are drawn.

3.1 Motivation

Techniques like SARSA and Q-Learning are fundamentally altered by function approximation methods. The known theoretical guarantees of Q-Learning of policy convergence, for example, have not been extended to include the use of TC. Instead we are left with a method that works well in practice but has little theoretical grounding. Furthermore, there are nuances of SARSA and

Q-Learning that are not fully understood and therefore their impact on TC is also not understood.

Understanding the theoretical properties of techniques helps users to exploit the useful aspects whilst avoiding the detrimental. For this reason theory should be established and ratified with empirical evidence.

3.2 Assumptions

The theoretical properties of SARSA presented in this chapter have been made with the following assumptions:

Assumption 1 (Transition Cycle)**.** *There exists a state s_0, a sequence of states s_1 to s_n, and a sequence of actions a_1 to a_n where $s_0 = s_n$ and $n \geq 1$ that starting in s_0 executing a_1 to a_n in order with probability > 0 will result in the states s_1 to s_n being visited in order. This is called a Transition Cycle.*

Assumption 2 (Identical Rewards)**.** *The reward yielded for each transition in the sequence of actions, a_1 to a_n, and states, s_0 to s_n, are the same, $R(s_i, a_{i+1}, s_{i+1}) = r$, for $0 \leq i < n$.*

Assumption 3 (Approximation)**.** *Computers approximate number values to varying degrees of accuracy. There is a variable value Δ such that if two numbers $|x_i - x_j| < \Delta$ then these numbers are indistinguishable.*

Assumption 1 stipulates that the MDP must contain at least one transition cycle. This assumption does not mean that the probability of the transitions occurring has to be one, rather it must be greater than zero, $\sum_{i=1}^{n} T(s_i, a_i, s'_{i+1} > 0$. The higher the probability of each transition the higher the probability the agent will be able to complete the transition cycle. Many of the commonly used RL environments for empirical analysis contain transition cycles; for example: Mountain Car, Puddle World, Car Parking, Cliff World, Windy World, Cart Pole, Acrobot, Predator-Prey, Mazes, and Flag Collection. This assumption is required for the theoretical properties and cannot be relaxed.

Assumption 2 stipulates that the reward for each transition in the cycle must yield the same reward r. This does not mean that the probability of r being yielded for every transition in the cycle must be one. However, the effects of our theory will only be seen during instances of learning when r is the only reward yielded. In general the probability of r being yielded for any given transition within the cycle must be greater than zero and less than or equal to one, $0 < P(r|R(s_i, a_i, s_{i+1})) \leq 1, for 0 \leq i < n$. This assumption is required for the theoretical properties and cannot be relaxed.

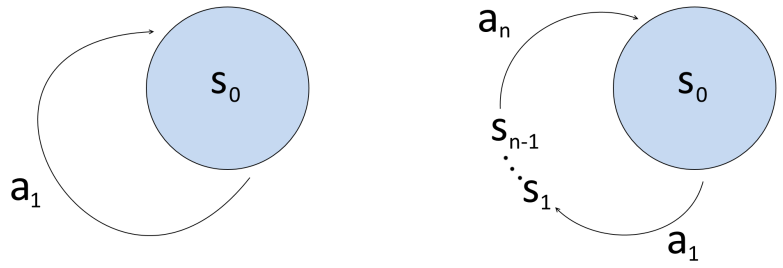
We define an Identical Reward Transition Cycle (IRTC) as a transition cycle within an MDP (Assumption 1) where the probability of every transition reward yielding an identical value r is > 0 (Assumption 2). Specifically, when discussing IRTCs we refer to the case where r is always given as the transition reward.

Finally, Assumption 3 stipulates that there is value Δ such that if the difference between the two distinct numbers is less than Δ the numbers are indistinguishable to the computer storing the numbers. The value of Δ is dependent on the storage method and the two number being compared. This assumption allows for meaningful understanding of the factors that impact the rate of convergence. Furthermore, it allows us to prove under specific conditions that in the scenario where an agent consecutively loops through an IRTC, it will always diverge from the optimal policy. This assumption is required for theoretical understanding of the rate at which Q-values within an IRTC convergence toward their common limit. However, it is not required in the proofs of the theorems defined later in this chapter.

To conclude, the first two assumptions are required for the theoretical properties to be presented as they are the foundation on which the theory is built. The final assumption allows for further understanding of impacts of the theory.

3.3 Theoretical Properties of SARSA

The theory to be presented assumes that there is at least one IRTC in the MDP. Figure 3.1 shows a couple of examples of potential IRTCs in a sub-MDP. Figure 3.1a shows the most simple possible IRTC that contains just one state and one action. Figure 3.1b shows a general case where there are n states and n actions. There can be, of course, other states and actions in the whole MDP.



(a) IRTC with 1 state and 1 action (b) IRTC with n states and n actions

Figure 3.1: Examples of IRTCs in a sub-MDP

For the remainder of the lemmas and proofs we will consider the sub-MDP consisting of only the IRTC. This will allow us to consider theoretically the case where the agent performs consecutive loops through the IRTC.

Unless the whole MDP consist solely of one IRTC our case will not be the only situation occurring during learning. If the performance of the agent is tested or learning ceases whilst the agent has been performing consecutive loops through the an IRTC the policy of the agent may have diverged from a previously held estimated optimal policy.

3.3.1 Formal Proofs

When an agent is performing consecutive loops through an IRTC there are two possibilities for how the Q-value will be updated. The difference being the Q-value used to estimate the expected future reward. This can be either the Q-value of the same state-action pair or another state-action pair. The former will occur when the IRTC is like Figure 3.1a. The latter will occur when the IRTC is like Figure 3.1b.

$$\text{Homogeneous SARSA Update: } Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s,a) - Q(s,a)) \tag{3.1}$$

$$\text{Heterogeneous SARSA Update: } Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{3.2}$$

where s and a are the state-action pair being updated, r is the transition reward, r is the transition reward, γ is the future discount factor, and s' and a' are a distinct state-action pair to s and a (at least one of the pairs must be different).

We will now refer to $Q(s,a)$ as Q and $Q(s',a')$ as Q'. We refer to Equation 3.1 as the *homogeneous* SARSA update as the same Q-value is used to estimate the expected future reward as being updated. We refer to Equation 3.2 as the *heterogeneous* SARSA update as a strictly different Q-value is used to estimate the expected future reward as being updated.

We conjecture that the following lemmas hold true under our assumed conditions:

Lemma 1 (Common Recurrence Relation)**.** *The i^{th} update of any Q-value in an IRTC of length n can be rewritten as the same non-homogeneous recurrence relation with n previous terms, where there are continuous, consecutive loops through the IRTC.*

Proof of Common Recurrence Relation. Our overall aim of this proof is to convince the reader that Lemma 1 holds true in the general case of IRTCs of length n. We have been able to prove the special cases of IRTCs of length one and two but have not been able to extend our proof to the general case; nevertheless, we believe our proof should set the groundwork for such an extension. For this reason we present a proof that naturally extends from IRTCs of length one, to IRTCs of length two, and then onward to the general case of IRTCs of length n. We concede that this means the proof presented below is of a higher complexity than is strictly necessary to prove the special case of IRTCs of length one.

We will begin by proving that the i^{th} update of any Q-value IRTC of length 1 or 2 can be written as respectively the same non-homogeneous recurrence relation with 1 or 2 previous terms. We will then conjecture that this can be extended to the general case.

When there is a IRTC of length 1 we find that the SARSA update rule can be easily rearranged to show the recurrence relation. We do this by substituting the Q' for the Q and rearranging the variables. We are able to substitute the Q for Q' since, by definition, in an IRTC of length 1 they

are the same. We now can define the Q-value Q after $i+1$ consecutive loops through the IRTC, or the $i+1$ update, as:

$$\begin{aligned} Q_{i+1} &= Q_i + \alpha \cdot (r + \gamma \cdot Q_i - Q_i) \\ &= Q_i + \alpha r + \alpha\gamma \cdot Q_i - \alpha \cdot Q_i \\ &= (1 + \alpha\gamma - \alpha) \cdot Q_i + \alpha r \end{aligned}$$

When there is a IRTC of length 2 we find that SARSA update rule forms 2 set of equations with reoccurring terms. We define the first Q-value in the IRTC of length 2 as $Q(s_0, a_0) = Q^0$ and the second as $Q(s_1, a_1) = Q^1$ We now consider the $i+1$ consecutive loop through the IRTC. Q^0 is defined in terms of itself and Q^1 of the previous loop, the i^{th} update. Q^1 is defined in terms of itself and Q^0 from the current loop, the $i+1$ update. The two equations then are:

$$\begin{aligned} Q^0_{i+1} &= (1-\alpha)Q^0_i + \alpha\gamma Q^1_i + \alpha r \\ Q^1_{i+1} &= (1-\alpha)Q^1_i + \alpha\gamma Q^0_{i+1} + \alpha r \end{aligned}$$

We can use the second equation to find another definition of Q^0_{i+1} and use the first equation to remove it:

$$Q^0_{i+1} = \frac{Q^1_{i+1} - (1-\alpha)Q^1_i - \alpha r}{\alpha\gamma} = (1-\alpha)Q^0_i + \alpha\gamma Q^1_i + \alpha r$$

$$\alpha\gamma Q^0_{i+1} = Q^1_{i+1} - (1-\alpha)Q^1_i - \alpha r = \alpha\gamma(1-\alpha)Q^0_i + (\alpha\gamma)^2 Q^1_i + \alpha^2\gamma r$$

$$\begin{aligned} ((\alpha\gamma)^2 + (1-\alpha))Q^1_i &= Q^1_{i+1} - \alpha\gamma(1-\alpha)Q^0_i - (\alpha\gamma+1)\alpha r \\ Q^1_i &= \frac{Q^1_{i+1} - \alpha\gamma(1-\alpha)Q^0_i - (\alpha\gamma+1)\alpha r}{(\alpha\gamma)^2 + (1-\alpha)} \end{aligned}$$

Next subtract Q^1_{i+1} from Q^0_{i+1} and substitute in the new of Q^1_i to find Q^1_{i+1} in terms of only

Q^0's:

$$\begin{aligned}
Q^0_{i+1} - Q^1_{i+1} &= (1-\alpha)Q^0_i + \alpha\gamma Q^1_i + \alpha r - (1-\alpha)Q^1_i - \alpha\gamma Q^0_{i+1} - \alpha r \\
&= (1-\alpha)Q^0_i + (\alpha\gamma - (1-\alpha))Q^1_i - \alpha\gamma Q^0_{i+1} \\
Q^1_{i+1} &= Q^0_{i+1} + \alpha\gamma Q^0_{i+1} - (1-\alpha)Q^0_i - (\alpha\gamma - (1-\alpha))Q^1_i \\
&= (1+\alpha\gamma)Q^0_{i+1} - (1-\alpha)Q^0_i - (\alpha\gamma - (1-\alpha))(\frac{Q^1_{i+1} - \alpha\gamma(1-\alpha)Q^0_i - (\alpha\gamma+1)\alpha r}{(\alpha\gamma)^2 + (1-\alpha)})
\end{aligned}$$

$$\begin{aligned}
((\alpha\gamma)^2 + (1-\alpha))Q^1_{i+1} &= ((\alpha\gamma)^2 + (1-\alpha))(1+\alpha\gamma)Q^0_{i+1} - ((\alpha\gamma)^2 + (1-\alpha))(1-\alpha)Q^0_i \\
&\quad - (\alpha\gamma - (1-\alpha))Q^1_{i+1} - (\alpha\gamma - (1-\alpha))\alpha\gamma(1-\alpha)Q^0_i - (\alpha\gamma - (1-\alpha))(\alpha\gamma+1)\alpha r \\
\alpha\gamma(1+\alpha\gamma)Q^1_{i+1} &= ((\alpha\gamma)^2 + (1-\alpha)(\alpha\gamma+1)Q^0_{i+1} - (1-\alpha)^2(1+\alpha\gamma)Q^0_i + (\alpha\gamma - (1-\alpha))(1+\alpha\gamma)\alpha r \\
\alpha\gamma Q^1_{i+1} &= ((\alpha\gamma)^2 + (1-\alpha)Q^0_{i+1} - (1-\alpha)^2 Q^0_i + (\alpha\gamma - (1-\alpha))\alpha r \\
Q^1_{i+1} &= \frac{((\alpha\gamma)^2 + (1-\alpha)Q^0_{i+1} - (1-\alpha)^2 Q^0_i + (\alpha\gamma - (1-\alpha))\alpha r}{\alpha\gamma}
\end{aligned}$$

Finally, we bump the update indices of the original equation for Q^0_{i+1} and substitute in the newest definition of Q^1_{i+1}:

$$\begin{aligned}
Q^0_{i+2} &= (1-\alpha)Q^0_{i+1} + \alpha\gamma Q^1_{i+1} + \alpha r \\
&= (1-\alpha)Q^0_{i+1} + \alpha\gamma\frac{((\alpha\gamma)^2 + (1-\alpha)Q^0_{i+1} - (1-\alpha)^2 Q^0_i + (\alpha\gamma - (1-\alpha))\alpha r}{\alpha\gamma} + \alpha r \\
&= (1-\alpha)Q^0_{i+1} + ((\alpha\gamma)^2 + (1-\alpha)Q^0_{i+1} - (1-\alpha)^2 Q^0_i + (\alpha\gamma - (1-\alpha))\alpha r + \alpha r \\
&= ((\alpha\gamma)^2 + 2(1-\alpha))Q^0_{i+1} - (1-\alpha)^2 Q^0_i + (1 + \alpha\gamma - (1-\alpha)\alpha r \\
&= (2 - 2\alpha + (\alpha\gamma)^2)Q^0_{i+1} - (1-\alpha)^2 Q^0_i + (\gamma+1)\alpha^2 r
\end{aligned}$$

Through a similar process we can show that Q^1 has exactly the same recurrence relation. The initial values for both recurrence relations may vary. We have, therefore, demonstrated that:

IRTC of length 1: $Q^k_i = (1 + \alpha\gamma - \alpha)Q^k_{i-1} + \alpha r$

IRTC of length 2: $Q^k_i = (2 - 2\alpha + (\alpha\gamma)^2)Q^k_{i-1} - (1-\alpha)^2 Q^k_{i-2} + (\gamma+1)\alpha^2 r$

Therefore, the i^{th} update of any Q-value in an IRTC of length n where $n \in \{1, 2\}$ can be written as the same non-homogeneous recurrence relation with n previous terms. This recurrence relation will define the Q-value at the i^{th} consecutive loop through the IRTC. □

Now that we have proven Lemma 1 holds true for IRTCs of length 1 or 2 we move on to our conjecture:[1]

[1] As will become especially apparent in the application to Tile Coding most of the IRTCs encountered will be of length

Conjecture. *Lemma 1 holds true for any IRTC of length n where $n \geq 1$.*

Argument. *In any IRTC of length n where $n > 2$ there are only the same two possible types of SARSA update that can happen from an IRTC of length 2:*

- *for Q_i^k where $0 \leq k < n-2$, the estimated expected future reward is the current value of the next Q-value in the cycle Q_i^{k+1};*
- *for Q_i^{n-1} the estimated expected future reward is the updated value of the first Q-value in the cycle Q_i^0*

We argue that with algebraic manipulation they can be reformulated into a recurrence relation with n previous terms. Furthermore, we have shown that, in the case of $n = 2$, the recurrence relations are the same for both Q-values in the cycle that fit the archetype of the two possible update types. Therefore, we argue that in a similar manner to the case of $n = 2$ for any $n > 2$ every Q-value in an IRTC of length n shares a common recurrence relation with the other Q-values in the cycle. Moreover, there are n previous terms to the recurrence relation.

Therefore, assuming our conjecture holds true, the i^{th} update of any Q-value in an IRTC of length n can be written as the same non-homogeneous recurrence relation with n previous terms, where there are continuous, consecutive loops through the IRTC and every transition yields the same reward.

Lemma 2 (Common Limits)**.** *All Q-values in an IRTC have a common limit dependent on α, γ, and r, where there are continuous, consecutive loops through the IRTC:*

$$\lim_{i \to \infty} Q_i^k = \begin{cases} \frac{r}{1-\gamma} & 0 \leq \gamma < 1 \\ +\infty & r > 0,\ \gamma = 1 \\ -\infty & r < 0,\ \gamma = 1 \end{cases}$$

where $1 \leq k \leq n$ and Q_i^k is the i^{th} update of k^{th} Q-value in the IRTC.

Proof of Common Limits. We will begin by proving the limits of an IRTC of length 1. Then we will reason how these limits apply in the general case.

When there is an IRTC of length 1 we know the recurrence relation is:

$$Q_i = (1 + \alpha\gamma - \alpha)Q_{i-1} + \alpha r$$

The possible values for the parameters of the equation above are: 1) $0 < \alpha \leq 1$; 2) $0 \leq \gamma \leq 1$; 3) $r \in \Re$. We do not allow α to be zero, $\alpha \neq 0$, since no learning would occur and so the Q-values would not change.

1 or 2.

Where $\gamma = 1$ and $r \neq 0$ the above equation can be reduced to:

$$\begin{aligned} Q_{i+1} &= (1 + \alpha - \alpha)Q_i + \alpha r \\ &= Q_i + \alpha r \end{aligned}$$

$$\therefore \lim_{i \to \infty} Q_i = \pm\infty$$

In this special case the series will tend toward $\pm\infty$.

Where $\gamma \neq 1$ the series does converge toward a single value. Since the recurrence relation is non-homogeneous we will manipulate it to remove the constant αr. For simplicity allow $c = \alpha r$ and $d = (1 + \alpha\gamma - \alpha)$ then:

$$\begin{aligned} Q_{i+1} &= dQ_i + c \\ Q_{i+2} &= dQ_{i+1} + c \end{aligned}$$

$$\begin{aligned} Q_{i+2} - Q_{i+1} &= (dQ_{i+1} + \not{c}) - (dQ_i + \not{c}) \\ Q_{i+2} - Q_{i+1} &= dQ_{i+1} - dQ_i \\ Q_{i+2} &= (1 + d)Q_{i+1} - dQ_i \end{aligned}$$

Let $b_1 = (1 + d)$ and $b_2 = -d$:

$$Q_i = b_1 Q_{i-1} + b_2 Q_{i-2}$$

The Distinct Roots Theorem of Recurrence Relations allows us to use a *characteristic function* to reformulate the definition of Q_i in terms of just initial values and parameters (no previous terms). The roots of the characteristic function are used as the variables of the reformulation; the coefficients are constants determined to fit the initial conditions. Let $p(t) = t^2 - b_1 t - b_2$ be a *characteristic equation* for the recurrence relation. First we find the roots of $p(t)$:

$$\begin{aligned} 0 &= t^2 - b_1 t - b_2 \\ 0 &= t^2 - (1 + d)t + d \\ 0 &= (t - d)(t - 1) \end{aligned}$$

$$\begin{aligned} &\therefore \\ &\text{Root } r_1 = d = (1 + \alpha\gamma - \alpha) \\ &\text{Root } r_2 = 1 \end{aligned}$$

Here r_1 and r_2 are the roots of $p(t)$. These roots are then used to define Q_i as follows:

$$\begin{aligned} Q_i &= k_1 r_1^i + k_2 r_2^i \\ &= k_1(1 + \alpha\gamma - \alpha)^i + k_2(1)^i \\ &= k_1(1 + \alpha\gamma - \alpha)^i + k_2 \end{aligned} \tag{3.3}$$

where k_1 and k_2 are determined to fit the initial conditions of the series. From Equation 3.3 it can be said that the series' first term, $k_1(1 + \alpha\gamma - \alpha)$, will at its limit become zero where $\alpha(\gamma - 1) < 0$. Therefore when $\gamma < 1$ the series will converge toward k_2 and when $\gamma = 1$ the series will diverge toward $\pm\infty$:

$$\lim_{i\to\infty}(1 + \alpha\gamma - \alpha)^i = \begin{cases} 0 & \text{where } \gamma < 0 \\ \pm\infty & \text{where } \gamma = 1 \end{cases}$$

In the cases where the series does converge we can use the original SARSA update rule to determine the value series converges toward, k_2. At convergence $Q_i = Q_{i-1}$, therefore we will substitute in Q for both these values:

$$\begin{aligned} Q &= (1 + \alpha\gamma - \alpha)Q + \alpha r \\ Q - (1 + \alpha\gamma - \alpha)Q &= \alpha r \\ \alpha(1 - \gamma)Q &= \alpha r \\ Q &= \frac{\alpha r}{1 - \gamma} \end{aligned}$$

$$\therefore \lim_{i\to\infty} Q_i = \frac{\alpha r}{1 - \gamma}$$

Now that k_2 is known Equation 3.3 can be rearranged so that k_1 can be found with only the initial value and the parameters of the series:

$$\begin{aligned} Q_i &= k_1 \cdot (1 + \alpha\gamma - \alpha)^i + k_2 \\ k_1 &= \frac{Q_i - k_2}{(1 + \alpha\gamma - \alpha)^i} \\ &= \frac{Q_i - (\frac{r}{1-\gamma})}{(1 + \alpha\gamma - \alpha)^i} \end{aligned} \tag{3.4}$$

where $i = 1$

$$k_1 = \frac{Q_0 - (\frac{r}{1-\gamma})}{(1 + \alpha\gamma - \alpha)}$$

In the above case where the IRTC is of length 1 we see that the update rule used to determine the recurrence relation and the limit was the homogeneous SARSA update rule of Equation 3.1. When we consider IRTCs of length greater than 1 we see that the update rule to be followed is the heterogeneous SARSA update rule of Equation 3.2. The estimated expected future reward Q' is the next Q-value in the cycle. We have already shown that any Q-value in an IRTC share the same recurrence relation with the other Q-values of that same IRTC. We argue that because of this all the Q-values of an IRTC of a particular length exhibit the same limits. Therefore, at the point of convergence Q equals Q'. This then transforms the heterogeneous limit into the homogeneous limit, which we have already shown the limits for.

Therefore, all Q-values in an IRTC have a common limit dependent on α, γ, and r, where there are continuous, consecutive loops through the IRTC and every transition yields the same reward. □

The only case not proven a limit for is when $\gamma = 1$ and $r = 0$. In this case empirical analysis shows that where the IRTC's length is greater than 1 and $\alpha < 1$ the series do converge to a single value but that value is determined by the initial Q-values and α. Empirically we can say that the point was always within the range of Q^0 to Q^{n-1} and tends toward the average of the initial values the smaller α is. In this same case empirical analysis shows that where $\alpha = 1$ and the IRTC's length is greater than 1 the series does not converge rather the Q-values cycle through the values of Q^1 to Q^{n-1} (here Q^0's value is lost).

Lemma 3 (Optimal Policy Change)**.** *Before convergence during continuous, consecutive loops through an IRTC where every transition reward is the same the currently held optimal policy can change. The frequency of this happening is dependent on the initial Q-values, α, γ, and r.*

Proof of Optimal Policy Change. For the optimal policy to change at least one state s_k in the IRTC of length n where $0 \leq k < n$ must have another action a' available to perform on it. Where this is true there are three possible cases to investigate where:

1. the Q-value of (s_k, a_k) in the IRTC is the highest of all Q-values on s_k, $\arg\max_a Q(s_k, a) = Q(s_k, a_k)$

2. another Q-value other than the Q-value of (s_k, a_k) is the highest of all Q-values on s_k, $\arg\max_a Q(s_k, a) = Q(s_k, a')$ where $a \neq a'$

3. two Q-values of s_k are maximal $\arg\max_a Q(s_k, a) = Q(s_k, a_k) = Q(s_k, a')$ where $a \neq a'$

In all of these cases it is possible for the currently held optimal policy to change given the amenable parameter values of α, γ, and r:

1. if $\arg\max_a Q(s_k, a) = Q(s_k, a_k)$ and $\lim_{i\to\infty} Q_i(s_k, a_k) <= Q(s_k, a')$ then the policy will change

2. if $\arg\max_a Q(s_k, a) = Q(s_k, a')$ and $\lim_{i\to\infty} Q_i(s_k, a_k) >= Q(s_k, a')$ then the policy will change

3. if $\arg\max_a Q(s_k, a) = Q(s_k, a_k) = Q(s_k, a')$ and $\lim_{i\to\infty} Q_i(s_k, a_k) \neq Q(s_k, a')$ then the policy will change

Therefore in all cases it is possible for the currently held optimal policy to change. Where the current policy is the true optimal policy then the agent will have diverged from learning the optimal policy.

Furthermore, the Q-value with the highest value in the IRTC may change during continuous, consecutive loops through the IRTC. There are two possible general cases where this can occur:

1. $\max(Q(s_0, a_0), ..., Q(s_{n-1}, a_{n-1})) = Q_i^{k+1}$ and $Q_i^k < \lim_{i\to\infty} Q_i^k$

2. $\max(Q(s_0, a_0), ..., Q(s_{n-1}, a_{n-1})) = Q_i^k$ and $Q_i^k > \lim_{i\to\infty} Q_i^k$

In the first case the next Q-value in the IRTC must have the highest value and the current Q-value in the IRTC must be less than the limit of the IRTC. In the second case the current Q-value in the IRTC must have the highest value and also be greater than the limit of the IRTC.

In the first case for the Q-value with the highest value to change we require that $Q_i^k < Q_i^{k+1} < Q_{i+1}^k$. By using the SARSA update rule and substituting in the inequality we can find

the range of values Q_i^{k+1} must have compared to Q_i^k for this case to occur:

$$Q_{i+1}^k = (1-\alpha)Q_i^k + \alpha\gamma Q_i^{k+1} + \alpha r$$

$$\begin{aligned} \therefore Q_i^{k+1} &< (1-\alpha)Q_i^k + \alpha\gamma Q_i^{k+1} + \alpha r \\ (1-\alpha\gamma)Q_i^{k+1} &< (1-\alpha)Q_i^k + \alpha r \\ Q_i^{k+1} &< \frac{(1-\alpha)Q_i^k + \alpha r}{1-\alpha\gamma} \end{aligned}$$

$$\therefore Q_i^k < Q_i^{k+1} < \frac{(1-\alpha)Q_i^k + \alpha r}{1-\alpha\gamma}$$

In the second case we simply need to reverse the inequalities to find the range of values Q_i^{k+1} must have compared to Q_i^k:

$$Q_i^k > Q_i^{k+1} > \frac{(1-\alpha)Q_i^k + \alpha r}{1-\alpha\gamma}$$

This also means that it is possible for the currently held highest Q-value in an IRTC to propagate backwards through the cycle stepping back at most once per complete cycle. This proves that if a state s is present more than once in an IRTC, $s = s_k = s_j$ where $k \neq j$ then the currently held optimal policy on that state can change under the above conditions.

Therefore, before convergence during continuous, consecutive loops through an IRTC where every transition yields the same reward the currently held optimal policy can change. □

Lemma 4 (Rate of Convergence Factors)**.** *The rate of convergence of a Q-value toward the common limit of an IRTC is dependent on the initial Q-values in the IRTC, α, γ, and r.*

Proof of Rate of Convergence Factors. We have already shown that the i^{th} update of any Q-value in an IRTC can be rewritten as a recurrence relation where there are continuous, consecutive loops through the cycle and the transition reward is the same. We have also already shown that limits are dependent on α, γ, and r. Therefore we can confidently state that the factors that affect the rate of convergence are the initial values of the Q-values in the cycle, α, γ, and r.

In the case of an IRTC of length 1 we can precisely define the rate of convergence. Assumption 3 states that computers approximate number values and therefore there is a variable value Δ where if the difference between two numbers is less than Δ they are indistinguishable to the computer thereby effectively converging.

The point of convergence is important to the definition of Δ since the value of Δ is dependent on the number it is trying to represent. In single-point precision (float) ≈ 7 significant figures in base 10 can be stored, in double-point precision (double) ≈ 16 significant figures. Computers

store numbers in normalised base 2 format with the exponent in single-point precision ranging from 128 to -127 and in double-point precision ranging from 1024 to -1023. This means numbers close to 0 can be extremely more accurate than those above to $|1|$.

Δ is calculated by knowing the direction that k_2 is being approached and the value of k_2. Once this is known the number of updates required is as follows:

$$k_1 \cdot (1 + \alpha\gamma - \alpha)^i \leq \Delta \tag{3.5}$$

$$\begin{aligned}(1 + \alpha\gamma - \alpha)^i &\leq \frac{\Delta}{k_1} \\ i &\leq \log_{(1+\alpha\gamma-\alpha)}(\frac{\Delta}{k_1})\end{aligned} \tag{3.6}$$

$$\begin{aligned}\therefore \text{ where } i &= \lceil \log_{(1+\alpha\gamma-\alpha)}(\frac{\Delta}{k_1}) \rceil \\ k_1 \cdot (1 + \alpha\gamma - \alpha)^i &\leq \Delta\end{aligned}$$

Equation 3.6 shows the parameters that affect the rate of convergence: the discount factor, γ; the learning rate, α; and the coefficient of the series, k_1. Below is a detailed list of what effects the rate of convergence and how:

- $\mathbf{k_2}$ effects the rate of convergence by effecting the value of Δ[2]
- $\boldsymbol{\gamma}$ has a direct relation to the rate of convergence
- $\boldsymbol{\alpha}$ has an inverse relation to the rate of convergence
- $\mathbf{k_1}$ has a direct relation to the rate of convergence

Therefore, the rate of convergence is dependent on the initial Q-values in the IRTC, α, γ, and r. □

3.3.2 Theorems

Theorem 1 (Correlation Theorem)**.** *There is a direct correlation between the probability of an agent performing consecutive loops through an IRTC that contains at least one state in the currently held optimal path where every transition yields the same reward and the probability that the currently held optimal policy will change.*

Proof of Correlation Theorem. From Lemmas 1 and 2 we can say that if the Q-values in the IRTC reach the limit, or some approximation of it such that their values become indistinguishable from each other, then the optimal policy could have changed. If the IRTC included a state-action pair that was in the currently held optimal policy and the limit is less than or equal to that of another

[2]Since the value of Δ is dependent on the convergence point.

Q-value on that same state then the policy will have changed. If the IRTC included a state that was in the currently held optimal policy and the limit is greater than or equal to that of the Q-value of the currently held optimal action in that same state then the policy will have changed.

From Lemma 3 we can say that if one of the states in the IRTC is also in the currently held optimal path then it is possible that its Q-value may change in such a manner that the currently held optimal path changes. In the first two cases where it is possible for the optimal policy to change it is required that Q-values in the IRTC are within a certain distance from each other. Since Lemma 2 states all values in the IRTC tend toward a common limit we know that the distance between the Q-values in the IRTC must decrease over time. Therefore given enough consecutive loops through the IRTC the conditions required by Lemma 3 will occur and so the policy will change before convergence has occurred.

Therefore, the greater the probability that consecutive loops through an IRTC will occur the greater the probability that the optimal policy will change if the IRTC contains at least one state in the optimal path. □

Theorem 2 (Probability Theorem)**.** *There are two ways to decrease the probability of the currently held optimal policy changing due to an agent's interaction with IRTCs: 1) decrease the Q-values' rate of convergence toward their common limit; or 2) decrease the probability the agent will perform consecutive loops through the IRTC.*

Proof of Probability Theorem. From Lemma 4 we know that the rate of convergence when an agent performs consecutive loops through an IRTC is dependent on the initial Q-values, α, γ, and r. The Q-values at the point when an agent begins to consecutively loop through an IRTC cannot be easily controlled unless the entire environment consists of a single IRTC. Altering the future discount factor γ can alter the goal of the environment and so should not be done. Whilst altering the learning rate α would alter the rate of convergence toward the limit of the IRTC Lemma 3 states that before convergence to the limit the policy can change; moreover, altering α would not impact the probability of an agent performing consecutive loops through an IRTC.

By the definition of an IRTC there are three ways for an agent to stop performing consecutive loops through an IRTC: 1) receive a transition reward strictly different to the transition reward of the IRTC; 2) transition to a state that is not a member of the IRTC; or 3) perform a different action than the next prescribed action of the IRTC. The agent has no control over the first two options since they are dependent on the reward function and the transition function respectively. The agent does have control over the last option; by recording a history of its state-action pairs along with the reward an agent could detect that it was in an IRTC and then attempt an action to stop consecutive loops. However, whilst this does guarantee that the agent will exit the current IRTC the agent could be entering a new IRTC or an extension of the original IRTC. □

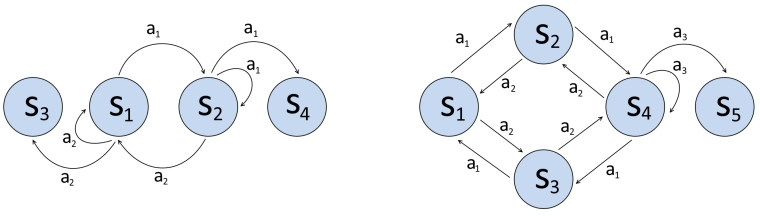
3.3.3 Empirical Demonstration

We will now conduct a series of experiments to empirically demonstrate the lemmas and theorems presented above. We will perform experiments in 3 different MDP environments.

The first MDP environment purely consists of IRTC. The experiment will be repeated with different lengths of IRTC. This will demonstrate that consecutive, continuous loops through an IRTC will lead to all Q-values in the IRTC tending toward a common limit (Lemmas 1 and 2). The MDP models the sub-MDPs shown in Figure 3.1.

The second MDP environment consists of 4 states, two of which are goal states. The agent begins in either state s_1 or s_2 and the episode will end when the agent arrives in state s_3 or s_4. There are 2 actions available a_1 and a_2. The reward for reaching state s_4 is 1, the reward for reaching state s_3 is -1, and all other transition rewards yield 0. Figure 3.2a shows the MDP. The probability of transitioning from a non-goal state to a goal state is 0.5, the probability of transitioning between two non-goal states is 1. The second MDP is designed in this way to demonstrate that even after an agent has learnt the optimal policy, the interaction with IRTCs can cause the agent's policy to temporarily diverge. Moreover, it is only with further exploration that the temporary divergence ceases.

The third MDP environment consists of 5 states, one of which is a goal state. The agent begins in state s_1 and the episode ends when the agent arrives in state s_5. There are two actions available in all states expect s_4 where there are 3 actions. The reward for arriving in s_5 is 1, all other transitions yield a reward of 0. Figure 3.2b shows the MDP. The third MDP is designed to demonstrate that another goal state is not required for temporary divergence from the optimal policy to occur.



(a) Second MDP (b) Third MDP

Figure 3.2: MDP environments to demonstrate theory

3.3.3.1 Experiments & Results

In our first environment, an MDP that purely consists of an IRTC; we have set up each the results to show examples, for each length of IRTC, the Q-values tending toward each of the three limits described in Lemma 2. We randomly choose the values of r, α, and γ for each experiment to

demonstrate that the theory is not dependent on particular parameter values. However, α and γ where within than bounds, $r \leq |100|$, and in the cases where the limit is $\pm\infty$ γ was set to 1.

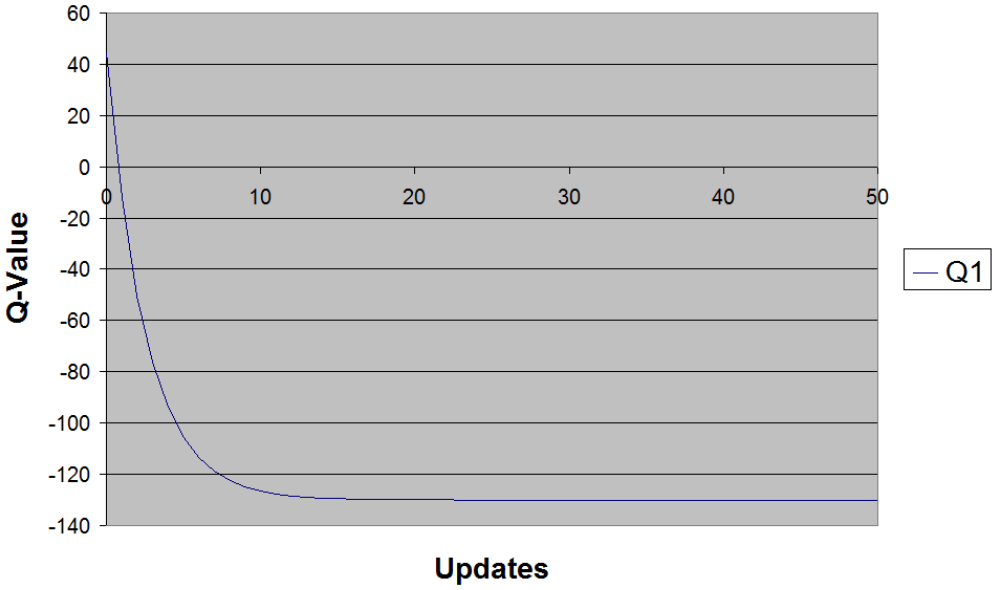


Figure 3.3: Results from an MDP consisting purely of a single IRTC of length 1. Transition Length = 1, $\alpha = 0.899628$, $\gamma = 0.63834$, $r = -47.018$

The graphs in Figures 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, and 3.11 show Q-values in the IRTC over the number of complete loops through the IRTC. The x-axis represents the number of updates the Q-values have had and the y-axis represents the value of the Q-value. Furthermore, Qn is the n^{th} Q-value in the IRTC. We find that the graphs in these Figures show precisely what the theory predicted. Lemmas 1 and 2 state that each Q-value in an IRTC has the same recurrence relation and that their limits are common. Lemma 3 states that the Q-value with the greatest value can change.

In our second and third MDP environment we set up the parameters as stated below the graphs (see Figure 3.14 and 3.15). These results show how even if looping through an IRTC is not continuous, if enough the policy can change. Furthermore, the Q-values never reach the limit, in this case 0, but the greedy policy does change. This empirically demonstrates Lemma 3. Moreover, the comparison between $\alpha = 0.2$ and $\alpha = 0.4$ strengthens Lemma 4 since the change in α coincides with a change in the frequency at which the policy changes. The graphs in Figures 3.14 and 3.15 show the Q-values changing over time. The x-axis shows the total number of numbers that have occurred and the y-axis shows the Q-value at that point.

In Figure 3.14 we observe all Q-values in the environment. We observe that the greedy policy changes when $Q(s_2, a_1)$'s value drops below $Q(s_2, a_2)$. At the times this occurs the

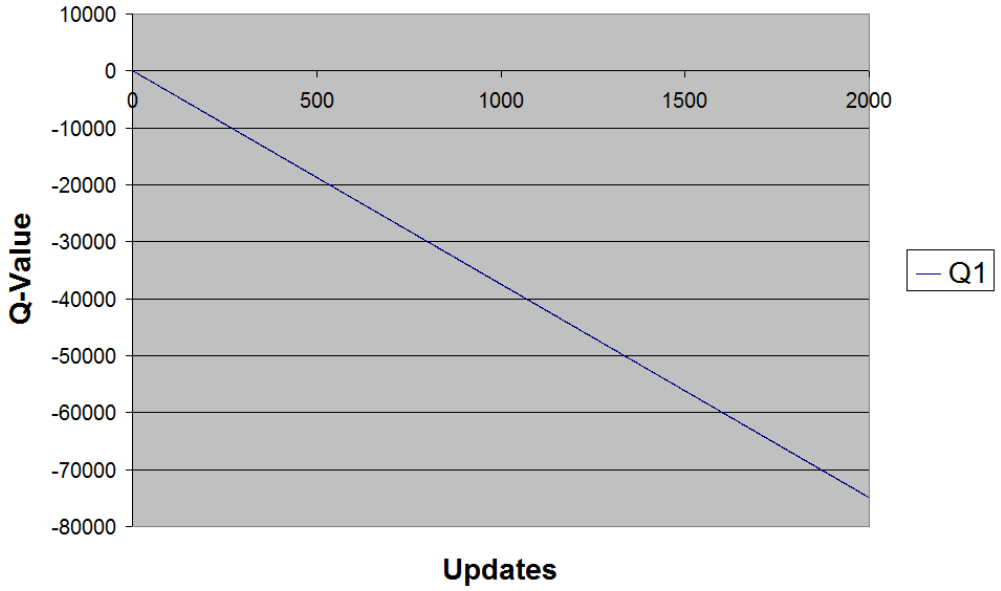


Figure 3.4: Results from an MDP consisting purely of a single IRTC of length 1. Transition Length = 1, $\alpha = 0.83881$, $\gamma = 1$, $r = -44.6894$

agents currently held optimal policy is to never perform a_1 in s_2 and therefore never receive the reward of 1. Furthermore, we observe that in the case where $\alpha = 0.4$ around update 3300 that at the same time as $Q(s_2, a_2) > Q(s_2, a_1)$ $Q(s_1, a_2) > Q(s_1, a_1)$; at this point the agent's currently held optimal policy is to move to s_3, the most sub-optimal policy. Therefore, these results empirically demonstrate our above theory.

In Figure 3.14 we observe the 3 Q-values available in state s_4. We observe that for the most part $Q(s_4, a_3)$ has the highest value. This matches the correct optimal policy of the environment. However, on occasion we observe that its Q-value drops below one of the other Q-values. At these points the agent's currently held optimal policy is not to attempt to move to state s_5 where the reward is 1, rather move to state s_2 or s_3. Therefore, once again these results empirically demonstrate that the policy can change before convergence has occurred in MDPs that have IRTCs.

3.3.4 Conclusion

We have stated that when SARSA is used and there are IRTCs present in the MDP, learning can temporarily diverge from the optimal policy. It occurs when the agent performs consecutive loops through the IRTC though it hard to predict exactly when the policy will change without knowing the future actions the agent will take and the transitions that will occur. It can be said, as stated in Theorem 1, there is a correlation between performing consecutive loops through such an IRTC and the policy changing. Therefore, it is not recommended to perform consecutive loops through

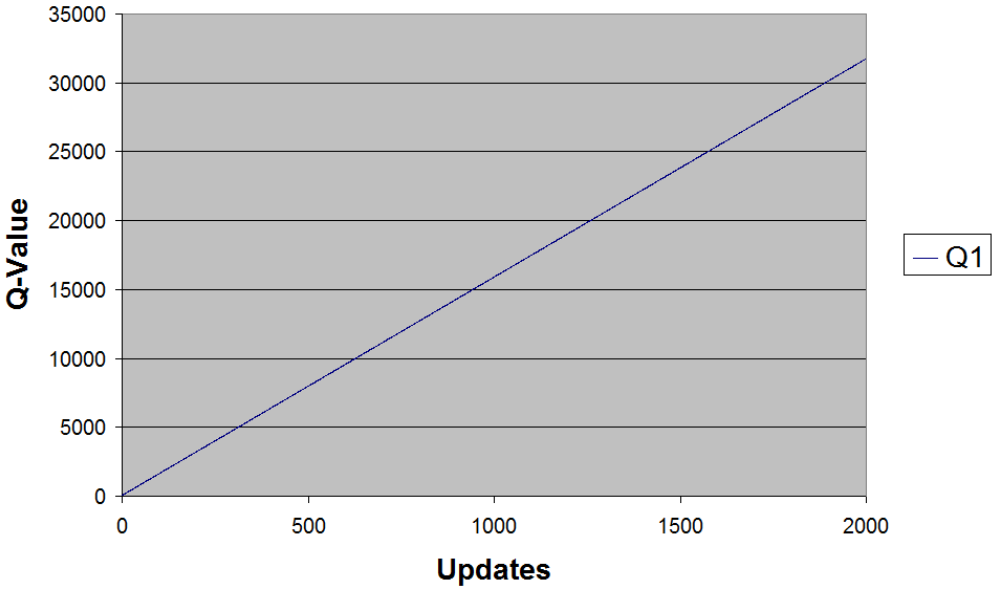


Figure 3.5: Results from an MDP consisting purely of a single IRTC of length 1. Transition Length = 1, $\alpha = 0.74085$, $\gamma = 1$, $r = 21.39147$

an IRTC. Theorem 2 states that to reduce the probability of performing consecutive loops either the agent must try to increase the probability of receiving varying rewards or visit more states. Furthermore, we have empirically shown that the theory holds and in MDPs which contain such IRTCs the policy can change, diverging from the currently held optimal policy.

3.4 Extension to Q-Learning

The phenomenon described by our theory in the previous section is not limited to SARSA but can be extended to include Q-learning. However, Assumption 1 must be altered. The other two assumptions to not need to be changed.

3.4.1 Altered Assumptions

The theoretical properties of SARSA presented in the previous section can be extended to include Q-learning with the following alteration to the assumptions:

Assumption (Transition Cycle)**.** *There exists a state s_0, a sequence of states s_1 to s_n, and a sequence of actions a_1 to a_n where $s_0 = s_n$, $n \geq 1$, and $\forall a_i, a_i = \arg\max_a Q(s_{i-1}, a)$ that starting in s_0 executing a_1 to a_n in order will result in the states s_1 to s_n being visited in order. This is called an Argmax Transition Cycle.*

We define an Argmax Identical Reward Transition Cycle ($\arg\max$ IRTC) in the same way we define an IRTC however the transition cycle of an $\arg\max$ IRTC must be an Argmax Transition

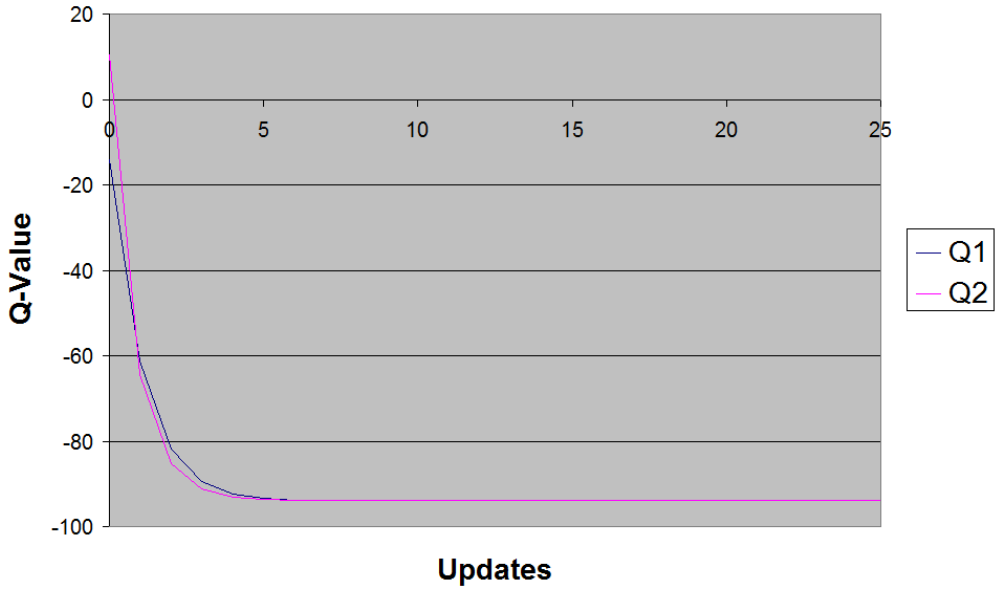


Figure 3.6: Results from an MDP consisting purely of a single IRTC of length 2. Transition Length = 2, $\alpha = 0.756187$, $\gamma = 0.168648$, $r = -78.1331$

Cycle.

The alteration to Assumption 1 is straight forward. The key and only difference is that every action in the sequence a_1 to a_n must be the $\arg\max_a$ action in the corresponding state in the sequence of states s_0 to s_{n-1}. This alteration makes the assumption significantly more restricting. We expect that since the $\arg\max_a$ of a state can change during consecutive loops through an IRTC, consecutive loops through an $\arg\max$ IRTC will halt at the point of the $\arg\max_a$ of a state changing. Furthermore, it is harder to model the updating Q-values when an agent uses Q-Learning since it is an off-policy learning algorithm. This means that steps through the $\arg\max$ IRTC do not need to occur directly after each other but they must occur in order. However, so long as this assumption holds so does all the theory proven in the previous section.

3.4.2 Formal Proofs and Theorems

A Q-Learning agent has the same type of possible Q-value updates when performing consecutive loops through an $\arg\max$ IRTC that a SARSA agent has when performing consecutive loops through a general IRTC. It follows, therefore, that the Lemmas used in the previous section to show the theoretical properties of SARSA can be altered without reproofs by requiring them to use $\arg\max$ IRTCs in lieu of general IRTCs. A special note must be made in regards of Lemma 3 as, by definition, when the agent's policy changes as described in the Lemma then that $\arg\max$ IRTC the agent was looping through will cease to exist. However, by that same definition the agent will have temporarily diverged from its currently held optimal policy.

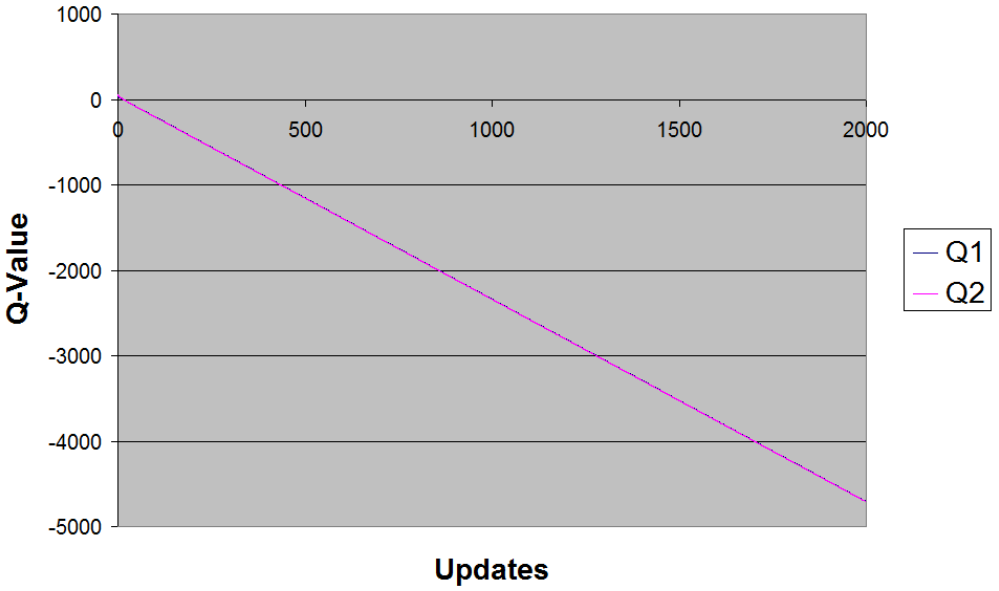


Figure 3.7: Results from an MDP consisting purely of a single IRTC of length 2. Transition Length = 2, $\alpha = 0.094648$, $\gamma = 1$, $r = -25.0441$

In the same manner as the Lemmas the Theorems that apply to a SARSA agent can be altered without reproof to apply to a Q-Learning agent. The theorems for a Q-Learning agent require an $\arg\max$ IRTC in lieu of a general IRTC.

3.4.3 Empirical Demonstration

The second and third MDPs (see Figures 3.2a and 3.2b) will be used to empirically demonstrate that in environments where $\arg\max$ IRTCs exit, consecutive loops can lead to policy change. There is no need to repeat the experiments of the first MDP where the MDP purely consists of a single IRTC as such is, by definition, an $\arg\max$ IRTC.

In our second and third MDP environment we set up the parameters as stated below the graphs. These results show how even if looping through an IRTC is not continuous, if regular enough the policy can change. Furthermore, the Q-values never reach the limit, in this case 0, but the greedy policy does change. This empirically demonstrates Lemma 3 for Q-Learning. Moreover, the comparison between $\alpha = 0.2$ and $\alpha = 0.4$ strengthens Lemma 4 since the change in α coincides with a change in the frequency at which the policy changes. The graphs in Figures 3.14 and 3.15 show the Q-values changing over time. The x-axis shows the total number of numbers that have occurred and the y-axis shows the Q-value at that point.

In Figure 3.14 we observe all Q-values in the environment. We observe that the greedy policy changes when $Q(s_2, a_1)$'s value drops below $Q(s_2, a_2)$. At the times this occurs the agent's currently held optimal policy is to never perform a_1 in s_2 and therefore never receive the reward

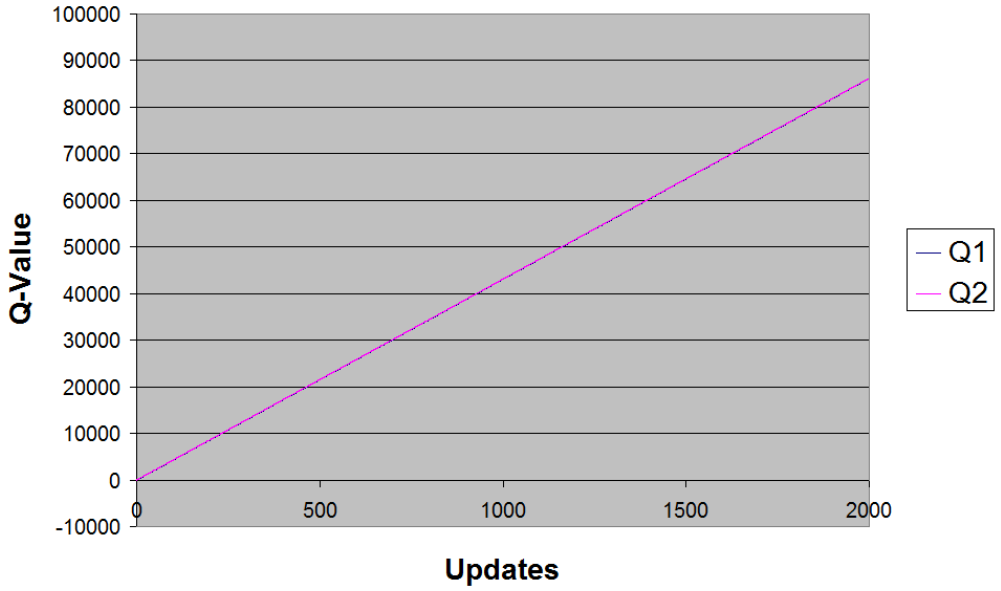


Figure 3.8: Results from an MDP consisting purely of a single IRTC of length 2. Transition Length = 2, $\alpha = 0.683996$, $\gamma = 1$, $r = 63.0188$

of 1. We observe that, compared to SARSA updates, the policy is more consistent. Furthermore, the distance between the Q-value for the worst action and the other Q-values stays greater than that of SARSA. However, we do observe the policy change, empirically demonstrating the theory applies to Q-learning where $\arg\max$ IRTCs exist.

In Figure 3.14 we observe the 3 Q-values available in state s_4. We observe that for the most part $Q(s_4, a_3)$ has the highest value. This matches the correct optimal policy of the environment. However, on occasion we observe that, where $\alpha = 0.4$, its Q-value drops below one of the other Q-values. At these points the agent's currently held optimal policy is not to attempt to move to state s_5 where the reward is 1, rather move to state s_2 or s_3. We do not see the same in the case where $\alpha = 0.2$, we do observe that the value of $Q(s_4, a_3)$ does regularly drop and we can observe the beginnings of a convergence curve. However, because of the lower value of α and the probability of the reaching the goal state the Q-value does not ever drop lower enough to become worse than the other Q-values on s_4. Therefore, once again these results empirically demonstrate that the policy can change before convergence has occurred in MDPs that have $\arg\max$ IRTCs but these policy changes occur less often then environments used.

3.4.4 Conclusion

We have stated that with alteration to the assumptions presented in the previous section the same lemmas and theorems apply to an agent using the Q-Learning algorithm. The altered assumption, the transition cycle assumption, is straight forward and required; the new assumption is signi-

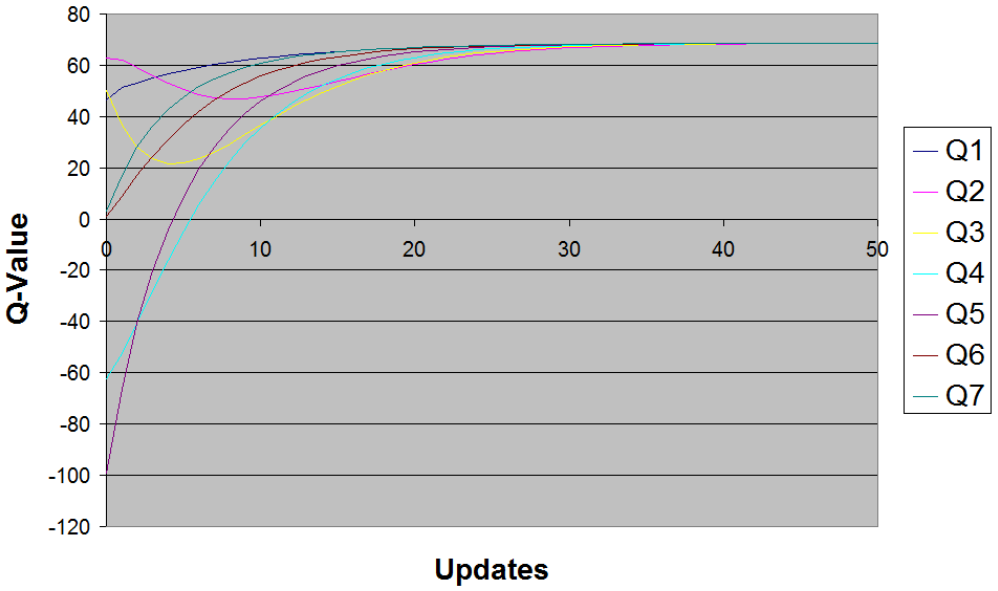


Figure 3.9: Results from an MDP consisting purely of a single IRTC of length 7. Transition Length = 7, $\alpha = 0.250782$, $\gamma = 0.54597$, $r = 31.16617$

ficantly more restrictive to the MDPs the theory will apply. Furthermore, we have empirically demonstrated that the same phenomenon occurs in MDPs with $\arg\max$ IRTCs when Q-Learning is used as when in MDPs where any IRTC is present and SARSA is used.

3.5 Conclusions

When an agent is learning using SARSA and there exists IRTCs where every transition yields the same reward the agent can temporarily diverge away from a learnt optimal policy while it performs consecutive loops through such an IRTC. This can happen at any point in learning, even after the agent can have appeared to settle into a stable policy. This means that a poorly timed decision to stop an agent's learning at such a point can result in an agent no longer being able to properly behave in an environment. This is also the case when an agent is learning using Q-Learning and there exists $\arg\max$ IRTCs where every transition yield the same reward.

The theory presented in this chapter that explains the phenomenon that can be seen empirically can more rigorously proven.

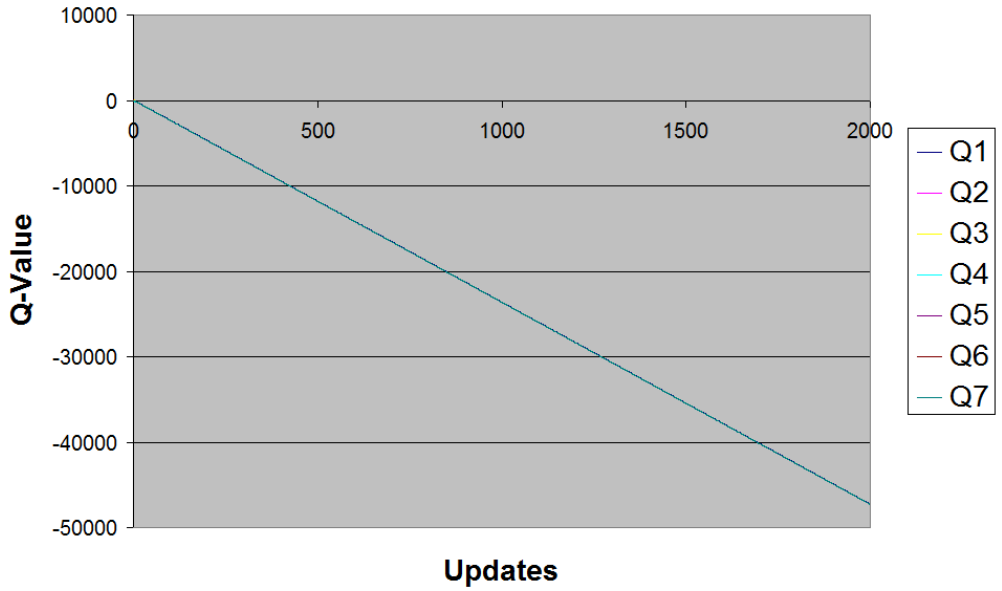


Figure 3.10: Results from an MDP consisting purely of a single IRTC of length 7. Transition Length = 7, $\alpha = 0.886697$, $\gamma = 1$, $r = -26.6789$

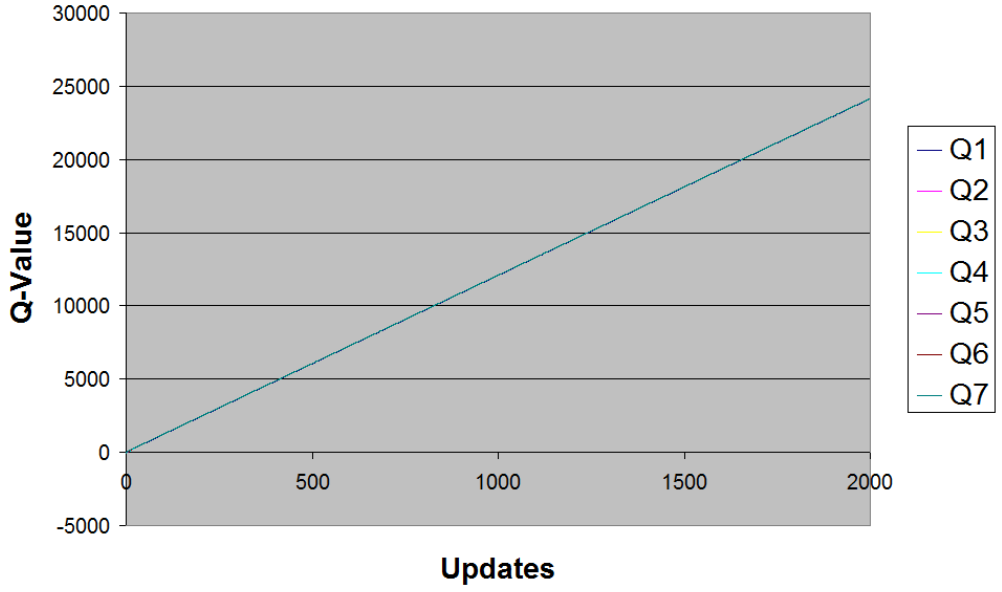
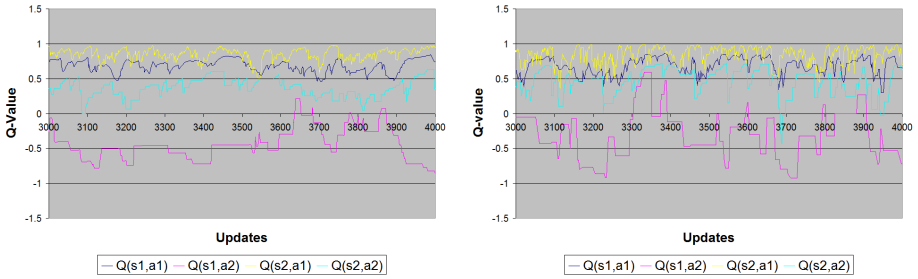
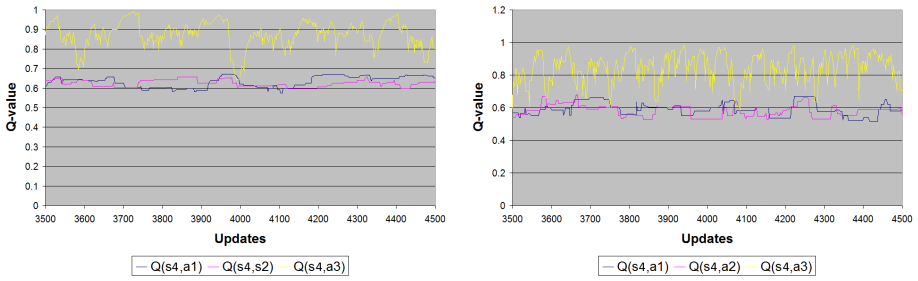


Figure 3.11: Results from an MDP consisting purely of a single IRTC of length 7. Transition Length = 7, $\alpha = 0.282216$, $\gamma = 1$, $r = 42.75248$



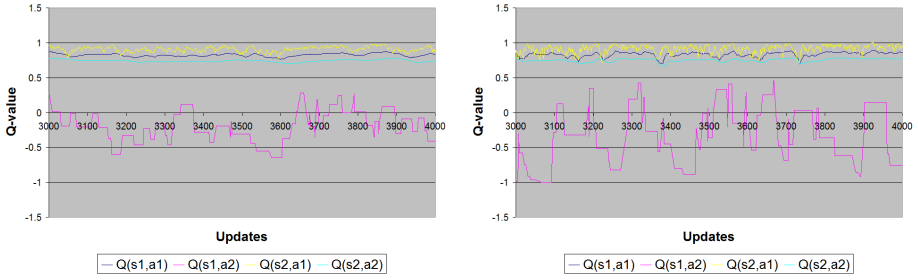
(a) $\alpha = 0.2, \gamma = 0.9$ (b) $\alpha = 0.4, \gamma = 0.9$

Figure 3.12: SARSA results from the MDP in Figure 3.2a



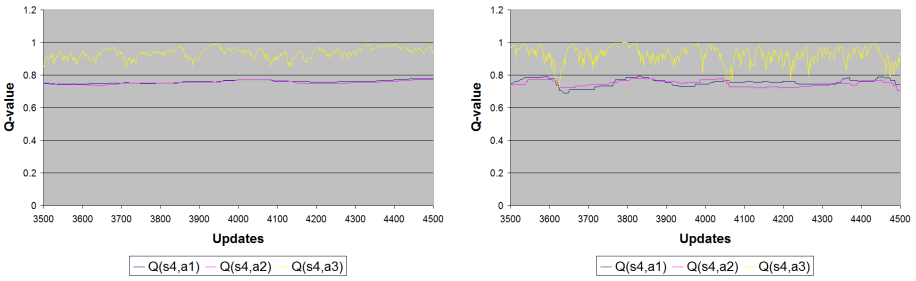
(a) $\alpha = 0.2, \gamma = 0.9$ (b) $\alpha = 0.4, \gamma = 0.9$

Figure 3.13: SARSA results from the MDP in Figure 3.2b



(a) $\alpha = 0.2, \gamma = 0.9$ (b) $\alpha = 0.4, \gamma = 0.9$

Figure 3.14: Q-Learning results from the MDP in Figure 3.2a



(a) $\alpha = 0.2, \gamma = 0.9$ (b) $\alpha = 0.4, \gamma = 0.9$

Figure 3.15: Q-Learning results from the MDP in Figure 3.2b

CHAPTER 4

Application to Tile Coding

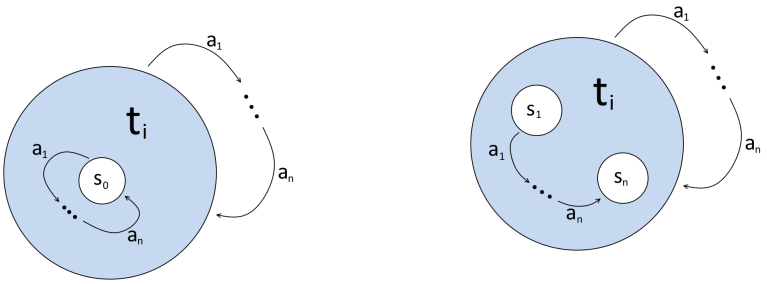
This chapter takes the theoretical properties of SARSA and Q-Learning and applies them to Tile Coding. This chapter introduces a new manual tile placement algorithm called Mixed Resolution Acyclic Tiling (MRAT). The theory from Chapter 3 will be used to derive heuristics, which will then be used to derive MRAT. Finally, in Chapter 5 an automated version of MRAT will be presented.

This chapter opens by applying the theoretical properties of SARSA and Q-Learning to Tile Coding. Then it motivates and conducts an empirical study on the impact that the shape of tile has on learning. Next, this chapter motivates the use of heuristics and their importance in solving problems. This chapter then proceeds to present the heuristics that can be derived from the theory present in Chapter 3 and the empirical results of study of tile shapes. Finally, with the derived heuristics, this chapter presents and discusses Mixed Resolution Tiling with an empirical study to verify its validity.

4.1 Application to Tile Coding

In Tile Coding, the one or more state features are exhaustively partitioned. Each partition being called a tile and the set of tiles is called a tiling. When TC is used, sets of states are grouped into a single tile. This means that the perceived IRTC would be the length of the smallest repeated subsequence of actions (i.e., if the action sequence was $1, 2, 1, 1, 2, 1$ the IRTC length would be three). Furthermore, if there is a path through the states in a tile the agent learning would also perceive this path, if there were repeated sequence of actions, as an IRTC with the length of that sequence. This also applied to when there is a path with repeated actions or an IRTC between

states in a fewer number of tiles. Effectively TC can, in the worst case, reduce the length of an IRTC to the minimum possible based on the size of the repeated subsequence of actions in an IRTC or create IRTCs where in the MDP there was none.



(a) IRTC with n states and j unique actions within a tile becoming an IRTC of j Q-values

(b) Transition path with n states and j unique actions within a tile becoming an IRTC of j Q-values

Figure 4.1: Examples of IRTCs within in a tile

Figure 4.1 shows two of the examples discussed where a longer IRTC or transition path in the MDP becomes a shorter IRTC. This would suggest that when TC is used alongside SARSA, as is often the recommended case, the agent learning is more susceptible to policy divergence from a learnt optimal policy. This suggests that function approximation methods like TC fundamentally alter the MDP in a way that can be detrimental to learning. This also applies to agents using Q-Learning with TC where $\arg\max$ IRTCs or $\arg\max$ transition paths within a tile exist in the MDP.

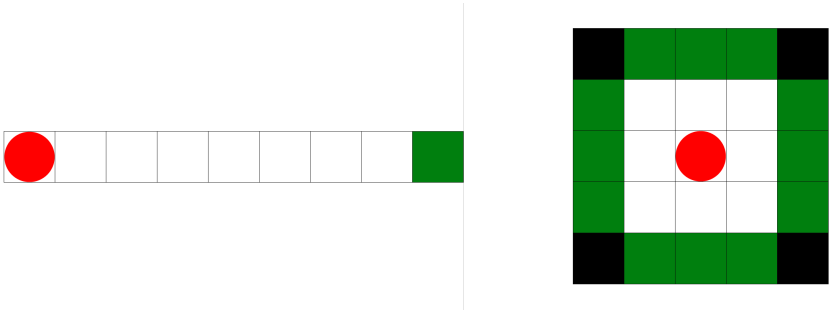
We know the theory described in this chapter applies to all MDPs that meet the assumptions stated for SARSA and the altered assumptions for Q-Learning. However, the second theorem, Theorem 2, is not useful in the general case since for an agent to be able to increase the probability of receiving a different reward or increase the number of states it is visiting it would require the transition (and reward) function. By definition, if the agent has both functions it has solved the problem, does not need to learn, and therefore the theory is moot. If the agent knows the transition function, it should be able to precisely learn the reward function and again the theory is moot since other techniques can be used to find the best path through a known MDP.

In the case of Tile Coding, however, an agent has more ability to increase the number of tiles it is visiting by reducing the size of the tiles it is currently visiting. Therefore, without the agent knowing the transition function the agent can potentially increase the number of tiles being visited. This, in turn, will reduce the probability of the policy changing since the probability of looping through an IRTC is also decreased thereby increasing the reliability of the agent's learning.

Experiments & Results

In this section we shall empirically demonstrate the application of the theory presented in the previous chapter. We will start by introducing the environments that are to be used. Two forms of the Random Walk problem will be used and are intentionally simple to allow the dynamics of learning to be observed clearly yet not so simple as to be trivial.

Environments



(a) 1d-Random Walk (b) 2d-Random Walk

Figure 4.2: Random Walk Environments (Agents moves from the red circle to a green square)

1d-Random Walk (1d-RW) is a 1-dimensional problem where the agent must traverse a corridor from one end of the environment to the other. The length of the corridor and the per-step reward are parameters of the environment. There are two possible actions: *FORWARD* and *BACKWARD*; and one goal state. This environment was used to demonstrate empirically the theoretical proofs. The top image in Figure 4.2a illustrates 1d-RW of length 15 corridor where the agent starts on the square with the red circle and the goal is the green square.

2d-Random Walk (2d-RW) is a 2-dimensional problem where the agent must traverse from some starting point, usually the centre of the environment, to the righthandside wall. The height and width of the environment, whether the other walls are sub-optimal goals, and the per-step reward are parameters. There are four possible actions: *NORTH*, *EAST*, *SOUTH*, and *WEST*; and one wall of optimal goal states with the option of 3 walls of sub-optimal goal states. This environment was used to further demonstrate empirically the theoretical proofs. The bottom image in Figure 4.2b illustrates 2d-RW of 15 by 15 where the agent starts at the square with the red circle and optimal goal is the right wall of green squares (all other green squares are sub-optimal).

1d-Random Walk (1d-RW)

The learning in 1d-RW environment was set up as follows: $\alpha = 0.4$, $\gamma = 0.9$, $\epsilon = 0.4$ (using ϵ-Greedy) with and without linear decay, the length of the corridor varied between 3 to 50 squares. The experiment was repeated for both neutral and negative step rewards. Each experiment was

repeated 10 times. Evaluation was conducted following purely the greedy action of the current learnt policy.

The theory presented in the previous chapter stated that during consecutive loops of an IRTC the Q-values of the state-action pairs of the IRTC would converge toward a common limit. This then could result in the agent's estimated optimal policy diverging. These experimental results show that if the agent switches between two or more IRTCs that have the same common limit that the same divergence happens. Thereby demonstrating that tiles can both induce IRTCs and these combinations of IRTCs with the same common limit can result in the agent's estimated optimal policy temporarily diverging away from the true optimal policy.

The Q-value of the tile in the different situations was analysed; this required looking at the values on a per run basis as averaging over multiple runs destroyed the information since the frequency of the phenomenon varies. We expect that the Q-value of the forward action to suddenly increase when the agent receives the goal reward and converge toward the common limit at all other times it is being updated.

Figure 4.3 is an arbitrary sample of Q-values where the phenomenon occurs; the y-axis donates the Q-value and the x-axis denotes the number of updates. As expected there are spikes upward and then a convergence curves to follow. The Q-values of the forward action has a jagged appearance. Each spike upward represents a time when the agent successfully found the goal state during learning. The downward curve afterwards represents agent moving within the tile. The Q-values of the backward action has a much smoother line; this is a consequence of the environment's only goal state being solely reachable by the forward action. The forward action is mostly above the backward action though there are periods where the two overlap; during these periods the agent is not retaining the previously learnt policy. The tile in Figure 4.3 is only 4 squares across which is creating an interesting dynamic; firstly the Q-values are above 1 because the tile is being exited so regularly and being rewarded by 1 point for doing so means that there is a convergence toward 10 (since $r = 1$ and $\gamma = 0.9$). Secondly, although the Q-value of the forward action drops after every spike it is only about every 50 updates that the Q-values start to overlap.

2d-Random Walk (2d-RW)

The 2d-RW environment was set up as follows: $\alpha = 0.4$, $\gamma = 0.9$, and $\epsilon = 0.4$ (using ϵ-Greedy) with and without linear decay. Experiments were repeated with and without non-optimal goals, and with neutral and negative step rewards. Each experiment was repeated 10 times.

2d-RW was used to further empirically demonstrate the application to TC by extending the problem to higher numbers of actions, state features, and goals. 2d-RW intends to model a general 2d tile within a larger tile placement. Each edge is lined with goal states but only one edge is the optimal set of goals states; this emulates one edge of tile being the correct edge to exit the tile on in the larger tile placement.

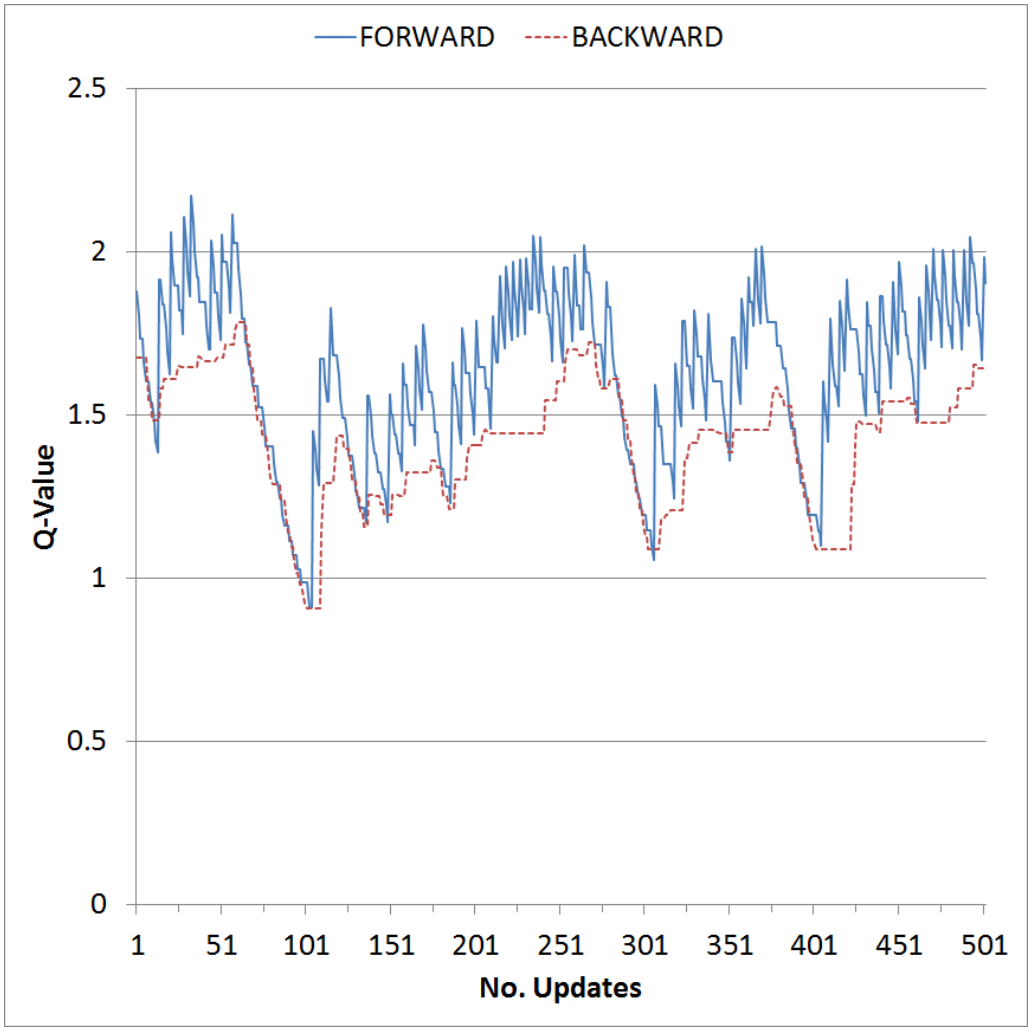


Figure 4.3: Example of Q-Value convergence in 1d-RW (Tile size 4, Neutral step reward, No epsilon decay, Every update over a 500 update period)

The results of 2d-RW were analysed in the same manor as that in 1d-RW: observing the per-update change in the Q-values. However in our figures we only show the optimal action's Q-values and another action's Q-values as to make the figure clearer and easier to interpret.

Figure 4.4 is an example of the changes over time of the Q-value of 2 of the 4 possible actions in 2d-RW. Due to the random chance of action selection this isn't an average of results rather an arbitrary example where the phenomenon occurs. Figure 4.4 is an example with sub-optimal goals and thus it can be seen that on occasion it is not the optimal action's Q-value that spikes upward rather a sub-optimal action's. There are periods of time where the actions Q-values overlap and, in contrast to the 1d-RW example, there are periods of time where the North action (a sub-optimal action) has a higher Q-value. The Q-values are between 0 and 1, this is

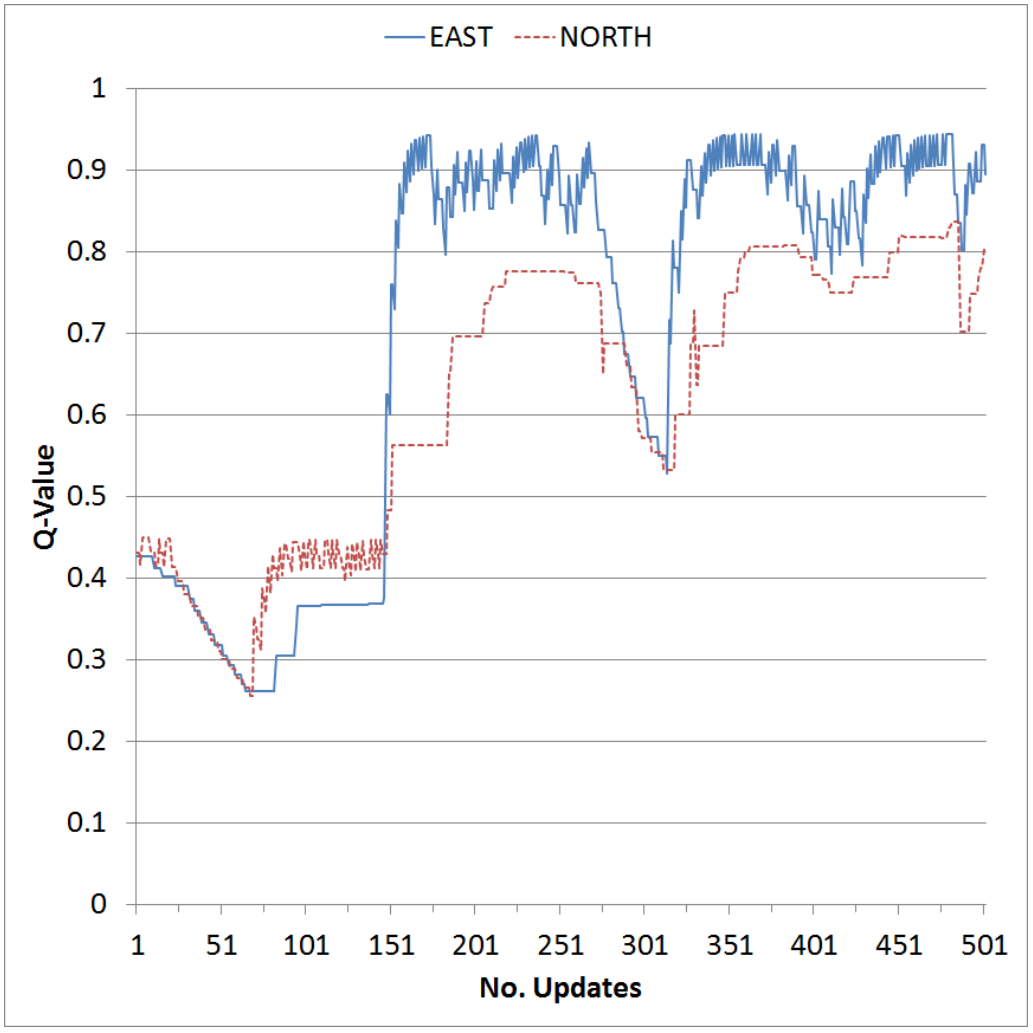


Figure 4.4: Example of Q-Value convergence in 2d-RW (Tile size 4x4, Neutral step reward, No epsilon decay, Every update over a 500 update period)

because the agent isn't exiting the tile regularly enough to create conflicting convergence series. Again, just as in the 1d-RW example, although after every spike there is a drop in value it is only sporadically that the periods of overlapping Q-values occur. These results further support our theory of the update dynamic that occurs within a tile.

Conclusions

The way function approximation methods like Tile Coding group states together means that they can cause IRTCs to appear even when in the MDP there is only a path with repeated actions. This means that TC can alter the MDP in a negative way which hinders stable learning. Furthermore, we can conclude that the larger a tile is the more likely it is to contain IRTCs and transition paths

with repeating actions. This means that large tiles are more likely to result in a learning agent performing consecutive loops through an IRTC and so more likely to have its policy change.

Finally, the theory presented in this chapter helps to explain further why the TC rule of thumb is a good rule. The theory gives another reason as to why larger tiles are result in worse performance. It is not only that a larger diversity of states can lead to an average which poorly reflects all states within but also that when there is no diversity of states-action-state rewards an agent can learn a random policy rather than one that reflects the environment.

4.2 Parameter Search

Our theory in the previous chapter has shown that in the specific case where there are IRTCs in an MDP, Q-Learning can undergo an unlearning phenomenon. In the previous section, we applied this theory to TC. The theory stated that there are 4 parameters that can impact the unlearning phenomenon, each of these were discussed theoretically. In the following set of experiments we aim to clearly identify the parameterisations where the unlearning phenomenon occur and which parameterisations are safe to use. We assume that these parameters are not independent of each other, however since there are 4 parameters we are limited to how we can present the results of these experiments. We aim to present the results in such a way as to highlight the nuances of the impacts the different parameters have on the unlearning phenomenon.

4.2.1 Experiment set up

We shall use a simple environment we shall call Left Right World, similar to the 1D-Random Walk environment. This environment is purposely designed to be a single IRTC which will allow us to view the impact of IRTCs as precisely as possible. Left Right World is constructed as follows: An agent is placed on one tile only. There are two possible actions to take: Left and Right. The agent receives a reward of 1 when $n_R - n_L \geq k$ where n_L and n_R are the number of Left and Right actions the agent has performed respectively, and k is an integer used to model the tile size. The theory states that the phenomenon depends on:

1. Learning Rate, α;
2. Discount Factor, γ;
3. Tile Size, k;
4. and, Exploration Factor, ϵ

Therefore, we will be varying all four of these parameters to discover how the unlearning phenomenon depends on these parameters. Specifically, the experiment will be set up as follows: The agent will initially be in a state where it has performed 0 left and 0 right actions; the agent will be allowed up to 1000 actions to learn, and will update its Q-Table after every action performed; if the agent successfully performs k more Right actions than Left then it receives a reward of 1

and returned to its initially environment state of 0 Left and 0 Right actions. The experiment will be run with all combinations of the following sets of parameters:

1. $\alpha \in 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
2. $\gamma \in 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
3. $\epsilon \in 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
4. $k \in 1, 2, 3, 4, 5$

The agent will learn using the Q-Learning algorithm and its Q-Table will be initialised pessimistically to 0 for every state-action pair. The agent will decide its next action using an ϵ-Greedy action selection policy. We will count the number of times after the agent has updated its Q-Table that the Q-value of the non-optimal Left action is greater than or equal to the optimal Right action. We will only start counting after the agent receives the positive reward of 1 for performing k more Right actions and Left actions. We will calculate the percentage of updates that result in the Left action's Q-value being greater than or equal to the Right actions Q-value, this will be refer to this as the "frequency of unlearning".

4.2.2 Results

Our results will be shown as heat maps. The temperature at any point in a graph will be the frequency of unlearning. The darker an area of the graph, the lower the temperature and therefore the lower the frequency of unlearning. The lighter an area of the graph, the higher the temperature and therefore the higher the frequency of unlearning.

In figures 4.5 and 4.6 we see a graph of graphs. In both of these figures the columns of graphs increment from left to right the tile size k; and, the rows of graphs increment from bottom to top the Learning Rate α and the Exploration Factor ϵ in figures 4.5 and 4.6 respectively. The individual graphs are heat maps; in both figures each of the x-axis in every graph increments left to right the Discount Factor γ; and, each of y-axis in every graph increments bottom to top the Exploration Factor ϵ and the Learning Rate α in figures 4.5 and 4.6 respectively.

The graphs have been smoothed using interpolation, by a factor of 10 in both axis.

The data represented in 4.5 shows that α has a higher impact on the frequency of unlearning than k. We can also see that ϵ has a larger impact than γ, though this difference is reduced as α is increased. Furthermore, γ is seen to be most impactful at low values of ϵ and middling values of k. Specifically, Figure 4.5 shows:

1. a direct correlation between α and the frequency of unlearning;
2. a minor, direct correlation between k and the frequency of unlearning;
3. a direct correlation between ϵ and the frequency of unlearning, which decreases in scale as α increases;

4. and, a minor, direct correlation between γ and the frequency of unlearning, which decreases in scale as α increases.

The data represented in 4.6 shows that at lower ϵ values k has a larger impact on the frequency of unlearning. Whereas at higher ϵ values k's impact is almost negligible on the frequency of unlearning. We also see that α has a higher impact at lower ϵ values and a rather smaller impact at higher ϵ values. Finally, γ seems to be most impact when ϵ, α, and k have low values. Specifically, Figure 4.6 shows:

1. a direct correlation between ϵ and the frequency of unlearning;
2. a minor, direct correlation between k and the frequency of unlearning at low values of ϵ;
3. a direct correlation between α and the frequency of unlearning, which decreases in scale as ϵ increases;
4. and, a minor, direct correlation between γ and the frequency of unlearning, which decreases in scale as ϵ increases.

4.2.3 Conclusion

We can conclude, therefore, that α and ϵ have a larger impact on the frequency of unlearning than γ and k. This means that the more random a selection policy an agent uses, the higher the frequency of unlearning will be since performing random actions will result in the agent spending longer within the bounds of a single tile. Spending longer within the bounds of a single tile increases the number of updates the Q-values will undergo whilst in an IRTC, thereby increasing the frequency of unlearning. Moreover, as α tends toward 1 the change in value of a Q-value increases; this means that every update of a Q-value whilst the agent is in an IRTC is larger. This also increases the frequency of unlearning.

Our results suggest that both α and ϵ should either be decreased over the time of learning, or always be low to minimise the impact of IRTCs in learning optimal policies of MDPs. However, at lower values of ϵ and α, k has a larger impact. Therefore, again the learning agent should either always have lower values of k or decrease them over the time of learning to reduce the unlearning phenomenon.

Furthermore, as γ tends toward 1 there is an increase in the frequency of unlearning. However, setting $\gamma = 1$ reduces the frequency of unlearning in most cases presented in our results; especially at low values of ϵ and α. Theoretically, when $\gamma = 1$ the limit that the Q-values tend toward is infinite, if the reward is non-zero. This is significantly different to instances where $\gamma < 1$ as the limit is bounded. This difference in limits suggests that, in general, $\gamma = 1$ is safer than $\gamma < 1$. However, altering the value of γ can result in changes to the optimal policies of an MDP and therefore is not so variable as the other parameters discussed.

The theoretical predictions from Chapter 3 state that according to Lemma 4, γ has a direct relation to the rate of convergence which should result in an inverse correlation to the frequency of unlearning and α has an inverse relation to the rate of convergence which should result in a direct correlation to the frequency of unlearning. This is not, of course, taking into account Lemma 3, which states that policy change can occur before convergence. What we see in these empirical results is that α indeed does have a direct correlation to the frequency of unlearning but γ has a minor *direct* correlation to the frequency of unlearning. This would suggest that the prediction from Lemma 3 is in effect and has a greater impact on γ than α over the predictions of Lemma 4. We also see that the tile size, k, does have a minor, direct correlation to the frequency of unlearning which also further supports Lemma 3.

These results support our theorems that, Theorem 1, there is a direct correlation between the probability of an agent performing consecutive loops through an IRTC and the policy changing, and that, Theorem 2, there are two ways to decrease the probability of the policy changing: decrease convergence rates and decreasing the probability that the agent performs consecutive loops through an IRTC.

Finally, these results are based on the agent only being able to learn a single tile and that agent starting an episode at an increasingly further distance from the goal as k increases. This could skew the results against k.

4.3 Impacts of Tile Shape

In chapter 3 Theorem 1 introduced a correlation between an agent performing consecutive transition cycles (Assumption 1) in a set of states with the same transition reward (Assumption 2) and the probability that an agent will learn the random or a random policy. This is not desirable behaviour for a learning agent. Theorem 2 stated that there were two ways to decrease the probability of an agent temporarily diverging from its estimated optimal policy:

1. increase the probability of the agent receiving a different transition reward from the IRTC reward; or

2. increase the probability of exiting the IRTC into a new state/action pair.

The former of these two options is not possible under the normal assumptions of RL. Increasing the probability of encountering different transition rewards would require knowledge of both the reward function and the transition function. The reward function is required to determine a state-action-state transition that yields a different reward to the IRTC. The transition function is required to determine the viable state-action-state transitions from a current state in the IRTC. Neither of these functions are available to an RL agent. Furthermore, if the agent did have access to the reward and transition functions then the problem can be solved perfectly using Dynamic Programming and an RL agent is not required. Another option is that the agent periodically alters

its exploration policy in the hope that by random chance it chooses a policy which increases the probability of receiving different transition rewards.

The latter option, increasing the probability of exiting the IRTC requires the RL agent to keep a history of its prior actions and rewards. The agent needs to inspect its history to determine whether it is in an IRTC, assuming the agent is then it needs to select an action $a \in \bar{A}$ where $\bar{A}$ is the subset of actions not performed in the state s in the IRTC. This method is not always possible since $\bar{A}$ could contain no members. Furthermore, this method could result in the agent entering another IRTC, then another and so on until the agent returns to the original IRTC; in this case, whether the identical rewards of each transition cycle remain identical then a larger IRTC is formed. However, when an agent is using TC, individual tiles can induce IRTCs where previously there was an identical reward path through the state space. This occurs when a tile contains a path through the state space with an identical action. In these cases of induced IRTCS the probability of exiting the IRTC can be increased by reducing the number of states contained within the tile; this is achieved by splitting the tile into 2 or more parts.

4.3.1 Shape of the split tile

Considering the situation where an agent has detected an IRTC that consists of one tile. The larger that tile is the higher the probability the agent will stay on that tile for more updates. This increase in probability of time spent on a single tile increases the probably that the unlearning phenomenon will occur, thus a higher probability that the agent's policy will temporarily diverge from its estimated optimal policy. Therefore the agent should, as discussed in the previous section, reduce the size of the tile. The agent is able to either reduce the size of the tile uniformly or non-uniformly along all features the tile encompasses. This renders the question: which of the two is most beneficial? To answer this question we must first define the significance of state features. A state feature is significant in an environment if it is used in the definition of a goal state. However, this definition is from the environment's point-of-view and not the agent's. A more apt definition would be to consider a state feature more significant if it is needed to distinguish states along the optimal path. The significance of a state feature could vary throughout the state space. In an area of the environment's state space where a subset of features are more significant than the rest, it would be intuitive to guess that non-uniform reduction biasing the more significant features would be more beneficial. The biased, non-uniform reduction would be achieved by only reducing the size of the tile along the significant features. Likewise, in an area of the environment's state space where no one feature is more significant than another, it would be intuitive to guess uniform reduction would be more beneficial. This leads to two more questions: *1*) can a learning agent reliably detect the significance of state features; and *2*) can we empirically demonstrate the validity of our intuition.

The first of the two questions, can a learning agent reliably detect which state features are more significant, is dependent on the confidence of the agent correctly identifying the optimal

path through the MDP. Once the agent has identified the path it believes to be optimal it can easily identify the state features required to distinguish the states along that path.

The second of the two questions, can we empirically demonstrate the validity of our intuition, can be answered through an empirical study. The empirical study will trial situations where the agent correctly identifies the significant feature, incorrectly identifies the significant feature, and assumes all have equal significance. By this we can inspect the frequency of correct identification required to gauge how confident an agent should be before biasing one subset of state features over another. 2-dimensional navigation domains are simple to understand and can easily be set up so that one or both features are significant. In our scenario we will assume our agent has one of three choices to reduce the number of states in a tile: uniform reduction in all features, see Figure 4.7a; or, non-uniform reduction favouring one feature over the other, see Figure 4.7b (here horizontal slats is the other option). We are deliberately keeping the area of the four new tiles being created the same for both uniform and non-uniform reductions. This ensures that we are addressing the problem of feature reduction and not generally reduction of the size of the tile.

4.3.2 Experiments & Results

4.3.2.1 2d-Random Walk (2d-RW)

2d-Random Walk (2d-RW) is a 2-dimensional environment where the agent must traverse from some starting point, usually the centre of the environment, to one of the goal states. There are four possible actions: *NORTH*, *EAST*, *SOUTH*, and *WEST*. The goal states are located around the outer perimeter of the environment. In Figure 4.8 the red circle is the starting point, any of the green squares are goal states, and the white squares are unreachable states. The agent receives a reward for reaching a goal state, all goal states along a particular edge will yield the same reward but the reward can vary between different edges. For example reaching any goal state to the East might yield a reward of 10 whereas any other goal state would yield 5.

Q-learning was set up as follows: $\alpha = 0.4$ and $\gamma = 0.9$, and ϵ-Greedy exploration was used with $\epsilon = 0.4$. Tile shapes used were limited to axis-aligned rectangles. Specifically, in each experiment all the tiles in a tiling had the same shape. We compared varying sizes of squares and oblongs (called *slats*) in various sizes of the environment. The reward for reaching different perimeter edges in the environment also varied. Experiments were set up so that there are unique optimal policies or multiple optimal policies. The average of 25 runs is recorded in the experiment results.

Figure 4.9 illustrates some of the different possible set-ups used in the empirical study. The bold, green edges indicate that the highest reward of 10 is given when the agent arrives at a state on that edge. When the agent arrives at any other edge it receives a reward of 5. The inner lines show the shape of the tiles used. The edges that yielded the highest reward were altered. For example, an instance of 2d-RW with a unique optimal policy could have the edge corresponding with the highest reward on any one of the North, East, South, or West perimeter edges.

2d-RW was used to empirically show the impact of tile shape on learning. 2d-RW can be viewed as modelling a single tile within a larger environment. Whereas in 2d-RW an episode ends when a perimeter state is reached, this would represent the agent leaving this tile in a larger environment.

4.3.2.2 Results of Tile Shape

Table 4.1 shows the mean average reward collected for the tile shape experiments. The results are separated into 4 different environment sizes: 03x03, 05x05, 09x09, and 11x11. Different tile shapes are evaluated in different instances of the 2d-RW environment, where the number of optimal policies is varied: unique, two adjacent, and two opposite (see Figure 4.9). We define three types of tiles: Square tiles, Parallel Tiles (slat tiles that lie parallel to the direction of movement toward the goal), and Perpendicular Tiles (slat tiles that lie perpendicular to the direction of movement toward the goal). It is worth noting in the cases where the goals are adjacent slat tiles are simultaneously parallel and perpendicular to the two goals and therefore we should expect similar performance. The rightmost column in Table 4.1 shows how frequently the optimal decision on alignment of slat tiles needs to be correctly guessed for slat tiles to outperform square tiles. The frequency, w, is calculated using the following expression:

$$w = \frac{MIN(r_{par}, r_{per}) - r_{sq}}{MIN(r_{par}, r_{per}) - MAX(r_{par}, r_{per})}$$

where $MIN(r_{par}, r_{per}) \neq MAX(r_{par}, r_{per})$, r_{sq} is the average reward of square tiles, and r_{per} is the average reward of the perpendicular slat tiles and r_{par} is the average reward of the parallel slat tiles and:

$$w = \frac{r_{sq}}{MAX(r_{par}, r_{per})}$$

where $MIN(r_{par}, r_{per}) = MAX(r_{par}, r_{per})$. In cases where $MIN(r_{par}, r_{per}) \geq r_{sq}$ the frequency calculation is not applicable since slat tiles perform at least as well as square tiles in the worst case.

The results of our experiments show that:

- when there is a unique optimal policy, using perpendicular slat tiles results in the quickest and most stable learning;
- when there are 2 optimal policies that are opposite to one another, using perpendicular slat tiles results in the quickest and most stable learning;
- when there are 2 optimal policies that are adjacent to one another, then no one tile shape consistently outperformed any other.

We can conclude therefore, when there is high certainty that all optimal policies have been correctly identified it is beneficial to use perpendicular slat tiles over square tiles. Whereas, when

03x03					
		Squares	Slats		w
			Perpendicular	Parallel	
Unique		9.999	**10**	5.072	100%
Dual	Adjacent	**10**	**10**	**10**	n/a
	Opposite	**10**	9.999	5.132	100%

05x05					
		Squares	Slats		w
			Perpendicular	Parallel	
Unique		9.015	**9.513**	5.010	89%
Dual	Adjacent	9.067	**9.493**	9.492	n/a
	Opposite	9.983	**9.998**	5.017	100%

09x09					
		Squares	Slats		w
			Perpendicular	Parallel	
Unique		8.704	**8.993**	5.014	93%
Dual	Adjacent	9.003	8.995	**9.010**	53%
	Opposite	9.883	**9.979**	5.031	98%

11x11					
		Squares	Slats		w
			Perpendicular	Parallel	
Unique		8.809	**8.983**	5.025	96%
Dual	Adjacent	**9.308**	9.186	9.262	160%
	Opposite	9.607	**9.917**	5.060	94%

Table 4.1: Average rewards of Tile Shapes on a range of environment sizes

there isn't high certainty then square tiles should be used instead.

Considering that automated tiling methods use estimated values to decide which tiles to split, it is unlikely they will have high enough certainty to make slat tiles beneficial. Therefore, it is

the conclusion of this study that automated tiling methods should split tiles evenly along each feature rather than giving priority to one feature over another.

This study has answered some questions but there are many still to be answered. Unfortunately these lie outside the scope of this body of work.

4.4 Heuristics

Heuristics can massively impact both the number of samples an agent requires to solve a problem and quality of the solution the agent produces. The impact of heuristics can be for good or for ill. Heuristics embody information about either a specific problem or problems in general. This information, when used by the agent, alters the way the problem is approached.

If we consider the set of all possible policies an agent can have for a particular problem then the application of a heuristic can do one of two things: *1*) reduce the size of the set of possible policies; or, *2*) encourage the agent toward a subset of possible policies. In the worst case some, if not all, good/optimal policies could be ruled out by the heuristic leaving the agent with a mediocre solution at best. The ideal heuristic would reduce the set of possible policies to only include the optimal solution. Good heuristics would reduce the set of possible policies to include fewer policies with bad performance.

4.4.1 Heuristics from theory

A heuristic that embodies information specific to a single problem limits its potential usefulness to that problem. A heuristic that embodies information specific to theory limits its potential usefulness to any problem that the theory applies to. In this way if we derive heuristics from the theory presented in chapter 3 it should be applicable to any problem that:

- has an agent that uses SARSA or Q-Learning's update rule;
- has an agent that uses Tile Coding; and
- has an environment which contains IRTCs or $\arg\max$ IRTCs.

For this reason we derive the following heuristics from the theory presented in chapter 3 as well as theory presented in chapter 2 to be used in the manual tile placement algorithm presented in the following section.

From Theorem 1 we know that IRTCs can lead to worse performance and therefore should be avoided. From Theorem 2 we know that decreasing the size of the tiles will decrease the inherent risk of IRTCs. Tiles along or near the currently held optimal path(s) need to be most reliable for two reasons: *1*) common exploration policies will spend most of their time on or near the currently estimated optimal policy; and, *2*) during evaluation a strictly greedy policy is often used and therefore errors on or near the currently estimated optimal path can result in poor performance. This leads us to the following heuristics that are based on theory and therefore applicable to any environment the theory applies to:

Heuristic. *Tiles that are in an IRTC should be reduced in size to improve reliability.*

Since performing consecutive loops through a longer IRTCs have a lower probability of occurring this heuristic aims to increase the length of a detected IRTC that spans multiple tiles.

Heuristic. *The more transition cycles or paths a tile contains the larger the risk of that tile becoming unreliable.*

Transition cycles or transition paths contained within a tile can induce short IRTCs in that same tile. In general the greater the number of states a tile contains the higher probability of those states having transition cycles or paths that can induce IRTCs.

Heuristic. *Tiles on or near the optimal path(s) should be reduced in size to improve quality and reliability.*

Since most agents use a measure of exploitation when learning states on or near the currently held optimal path will be visited more frequently than other states. Increased numbers of updates on these states increases the probability of the agent performing consecutive loops through IRTCs that may be contained therein. Therefore, these tiles should be the most reliable.

4.5 Heuristic-based Tile Placement

In this following section we will refer to the number of tilings used in Tile Coding, and the size and position of the tiles which comprise them as their *Tile Coding* (TC). Furthermore, we will refer to size and position of the tiles which comprise a TC as their *Tiling Placement* (TP).

The definition of the best TC for a problem can be specified as the TC that yields optimal performance whilst minimising the constraints that the TC must adhere to. In our case we consider one main constraint: the number of updates taken; and one subsidiary constraint: the real amount of time taken. However, in an unknown problem there is no way to verify the optimality of the solution or that the minimum number of updates have been taken without trialling all possible policies on all possible TCs. Such verification would be impractical at best and infeasible at worst. A practical approach would be to consider a set of TCs and choose: the TC that yields the best performance within a set number of updates, episodes, or real amount of time; the TC that yields a performance to some satisfactory level in the least number of updates, episodes, or real amount of time; the TC that balances best the minimisation of memory usage whilst yielding a satisfactory performance; etc. as the criteria for determining the best TC in the given set of TCs. Which criteria to use would be the choice of the stakeholders. We must also consider that if a fixed TC is used the number of trial-and-errors that will have occurred before selecting that TC. Furthermore, we must consider that if there is a tuneable parameter of a variable TC then the number of trial-and-errors that will have occurred fine-tuning the parameter(s). In both these cases we are considering the hidden cost of learning (Harutyunyan et al. 2015).

This leads to the question of determining the TCs to be compared. The TC can be fixed or variable. If the TC is variable then the initial TP of each tiling may not reflect their terminal TP. To our knowledge there has not been any work on variable TCs where the number of tilings changes automatically.

When choosing a set of fixed TCs to compare either the first TC can be chosen at random or be predetermined. The following TCs can be either chosen at random, be predetermined, or determined based on the history of performances. The upside to using fixed TCs is that once they have been initialised their only cost is a look up to see which tile in each tiling has been activated which is very predictable. The downside to using fixed TCs is that chance is a large factor in finding the TC that performs the best.

When choosing a set of variable TCs to compare either the algorithm used which makes it variable only has a set of parameters to alter, or it can have a set of parameters and have different initial number of tilings and TPs. The upside to using variable TCs is that they can reduce the size of the set of TCs needed to meet the criteria of the stakeholders. The downside to using variable TCs is its mechanism for varying the TC could limit the performance.

Another issue is that of the reliability of the performance during learning. As described by the theory in chapter 3 if the learning agent at any point gets caught in a consecutively repeating IRTC there is a chance that the policy on the tile(s) in the IRTC will become unreliable. This leads to desiring a TP algorithm that can reduce the risk of tiles becoming unreliable.

4.5.1 Mixed Resolution Acyclic Tiling

We demonstrate the effectiveness of the heuristics with a new TP method called Mixed Resolution Acyclic Tiling (MRAT). MRAT is based on the heuristics stated in section 4.4.1. MRAT can use one or more tilings. MRAT supposes that there is an optimal route through the environment and that it is known. Furthermore, MRAT supposes that the transition function is known. MRAT then places tiles of high resolution, small tiles, along this route and puts tiles of increasingly lower resolution, larger tiles, increasingly further away from the optimal route. The resolution is capped so that resolution does not become too low.

Since during learning an agent spends the majority time on or near its currently held optimal policy having high resolution near the optimal policy reduces the likelihood of the agent experiencing small IRTCs. Furthermore, high resolution tiles improves the quality of learning and reliability.

To analyse MRAT we use a simplified version of the Puddle World environment called Mud World. The Mud World environment is a non-trivial problem which allows for the clear creation of a single/multiple optimal route(s) and easy scaling of complexity by increasing the number of states. We will show that there is a significant increase in the speed of learning as compared to an agent without TC when the heuristic is used correctly (single, fixed TCs failed to learn in Mud World). Also that when applied incorrectly that such a tiling can entirely hinder learning; this

is an important result to have as it shows that particular TCs can both positively and negatively impact the performance of an agent.

4.5.2 Experiments & Results

4.5.2.1 Mud World Environment

The Mud World environment was devised to be able to quickly and easily create non-trivial environments with set numbers of optimal routes that can also be scaled in complexity. Mud World is a simplified version of Puddle World; Mud World has the same objective which is to traverse from the top-left to the bottom-right of a 2d grid. The Mud World grid consists of two types: Mud and Road, which define the transition reward for the agent. The transition reward to Mud is strictly less than the transition reward to Road (typically -2 and -1 respectively); this allows the environment to mostly consist of Mud except for the optimal route(s). In Figure 4.10 we see 3 examples types of the Mud World Environment: one optimal route; a few optimal routes; and, many optimal routes. Where the many optimal routes type of Mud World Environment consists of patch of mud which is $\frac{2}{3}$ of the total size of the environment.

4.5.2.2 Results

The Mud World environment was set up with the following parameters: $\alpha = 0.4$, $\gamma = 0.9$, and $\epsilon = 0.4$ (using ϵ-Greedy) with and without linear decay. Experiments were repeated with different single optimal routes, and different set-ups of one, a few, or many optimal routes. Each experiment was repeated 10 times and the agents were evaluated after set amounts of updates to the Q-values. We have purposefully deliberately excluded results comparing MRAT to uniform TC as it failed entirely to learn in Mud World.

Figures 4.11, 4.11, and 4.13 show the results of a Q-Learning agent with MRAT TP (labelled MRAT) vs an agent using Q-Learning without tile coding (labelled Q-Learning). The Mud World environment had one optimal route, a few optimal routes, and many optimal routes respectively. The x-axis shows the number of updates to the Q-values that have occurred. The y-axis shows the number of steps the agent took in the evaluation episode. In all 3 cases the figure shows, with statistical significance, MRAT learns the optimal policy faster than Q-Learning. We see that the difference is of a lesser magnitude in Figure 4.13. These results confirm that MRAT can be an effective method to speed up learning whilst without having to compromise the performance.

Figures 4.14 and 4.15 show the results of different MRAT TPs: MRAT All indicates that all of the optimal routes are in high resolution; MRAT Some indicates that a few of the optimal routes are in high resolution; and, MRAT One indicates that 1 of the optimal routes are in high resolution. The environments had a few optimal routes and many optimal routes respectively. The x-axis shows the number of updates to the Q-values that have occurred. The y-axis shows the number of steps the agent took in the evaluation episode. In Figure 4.14 MRAT One learns faster but is not as stable. Whereas MRAT All learns more slowly but never diverges from the optimal

policy once learnt. Since MRAT One is more approximate than MRAT All we should expect these results. In Figure 4.15 only MRAT All was able to learn the optimal policy. However, we see the same trend where the larger the approximation used the faster the agent reached its best learnt policy. These results show that as the number of possible optimal routes increases MRAT must also try to cover the majority of optimal routes with high resolution tiles.

Figures 4.16 and 4.17 show the results of different MRAT resolution steps. Either the MRAT TP had high-medium-low resolution tiles (labelled MRAT [H-M-L]) or high-low resolution tiles (labelled MRAT [H-L]). The low resolution tiles were double the size of medium resolution tiles which were double the size of high resolution tiles. The environments had one optimal route or a few optimal routes respectively. The x-axis shows the number of updates to the Q-values that have occurred. The y-axis shows the number of steps the agent took in the evaluation episode. In Figure 4.16 we see both MRAT TPs have statistically similar performance however MRAT [H-M-L] is more stable. In Figure 4.17 we see that both TPs have statistically the same performance. These results show that there are minor gains to be had by using gradual change from high resolution to low resolution tiles over sudden change.

4.6 Conclusions

This chapter introduced new heuristics for tile size TP. These heuristics where applied in the TP method called Mixed Resolution Tiling (MRAT). MRAT TP must be provided to the agent before learning. An MRAT TP specifies that tiles on the optimal route should have a high resolution, and tiles can safely be larger the further they are from the optimal policies route. MRAT also states that there should be an upper bound on the size of a tile. Initial results into the performance of MRAT were shown in the Mud World environment. Using MRAT can speed up learning without sacrificing performance. However, for best performance MRAT must cover the majority of optimal routes in high resolution tiles. Furthermore, there are minor stability of learning gains to be had by using gradual change between high and low resolution tiles.

Research is needed to discover if MRAT can be in a partial state between a more general uniform tiling and an MRAT. Additionally research is needed to know whether MRAT can be automated and if an automated version would be beneficial. These two questions will be covered in the next chapter where we automate MRAT.

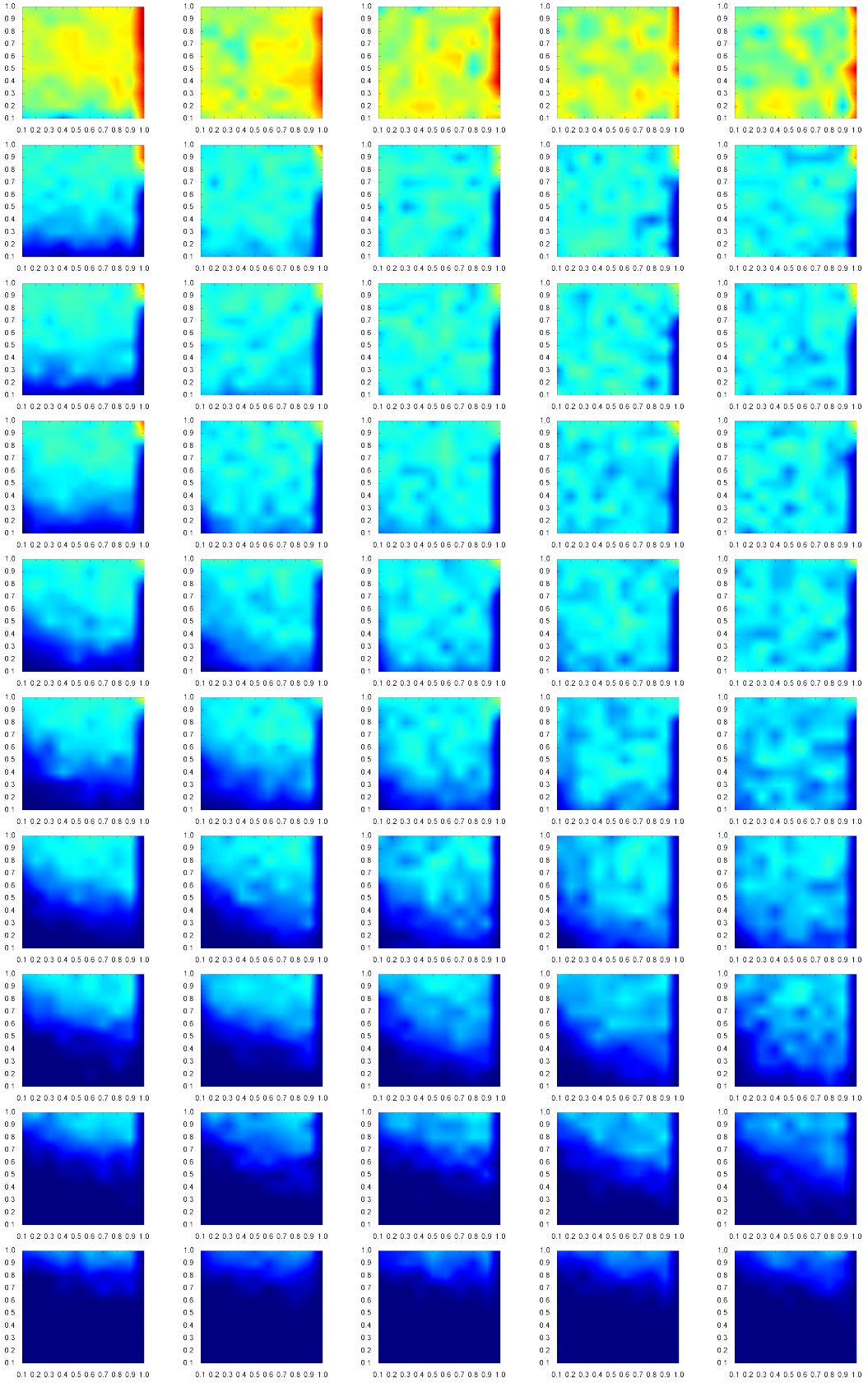


Figure 4.5: Frequency of Unlearning Heat Map, k (Tile Size) x α (Learning Rate)
k increases from left to right per graph, α increases from bottom to top per graph
γ increases along the x-axis in each graph, ϵ increases along the y-axis in each graph

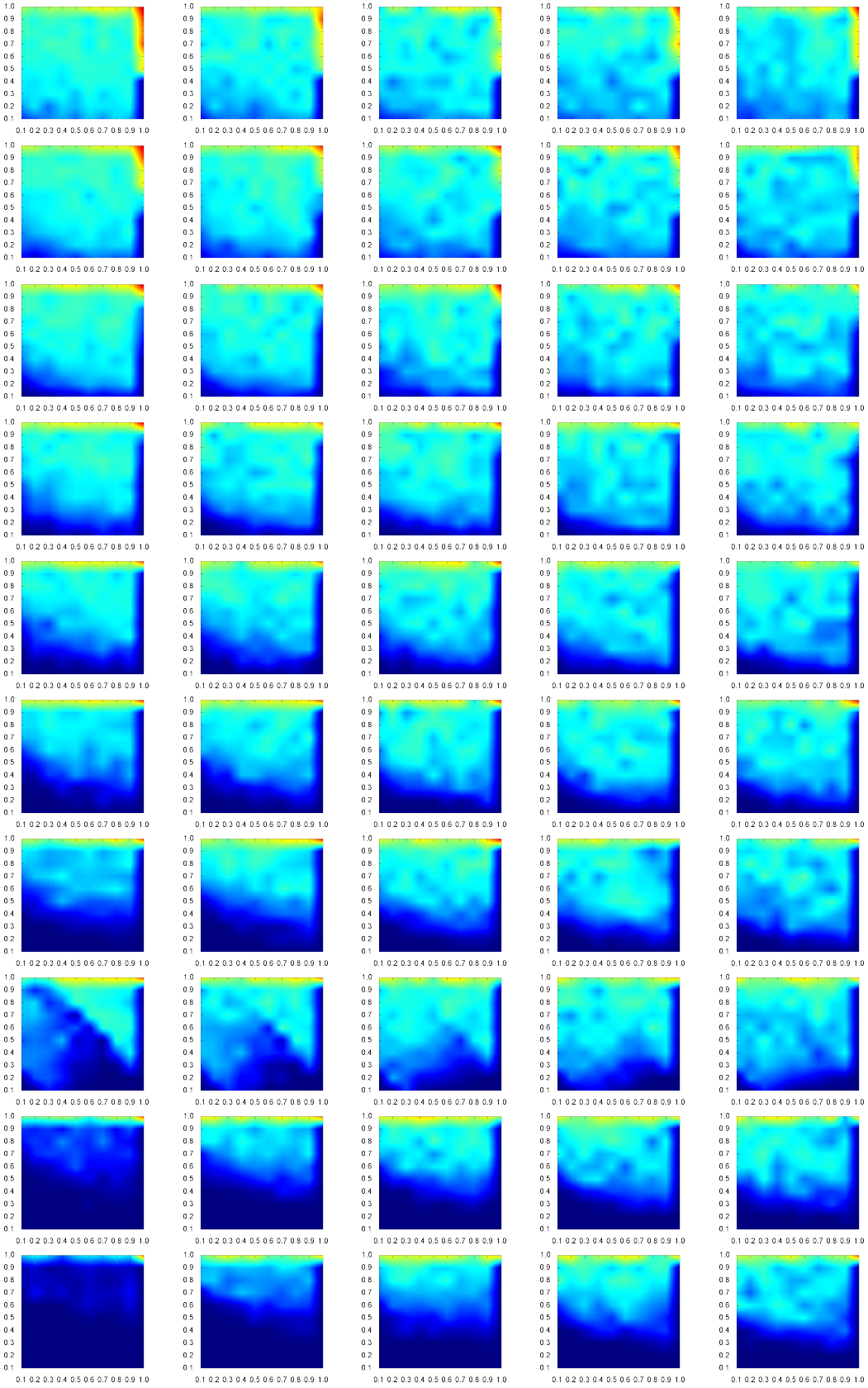
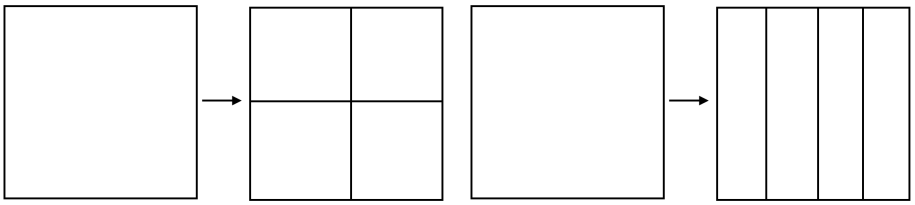


Figure 4.6: Frequency of Unlearning Heat Map, k (Tile Size) x ϵ (Exploration Factor)
k increases from left to right per graph, ϵ increases from bottom to top per graph
γ increases along the x-axis in each graph, α increases along the y-axis in each graph



(a) A uniform split of a tile (b) A vertical slats split of a tile

Figure 4.7: Examples of different ways to split tiles

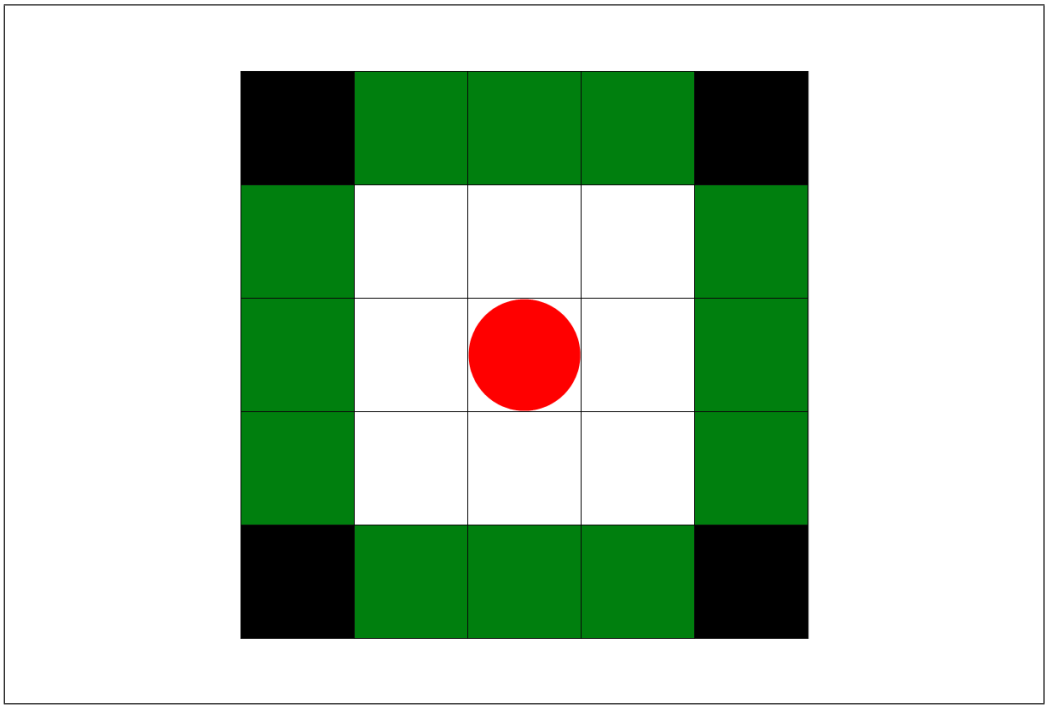
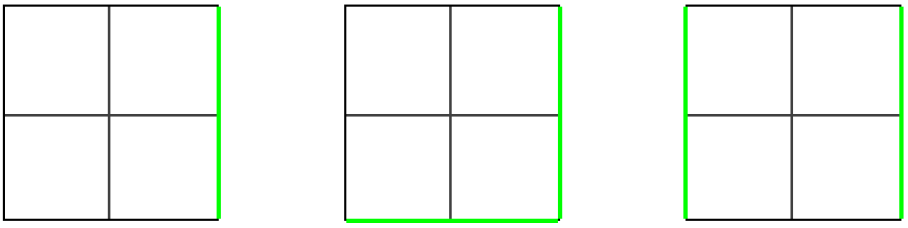
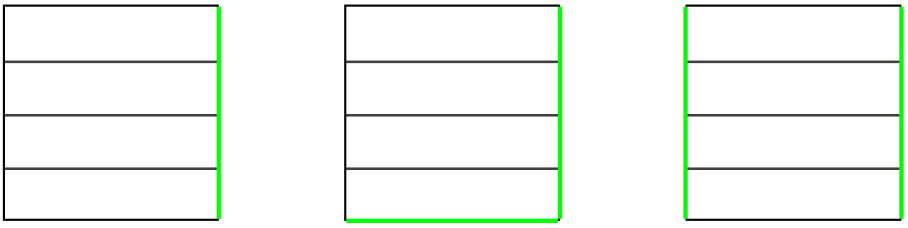


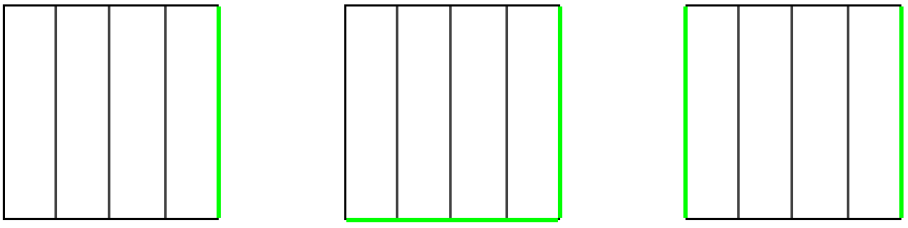
Figure 4.8: A 2d-Random Walk Environment (Agents moves from the red circle to a green square)



(a) Square tiles with unique, dual adjacent, and dual opposite optimal policies respectively

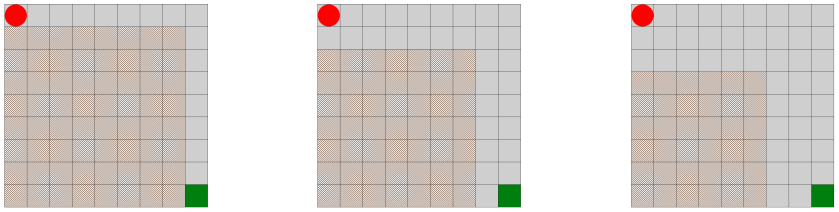


(b) Parallel slat tiles with unique, dual adjacent, and dual opposite optimal policies respectively



(c) Perpendicular slat tiles with unique, dual adjacent, and dual opposite optimal policies respectively

Figure 4.9: Examples of different experiment set-ups



(a) 1 optimal route (b) A few optimal routes (c) Many optimal routes

Figure 4.10: An example Mud World Environment (Agents move from the red circle to the green square, brown squares are mud, grey squares are road)

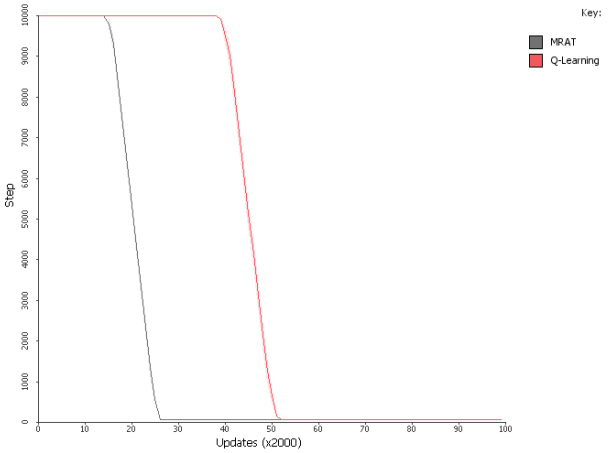


Figure 4.11: Mud World with one optimal route. Q-Learning vs MRAT. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

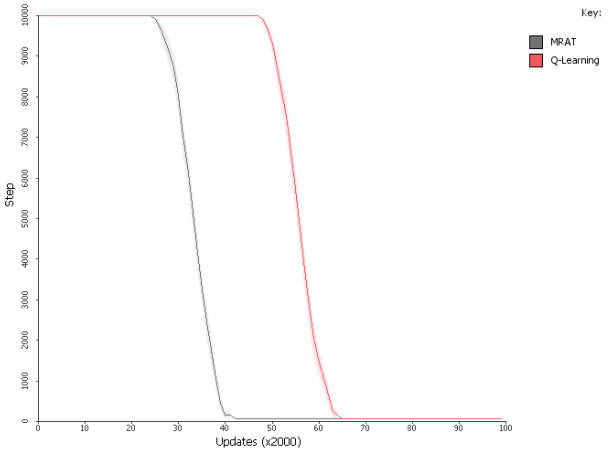


Figure 4.12: Mud World with a few optimal routes. Q-Learning vs MRAT. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

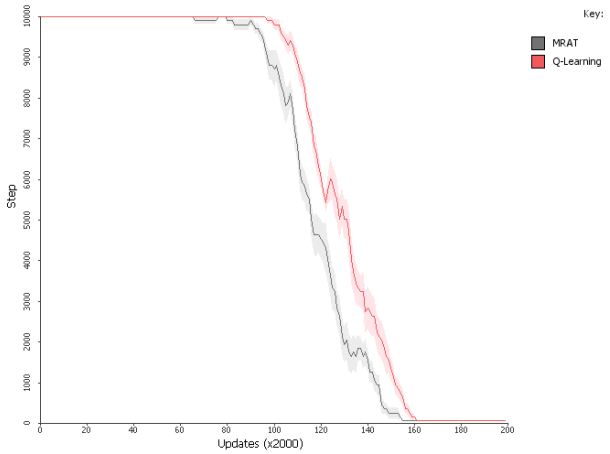


Figure 4.13: Mud World with many optimal routes. Q-Learning vs MRAT. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

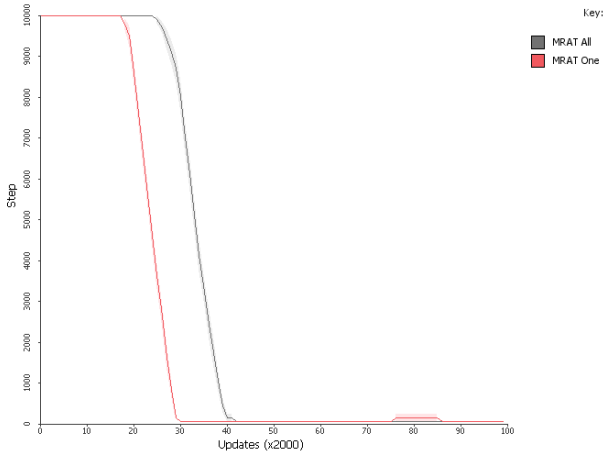


Figure 4.14: Mud World with a few optimal routes. MRAT with all optimal routes in high resolution vs 1 optimal route in high resolution. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

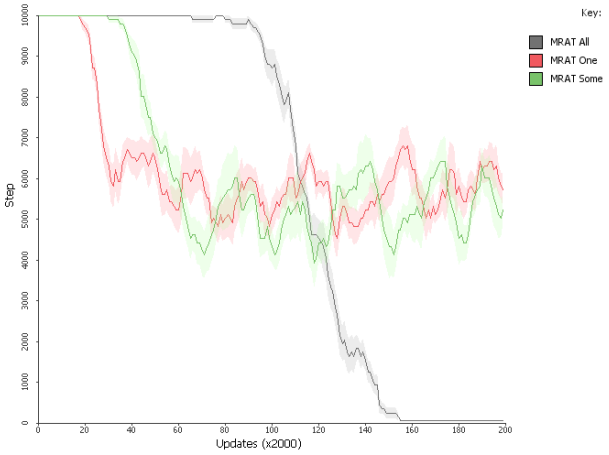


Figure 4.15: Mud World with many optimal routes. MRAT with all optimal routes in high resolution vs a few of the optimal routes in high resolution vs 1 optimal route in high resolution. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

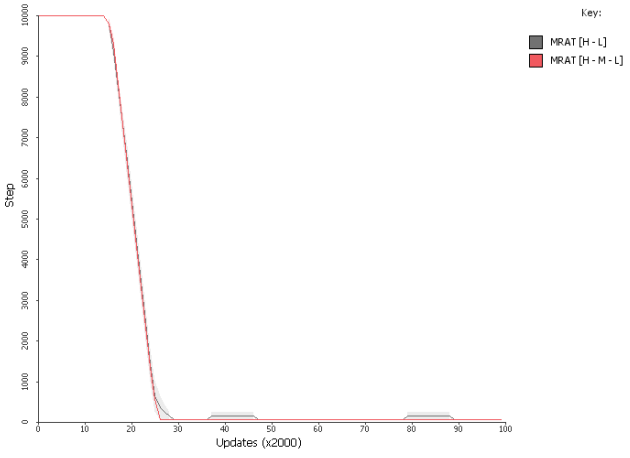


Figure 4.16: Mud World with one optimal route. MRAT with high-medium-low resolutions vs MRAT with high-low resolutions. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

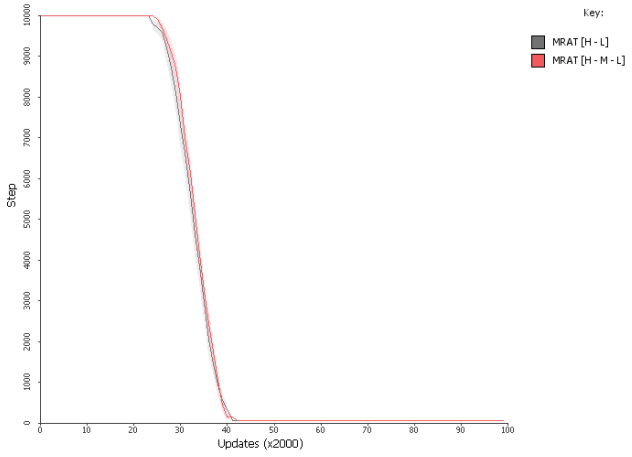


Figure 4.17: Mud World with a few optimal routes. MRAT with high-medium-low resolutions vs MRAT with high-low resolutions. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

CHAPTER 5

Automation of MRAT

This chapter introduces a novel algorithm called Automated Mixed Resolution Acyclic Tiling (AMRAT). AMRAT is an automated version of the manual tile placement strategy Mixed Resolution Acyclic Tiling introduced in chapter 4. MRAT itself was based on heuristics derived from theory introduced in chapter 3 and their application in chapter 4. AMRAT alters the tile code representation of an environment during learning.

This chapter begins by motivating the need for automated tile placement algorithms. This is followed by a discussion of how to automate MRAT, going into depth about the possible implementations. Then an empirical study is conducted to rate the effectiveness of each implementation choice. Next, the AMRAT algorithm is finalised and formally defined. This chapter concludes by comparing AMRAT to its state-of-the-art competitors.

5.1 Motivation

Tile Coding is an effective and efficient means of allowing RL agents to learn in environments that would be otherwise infeasible to do so. TC achieves this by approximating the environment by grouping states exhaustively into tiles. Chapter 3 introduced theory regarding SARSA learning agents in environments with Identical Reward Transition Cycles (IRTCs), the theory states that an agent can temporarily diverge from a learnt policy whilst looping through IRTCs. [1] This means then, where possible, looping through IRTCs should be avoided. However, in RL the transition function is assumed to not be known. This means then a tile placement algorithm that avoids IRTCs must be used in conjunction with learning the environment. This then motivates us

[1] This also applies to Q-Learning agents where $\arg\max_a$ IRTCs exist.

to create an automated version of MRAT so that the transition function is not needed to be known beforehand, rather the agent can apply its current learnt knowledge to the tile placement.

Furthermore, the layout of tiles, their size, shape, and position, have a large impact on the performance of learning; like heuristics, these impacts can be positive or negative: either improving the rate or quality of learning, or, in contrast, impeding. As empirically demonstrated, domain knowledge is required for anything other than uniform tile placement to be used effectively without chance. The domain knowledge can come from a human designing the tiling for a specific domain. This is limiting as the human designer would need expert knowledge of both the domain and tile coding. Furthermore, in situations where the human designer was either an expert of tile coding or of the domain they would be left to trial-and-error, repeatedly experimenting with different configurations until a satisfactory performance was reached. This procedure is lengthy and often infeasible. Nevertheless, the domain knowledge can also by automatically acquired by the agent learning in the domain. This has been shown to be very effective (Whiteson et al. 2007; Nouri and Littman 2009b). In these cases as the learner gains domain knowledge it selects specific tiles to split. These methods gain the added benefit of starting very generalised and progressively becoming more specialised (Munos and Moore 2002; Sherstov and Stone 2005).

5.2 Automation of MRAT

In the following subsection we will explain, discuss, and analyse Automated Mixed Resolution Acyclic Tiling (AMRAT).

Mixed Resolution Acyclic Tiling (MRAT) was introduced in the previous chapter as a manual method of devising tile placement which consists of tiles of heterogeneous size. Specifically, MRAT uses knowledge of optimal transition path(s) through the state space of an MDP; tiles with “high resolution”, small tiles, should be used on or near an optimal path, “low resolution”, large tiles, should be used away from the optimal paths, and that the resolution of the tiles should be tapered in between. Furthermore, tiles that constitute a high risk of inducing IRTCs should also be split. MRAT was based on the heuristics derived from theoretical properties of TC introduced in chapter 3.

One of the main goals in automating MRAT would be to remove the need for the tiles to be placed manually. MRAT requires domain knowledge to discern where the different resolution tiles should be placed. However, we assume that an agent initially has no knowledge of the domain, therefore AMRAT must construct the tiling whilst learning about the environment. However, without careful consideration of when AMRAT should alter the TC, the results could be vastly different tile placements and therefore vastly different performances depending on how the agent explored the environment.

5.2.1 Implementation Options

There are, of course, many different ways that AMRAT could be implemented but there are two components that it must contain: 1) a mechanism for detecting tiles that induce IRTCs; and, 2) a mechanism for detecting the currently held optimal path through the MDP. It would be ideal if these two mechanisms could work cooperatively rather than one negatively impacting the effectiveness of the other and vice-versa. Both of these mechanisms require a means of determining which tile(s) to split and when they should be split.

Detecting tiles that induce IRTCs

One straightforward method of detecting tiles that induce IRTCs is to detect all IRTCs the agent passes through during learning. This would be done by storing a history of the agent's previous tile-action pairs and the transition reward then on each new transition the agent can either erase its history if the transition reward changed or look back at its history to determine whether it is in an IRTC. This would be the most inclusive method and the worst-case time complexity would scale linearly with respect to the maximum number of steps allowed per episode; however, if the agent regularly receives different rewards or regularly revisits tiles then the average-case time complexity would be much better as the agent's history would be short.

Another, more simplistic method would be to only detect IRTCs of length 1. Tiles can induce IRTCs of length 1 if a repeated action forms a path through the states within that tile. This would be achieved by taking note when an agent performs an action and remains on the same tile whilst receiving the same reward. This is the least encompassing method but its worst-case time complexity is constant time as the history the agent is looking back at is a fixed length.

The final method would be to detect IRTCs of length $\leq n$ where n is either a parameter set by the user or variable on some metric relative to exploration and/or the environment. This method would have a time complexity of $O(n)$ where n may be variable if it is a metric relative to exploration and/or the environment. The time complexity is $O(n)$ as the agent would need to look back at most n steps into its history to determine whether it was in an IRTC. However, this possibility of implementation will not be considered further for the following reasons:

1. the theory stated that induced IRTC would be short;
2. we wish to reduce the number of user set parameters; and,
3. there is no obvious metric to create a variable from.

Detecting the, currently held, optimal path

One straightforward method of detecting the optimal path would be to keep track of the path the agent uses whilst learning to reach a goal state. However, if the agent is performing exploration actions then this sub-optimality will be introduced into the path. The time complexity for this

is constant and the memory complexity scales linearly with respect to the maximum number of steps per episode.

Another, more complex method would be to simultaneously learn a transition model of the TC. Then a weighted path-finding algorithm could be applied to find the currently held optimal path. The weights would be the normalised difference in value of the tiles. This would provide a more accurate optimal path, dependent on the path-finding algorithm used, but would incur the cost of running the path-finding algorithm each time it was run. The time complexity for learning the transition model is constant time whilst the memory complexity scales $1:1$ with the number of tiles. However, the time and memory complexities for the weighted path-finding algorithm would also need to be taken into consideration. The path-finding algorithm could be a weighted, shortest path algorithm such as A^*[2] or the current greedy policy. A^* has a worst-case runtime complexity of $O(|E|)$ where E is the set of edges in the transition model. The current greedy policy has a worst-case runtime complexity of $O(|V|)$ where V is the set of vertices in the transition model. Due to the likelihood that the path-finding algorithm will be used frequently A^* will not be used in any of our implementations. If using the greedy policy there would need to be an implementation choice of whether to begin the search from the initial tile the agent started the episode on or the tile that contained the goal state the agent reached.

Deciding how to rank tiles to split

Now that we have a number of different methods of implementing the two detection mechanisms we need to consider how to priority rank the tiles the two mechanisms flag to be split.

The two mechanisms could either have two independent ranking systems, a single unified ranking system, or a trigger so that once a certain event has occurred the algorithm switches from one to the other.

The first option would require either a comparator to determine when one ranking would be prioritised over the other or risk that the two mechanisms could conflict with one another. The second option is equivalent to having two independent ranking systems with a comparator. The conflicts would occur if splitting a tile would alter the other ranking system. The IRTC detection mechanism could be altered by the optimal path detection mechanism if one or more of the ranked IRTCs contained tiles on the optimal path. The optimal path detection mechanism could be altered by the IRTC detection mechanism by the fact that splitting a tile will alter the transition model and therefore potentially the optimal path through the transition model. The final option, a trigger would ensure that neither mechanism interfered with the other. An obvious choice of trigger would be when a goal state is first reached.

Deciding when to split tiles

The next part of implementing AMRAT is determining when the most eligible tile should be split. Commonly an algorithm requires a user defined parameter to be used as a threshold limit

[2]Please refer to Hart et al. (1968) for more information on the A^* path-finding algorithm.

and once that limit is reached then it would be deemed appropriate to, in this case, split a tile. Ideally the metric used with the threshold would not need to be fine-tuned for each environment, or class of environment the algorithm was applied to. As Harutyunyan et al. (2015) discuss in their paper, some algorithms can appear very effective but there are large amounts of learning hidden through the parameter tuning applied to that algorithm in each environment it is used. The requirement of tuning for good performance can defeat the benefit from using one parametrised algorithm over another such as pure TC where trial and error is used to determine the number of tiles and tilings. To this end it would be useful if the metric was generic in some way as to apply to all environments without the need for any fine-tuning.

A good example of such a metric that applies to all environments is used by Whiteson et al. (2007). They track the *Bellman error* (i.e., ΔV). Their premise is that whilst V is improving $|\Delta V|$ will tend to decrease over time. They argue that if a certain period of time has passed since $|\Delta V|$ has decreased then learning has likely converged. Therefore, if learning has converged then it is an appropriate time to split a tile and allow the algorithm to further improve its knowledge. They note however, that "*this quantity is extremely noisy, since updates to different tiles may differ greatly in magnitude and updates to different state within a single tile can move the value estimates in different directions*". For this reason in their algorithm they use a heuristic which for each tile tracks the lowest $|\Delta V|$ occurring in updates on that tile. It also uses a global counter that increments when no new lowest $|\Delta V|$ is found on a tile and resets when a new lowest $|\Delta V|$ is found on a tile. If the global counter reaches a threshold then it is deemed time to split a tile. This was an effect method in their implementation as the algorithm was randomly given sample states from the environment on which to act and learn. However, in an implementation where the algorithm learns through enacting episodes this method will not be as effective (as shown in later empirical results). This is because of induced IRTCs causing Q-values to converge toward the common limit of that IRTC and therefore learning is not deemed to plateau in regions of the state space where there are identical transition rewards. They state that one source of noise comes from value estimates moving in different direction; since tiles can induce IRTCs and the convergence value of an IRTC is often different to the true value, this method may be too noisy to be effective. The principle of using whether learning has plateaued as a metric for when to split a tile is sound.

Since AMRAT requires a mechanism for estimating the optimal path from an initial state to a goal state, a good point to evaluate what the optimal path through the transition model is when a goal state is found; therefore, when a goal state is found could also be used as a point to split tiles. This has the disadvantage that if the goal states are rare or hard to achieve then the algorithm will not split tiles often and therefore could struggle to learn. Furthermore, this method would only work with detecting the estimated optimal path and not detecting IRTCs.

AMRAT is already ranking the tiles in terms of eligibility for being split. A threshold metric could be a set limit of the ranking. This has the advantage of being easy to compute and check

but, depending on the ranking system, may need to be tuned for each environment AMRAT is used in.

Another threshold metric could be to count the number of updates that have occurred on any tile. This would be very easy to implement but its effectiveness would be mostly reliant on chance. Therefore we shall not consider this implementation method any further.

How many tiles to split

The final part of implementing AMRAT is the decision of whether a single tile should be split or a group of tiles should be split simultaneously. This is closely linked to deciding when to split a tile. As Whiteson et al. (2007) state in a footnote "*The agent splits only one tile at a time. It could split multiple tiles but doing so would be similar to simply reducing [the threshold]*". This holds true in their case since the criterion used to order tiles to be split is independent to the metric for determining when tiles should be split. If the threshold metric was linked to the ranking system the question becomes complicated. On one hand it could simply be once a tile has reached the threshold split the top n most eligible tiles. On the other hand it could become a question of when to split tiles either: 1. how many tiles need to reach the threshold before splitting tiles; or, 2. once a tile has reached one threshold all tiles above another threshold are split.

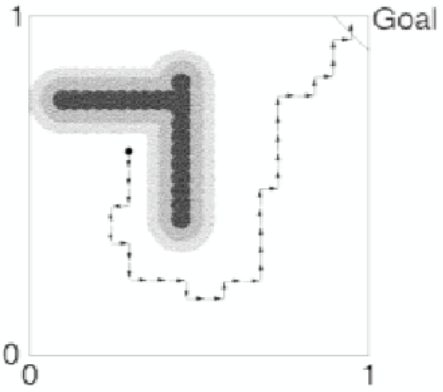
5.2.2 Evaluation of the implementations

We have discussed the possible implementation choices we have available to use in the previous section. Now we will conduct an empirical evaluation on the different components and their implementations. This allows us to make an informed decision on the design on AMRAT.

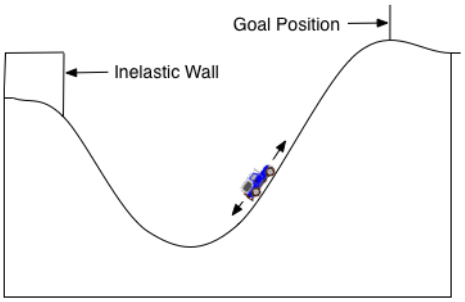
We will begin by individually evaluating the possible implementations and parameter settings of the two components AMRAT must contain: 1) a mechanism for detecting tiles that induce IRTCs; and, 2) a mechanism for detecting the currently held optimal path through the MDP. Analysing those results will allow us to determine whether they are both required or not. Assuming they are, we will empirically evaluate the different implementations of combining the two mechanisms. We will conduct our empirical evaluations in Puddle World and Mountain Car.

Puddle World

Puddle World is a continuous 2d navigation environment where an agent starts in the north-west and must travel to the south-east avoiding "puddles". There are four available actions: *NORTH*, *EAST*, *SOUTH*, and *WEST*. Although the movement distance is fixed the environment is continuous as Gaussian-random noise is applied to the agent's movement. There is a negative reward for each action the agent takes and it gets a reward of 0 for reaching the goal area. The agent also receives an increasingly large negative reward for being in a puddle the closer to the centre of that puddle the agent is. There are two state features: the x and y coordinate of the agent.



(a) A Puddle World Environment (Agents move toward the goal avoiding the puddles)



(b) A Mountain Car Environment (Agents escape the valley)

Mountain Car

Mountain Car is a continuous environment where an agent must control a vehicle stuck in a valley through forward, backward, and neutral actions. The objective is to arrive at the far right peak but this can only done if enough momentum is gained from coming down the left-hand side of the valley. The agent is accelerated toward the bottom of the valley by gravity. The agent receives a reward of -1 for every action taken and 0 for reaching the goal. There are two state features: the location and velocity of the car.

Detecting IRTCs

There are 4 possible ways to combine the implementation options discussed in the previous section:

1. No limit on IRTC length with a consecutive loop threshold to decide when to split tiles (*Unlimited Consecutive*).

2. No limit on IRTC length with plateaued learning to decide when to split tiles (*Unlimited Plateaued*).

3. IRTC length 1 with a consecutive loop threshold to decide when to split tiles (*Fixed Consecutive*).

4. IRTC length 1 with plateaued learning to decide when to split tiles (*Fixed Plateaued*).

For the threshold we will keep a counter on each tile. The counter is incremented by 1 for every consecutive loop through the current IRTC it has done. The counter is reset when the agent enters the tile again after it has previously stopped consecutive loops. For the plateau of learning we will use the same implementation used by Whiteson et al. (2007).

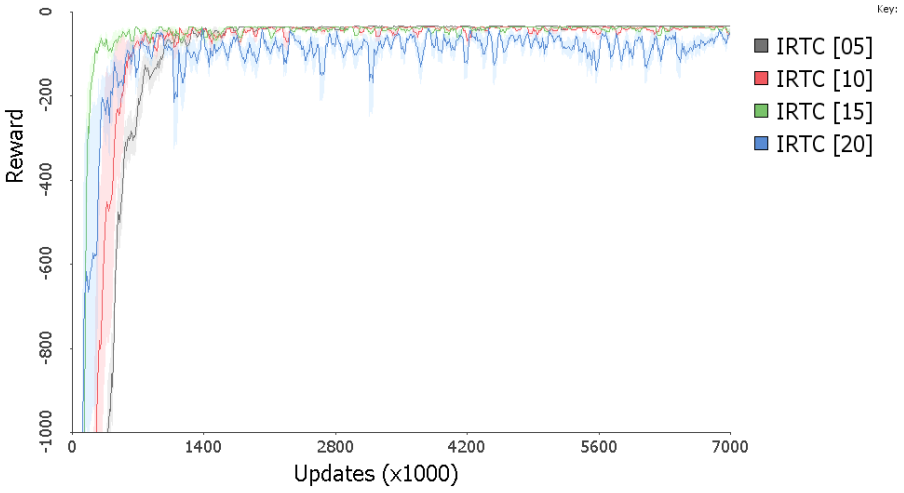


Figure 5.2: Puddle World. Different parameter settings for Unlimited Consecutive. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

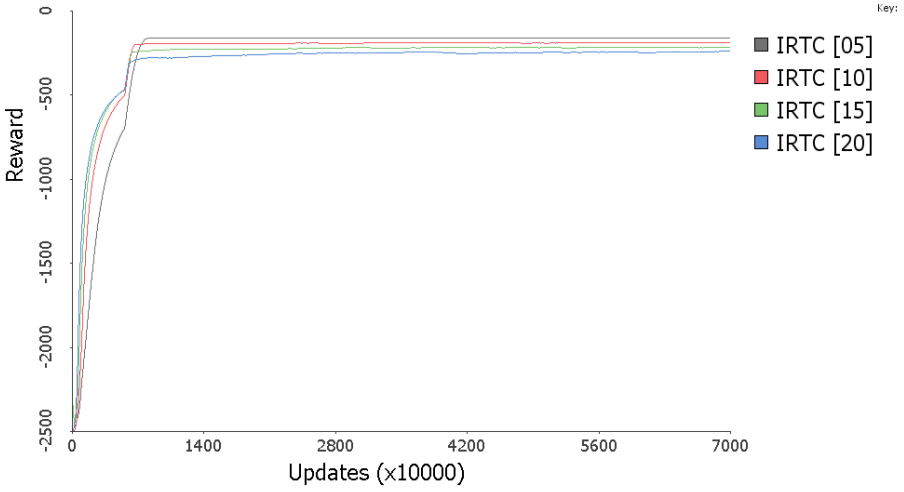


Figure 5.3: Mountain Car. Different parameter settings for Unlimited Consecutive. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

In Figures 5.2 and 5.3 we see the results of Unlimited Consecutive. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. We see that in both Puddle World and Mountain Car that the parameter settings have a similar impact on the quality of the final policy. We see that a threshold value of 5 is best down to a threshold value of 20 being the worst. We also see a rough inverse of this for the speed of learning. This result is as expected as the lower the threshold the faster tiles will

be split and also the smaller the final tiles will be. This matches the general rule of tile coding, smaller tiles learning more slowly but more accurately.

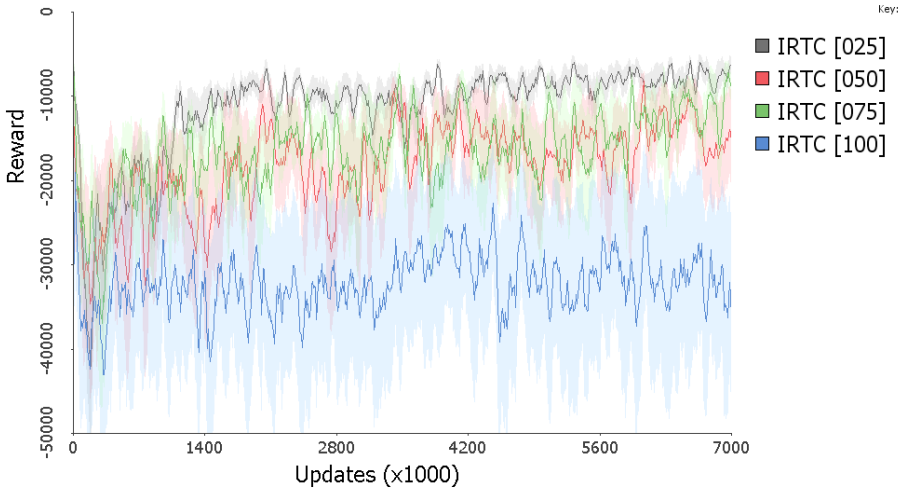


Figure 5.4: Puddle World. Different parameter settings for Unlimited Plateaued.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

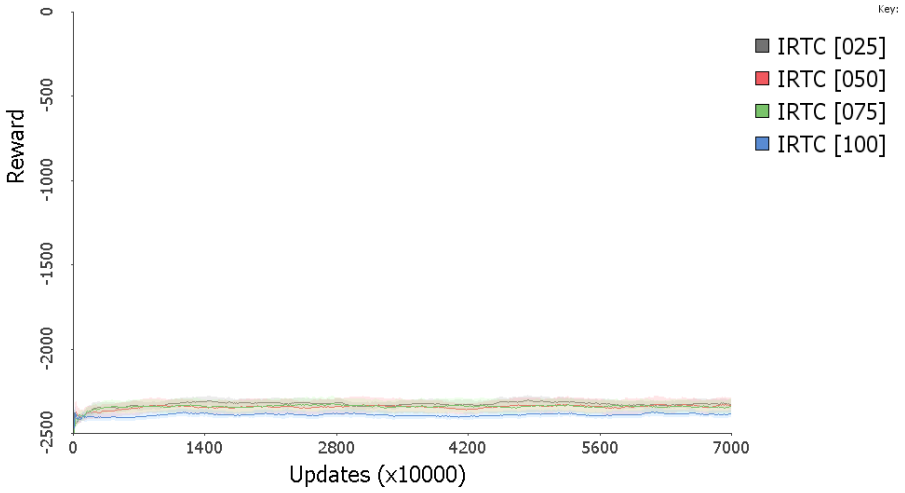


Figure 5.5: Mountain Car. Different parameter settings for Unlimited Plateaued.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

In Figures 5.4 and 5.5 we see the results of Unlimited Plateaued. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. We see that in both Puddle World and Mountain Car the parameter settings

have a similar impact on the quality of the final policy. We see roughly that a threshold value of 25 performs better than a threshold value of 50 and so on. However, we see that Unlimited Plateaued is significantly less effective compared to the results of Unlimited Consecutive (see Figures 5.2 and 5.3). This is due to the agent learning through episode roll-outs rather than random samples from the state space. The initial tiles are large and environments mostly give the agent the same transition reward. This results in the Q-values of the large tiles converging toward a common limit, as per our theory of IRTCs, and therefore the agent is almost always perceived to be learning and so tiles are not split effectively.

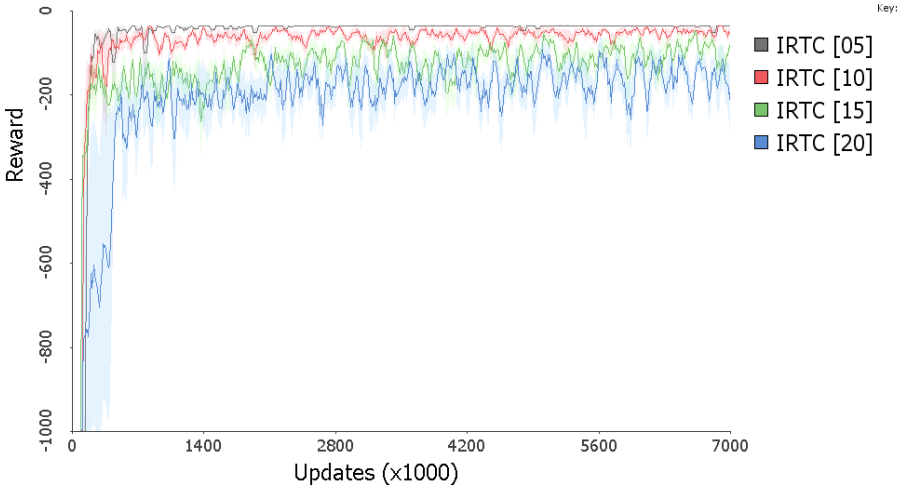


Figure 5.6: Puddle World. Different parameter settings for Fixed Consecutive.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

In Figures 5.6 and 5.7 we see the results of Fixed Consecutive. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. These results are similar to those of Figures 5.2 and 5.3 however the agent does not perform as well in either environment with the limitation of length of the IRTC being detected. This is especially apparent in the Mountain Car environment. Since Mountain Car models a car's motion and the actions are accelerate, decelerate, or coast the agent is more likely to move between tiles and therefore the IRTCs it faces in Mountain Car are longer than that of Puddle World.

In Figures 5.8 and 5.9 we see the results of Fixed Plateaued. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. Interestingly we see a different reaction to changing from unlimited IRTC length to IRTC length 1 to that of the other threshold. The agent's performance roughly stays the same. The

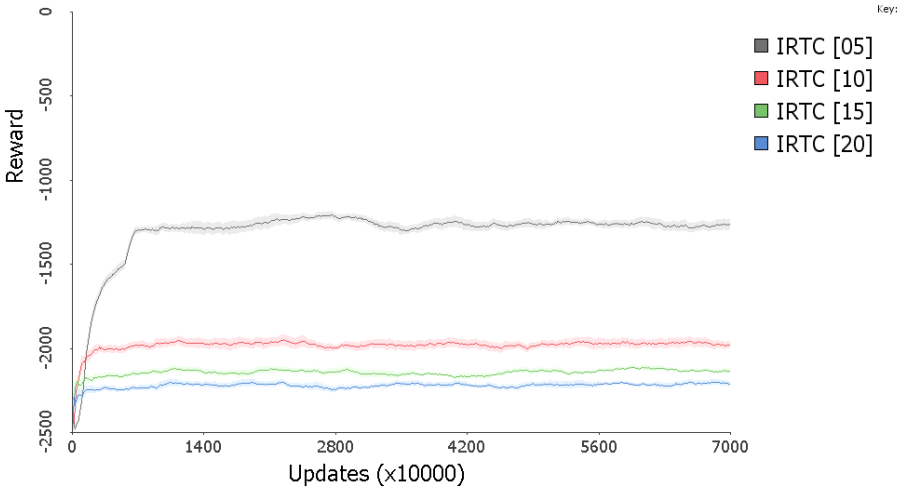


Figure 5.7: Mountain Car. Different parameter settings for Fixed Consecutive.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

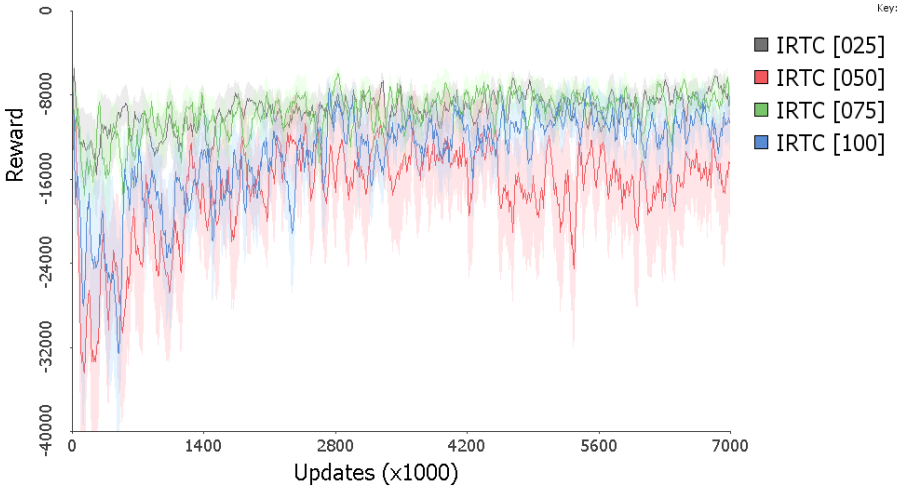


Figure 5.8: Puddle World. Different parameter settings for Fixed Plateaued.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

performance is still significantly worse than using consecutive loops through the IRTC as the threshold.

In Figures 5.10 and 5.11 we see the results of a comparison of the best parameter settings of the different implementation methods. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. From these graphs we can see that using detection of a plateau in learning as the basis of when to split

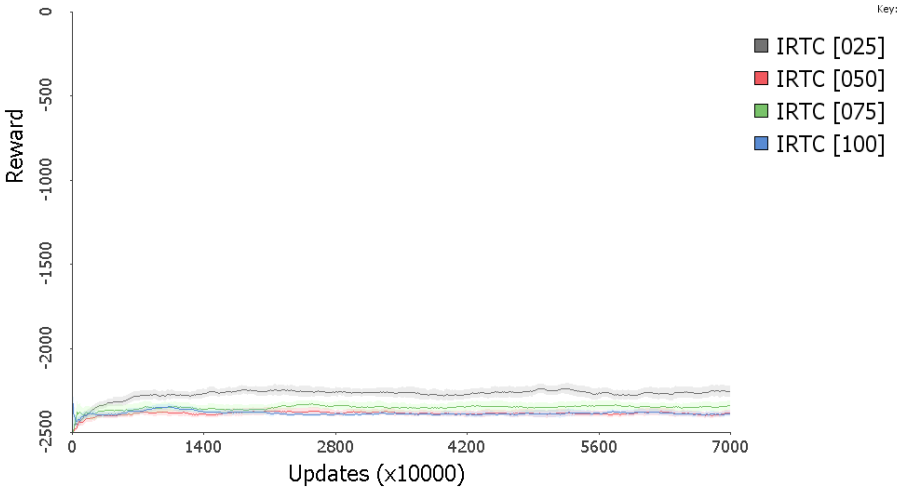


Figure 5.9: Mountain Car. Different parameter settings for Fixed Plateaued.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

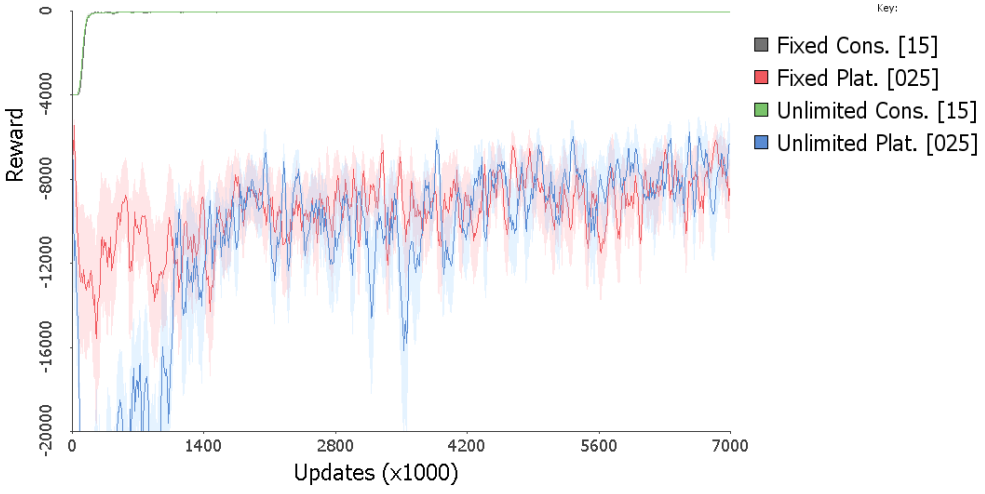


Figure 5.10: Puddle World. Comparison of the best IRTC detection implementations.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

a tile is ineffective compared to using detection of consecutive loops through an IRTC when an agent learns through episode roll-outs. Furthermore, we see that in Puddle World there is very little difference in performance between detecting ITRCs of any length compared to IRTCs of length 1. However, it is very clear in that in the Mountain Car environment detecting all IRTCs is significantly more effective than detecting IRTCs of length 1.

In conclusion we can see from the results that if we use an IRTC detection mechanism in AMRAT, the implementation that consistently out performed the other implementations analysed

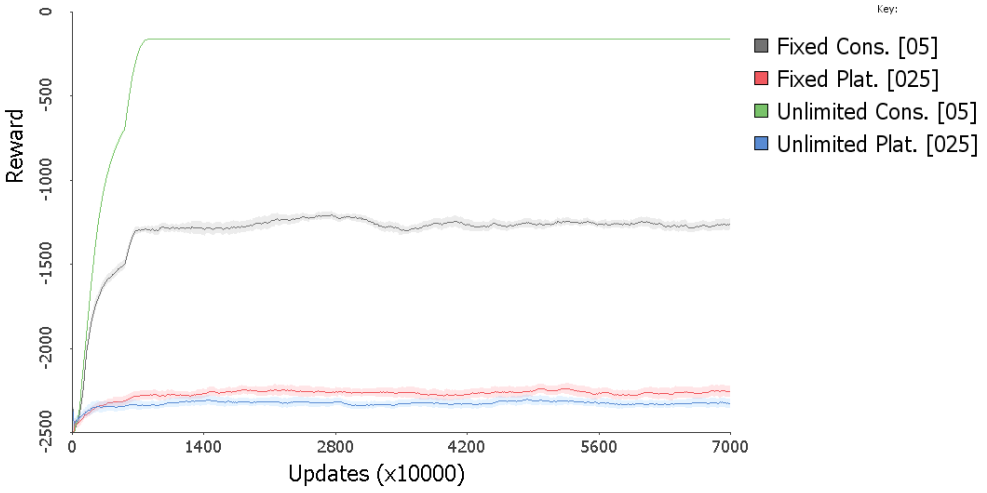


Figure 5.11: Mountain Car. Comparison of the best IRTC detection implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

was Unlimited Consecutive.

Detecting estimated optimal paths

There are 3 possible ways to combine the implementation options discussed in the previous section:

1. Episode history as path with reaching a goal to decide when to split tiles (*Episode History*).

2. Current greedy policy with plateaued learning to decide when to split tiles (*Greedy Plateaued*).

3. Current greedy policy with reaching a goal to decide when to split tiles (*Greedy Goal*).

There is no obvious candidate for a threshold metric to be used to decide when to split tiles in path detection. For the plateau of learning we will use the same implementation used by Whiteson et al. (2007). This does not make sense when the episode history is used to determine the path. When learning has plateaued all tiles that are on the estimated optimal path will be split. For reaching a goal state the agent will immediately determine the estimated optimal path and split all tiles that optimal path.

In Figures 5.12 and 5.13 we see the results of the Episode History combination. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. The parameters used were the number of initial tiles per state feature. We can see that in Puddle World using 2 tiles per feature had no advantage over using 4

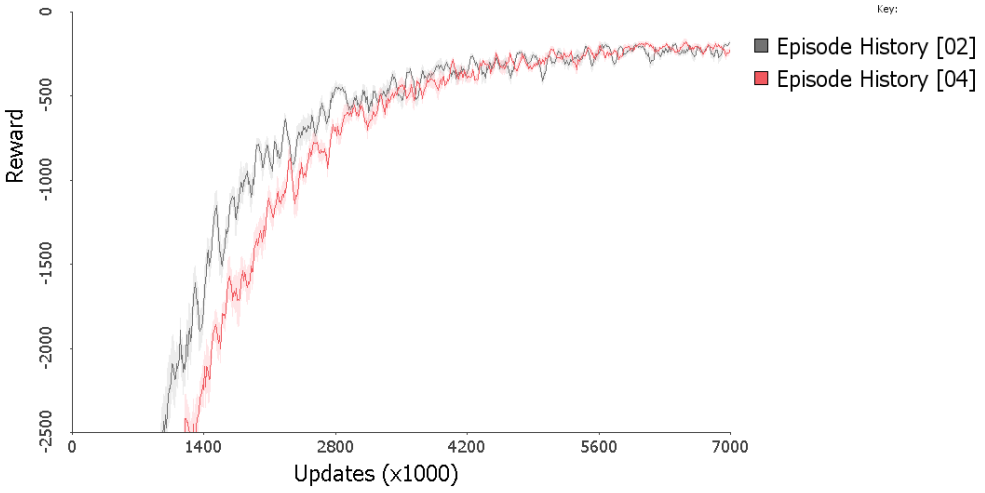


Figure 5.12: Puddle World. Results of Episode History combination.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

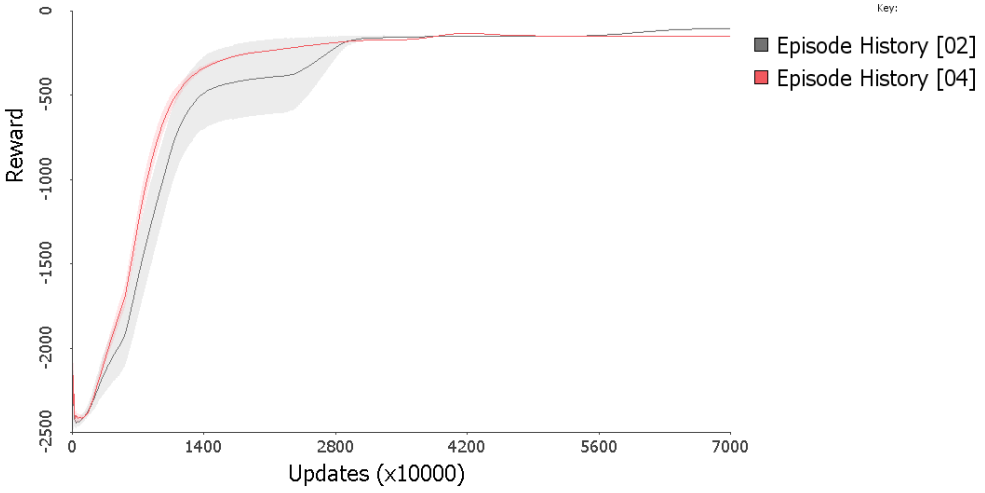


Figure 5.13: Mountain Car. Results of Episode History combination. $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

tiles per feature. However, we see a statistically significant increase in performance for using 2 tiles per state feature in Mountain Car over using 4 tiles per state feature.

In Figures 5.14, 5.15, 5.16, and 5.17 we see the results of the Greedy Plateaued combination. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. The backward search implementation only splits tiles if it has a goal tile to start the search from. The backward search also discards the goal tile after splitting the tiles on the path. Therefore the Backward search method is acting in a similar

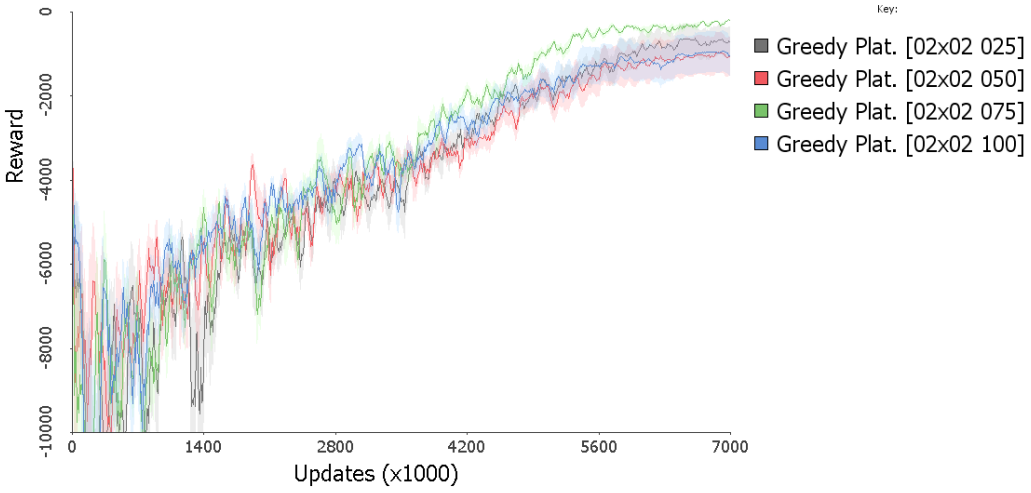


Figure 5.14: Puddle World - Backward. Results of Greedy Plateaued combination.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

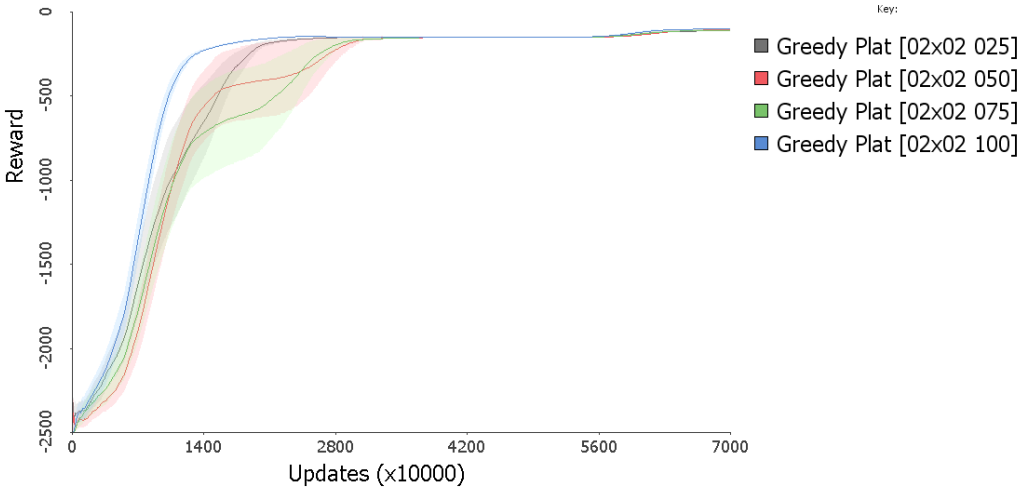


Figure 5.15: Mountain Car - Backward. Results of Greedy Plateaued combination.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

manner to the next implementation: Greedy Goal. It is for this reason we see that the Backward search method is significantly more effective at learning than the Forward search method. We see that the Forward search method is ineffective at learning in either environment.

In Figures 5.18 and 5.19 we see the results of greedy policy path detection with reaching a goal state to decide when to split tiles. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. The parameters

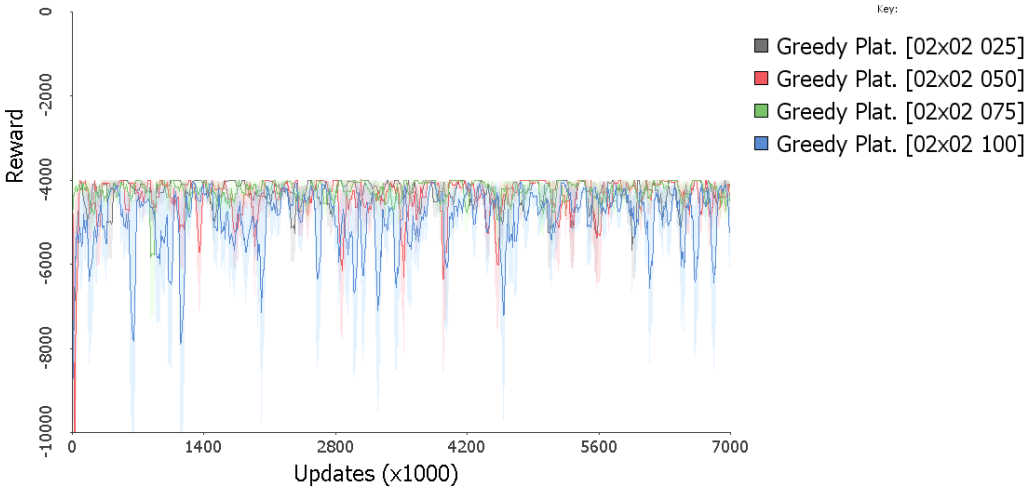


Figure 5.16: Puddle World - Forward. Results of Greedy Plateaued combination.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

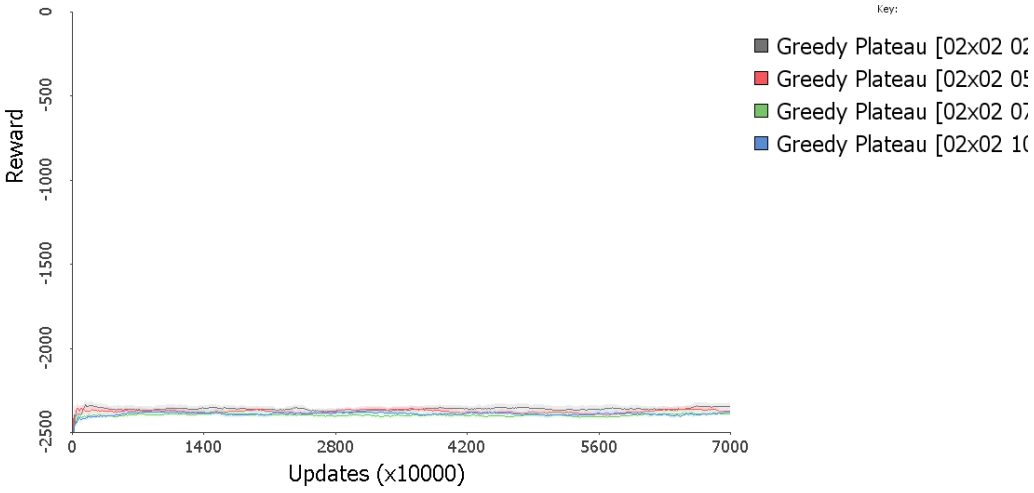


Figure 5.17: Mountain Car - Forward. Results of Greedy Plateaued combination.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

used were the number of initials tiles per state feature used. We see that the initial number of tiles per state feature makes a difference to the performance of the agent. In Puddle World the performance difference is caused when the agent does not explore the environment enough and learns that the puddle is in the direction of the goal. Therefore it tries to move away from the goal when exploiting. This causes a negative feedback cycle of the agent further not exploring enough and so on. In Mountain Car a similar dynamic occurs, however, because of the different way the agent transitions through the state space the agent can eventually break the negative feedback

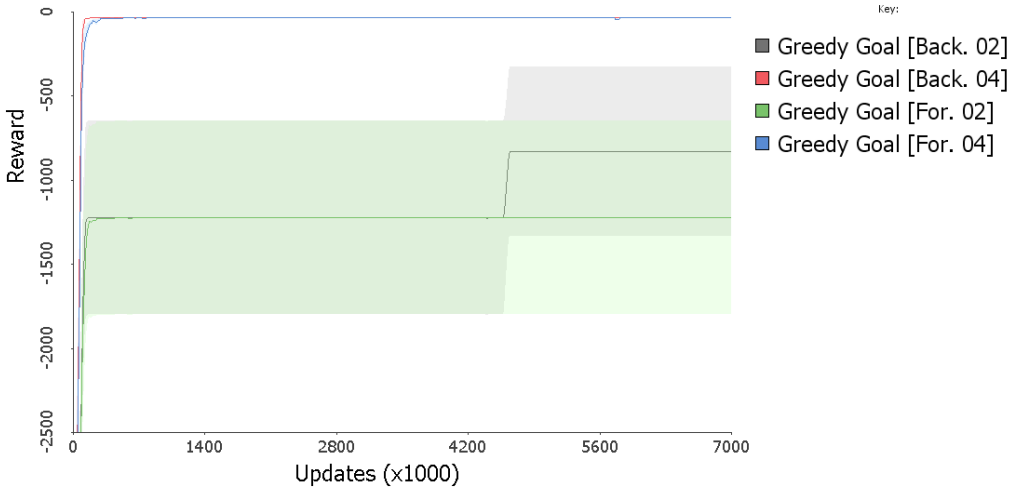


Figure 5.18: Puddle World. Current greedy policy with plateaued learning to decide when to split tiles.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

cycle and begin to learn properly. The agent's learning is slowed because of this and we see this reflected in their performance.

In Figures 5.20 and 5.21 we see the results of a comparison of the best parameter settings of the different implementation methods. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. We see in Puddle World the Backward search method for Greedy Plateaued out performs Episode History. However, both of these are significantly outperformed by both Greedy Goal. In Mountain Car the results are very different. Greedy Goal initially outperforms all others but by the end of learning is beaten by Greedy Plateau (Backward) and statistically the same as Episode History Goal. As previously stated, Greedy Plateau (Backward) has very similarly to Greedy Goal in the initial parts of learning.

We can see from the results that the most consistent implementation is Greedy Goal. However, it can be outperformed by using Greedy Plateaued which eventually outperforms all others in Mountain Car. Furthermore, Greedy Goal seems to be susceptible to performance desegregation based on the number of initial tiles per state feature used.

Combining the two mechanisms

Now that we have shown that both mechanisms are effective but have their limitations we shall try combining them. It is our hope that by combining these two mechanisms we can gain all the good aspects and remove their bad aspects.

Based on the empirical results we will combine one of the IRTC detection implementations

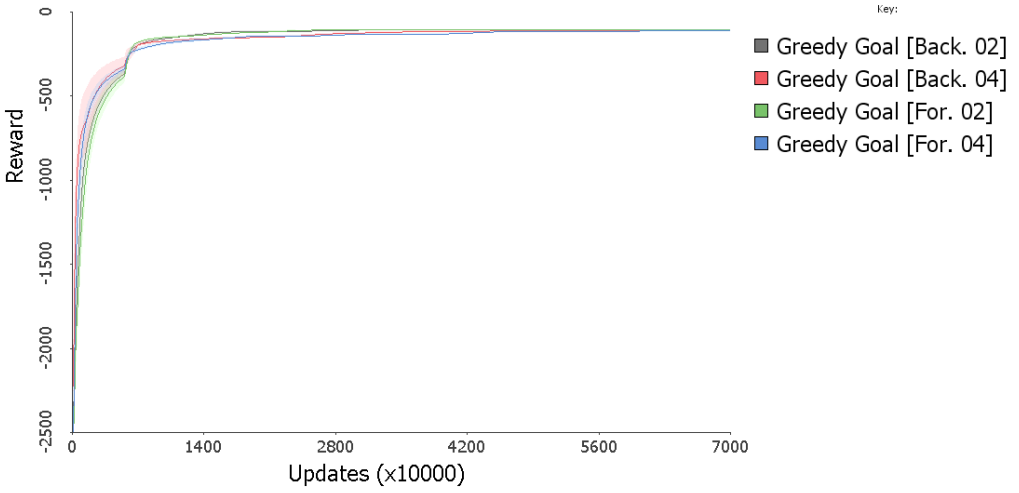


Figure 5.19: Mountain Car. Current greedy policy with plateaued learning to decide when to split tiles.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

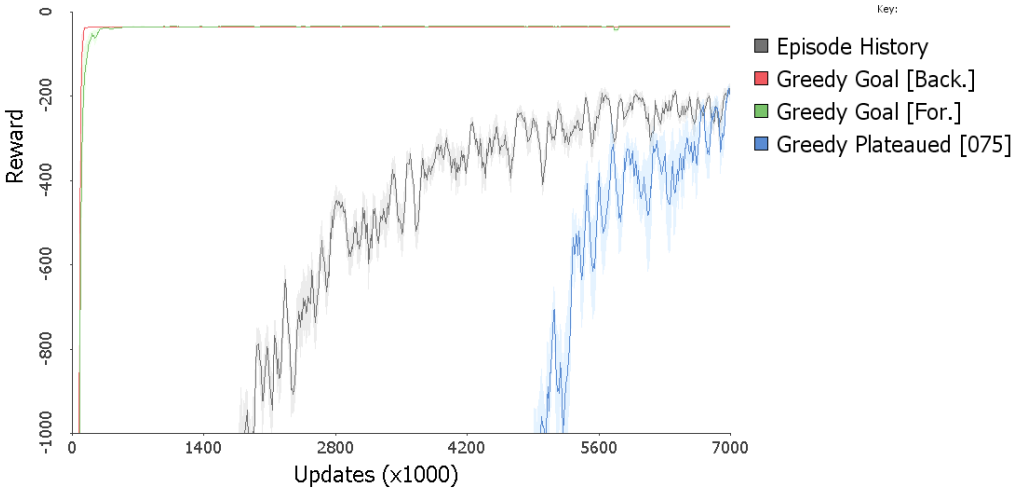


Figure 5.20: Puddle World. Comparison of the best IRTC detection implementations.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

with two of the estimated policy path implementations. We will use a trigger switch to ensure that these two mechanisms do not interfere with each other. The combinations will start by using IRTC detection and then switch to estimate optimal path detection once a goal state is found. We are using the IRTC detection method to set up the tile placement so that the tiles are not too large to learn on. We use the estimated optimal path detection for the rest of learning as it performs significantly better than IRTC detection. We will use a threshold value of 20 for IRTC detection

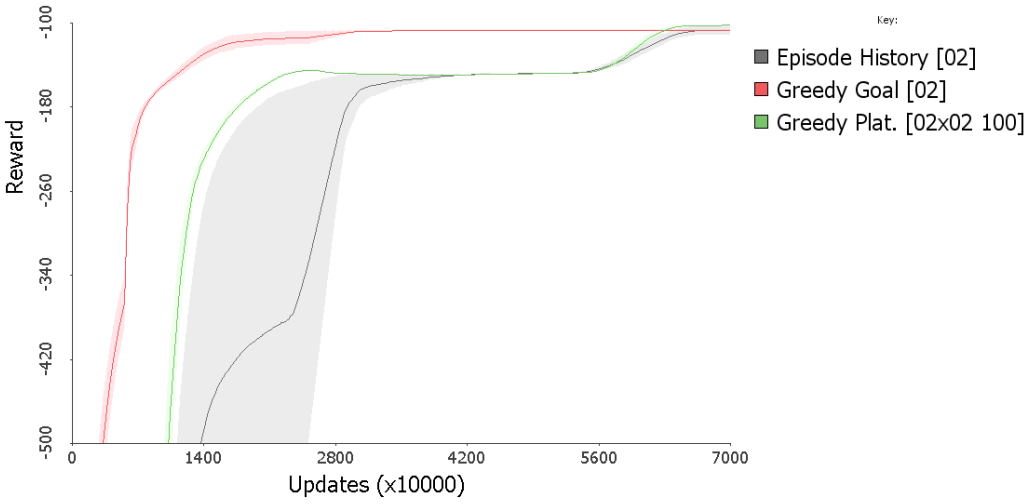


Figure 5.21: Mountain Car. Comparison of the best IRTC detection implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

mechanism and a threshold value of 100 for plateaued learning. The former so that tiles are safely large for the path detection mechanism to begin with. The latter as it was the best value found in our experiments.

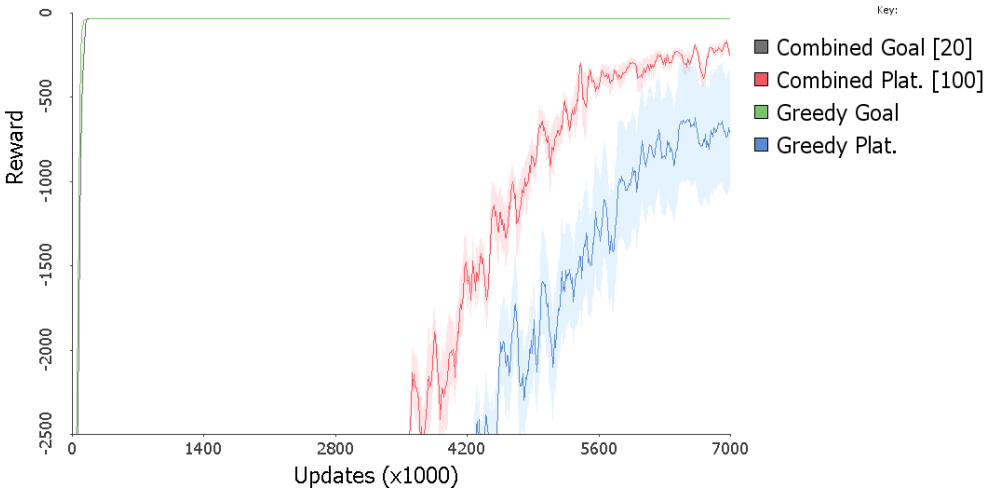


Figure 5.22: Puddle World. Comparison of the two combination implementations. Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

In Figures 5.22 and 5.23 we see the results of the combinations of the detection mechanisms as discussed above, alongside a comparison of those mechanisms by themselves. The x-axis shows the number of updates that have occurred and the y-axis shows the cumulated reward

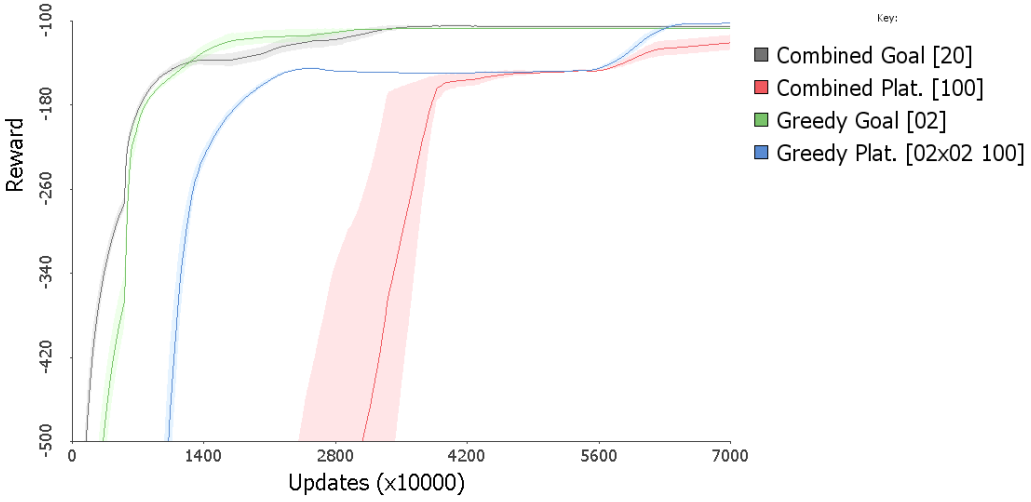


Figure 5.23: Mountain Car. Comparison of the two combination implementations.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

the agent received during an evaluation episode. The lightly coloured areas around each plot of the graph show the standard error. We see that the combination with reaching a goal state to determine when to split tiles (labelled Combined Goal) initially learns more slowly than Greedy Goal but at least matches if it betters its performance by the end of learning. We see that the combination with learning plateau to determine when to split tiles (labelled Combined Plateau) performs better in Puddle World but worse in Mountain Car.

We can see from the results that Combined Goal was most reliable algorithm in both environments. Furthermore, Combined Goal only has one parameter to set which is intuitive to understand and, we believe, need not be changed for any environment. Therefore Combined Goal is our best candidate for our implementation of AMRAT.

Design Decisions

Based on the discussion of implementation options and the empirical evidence presented in the previous section, AMRAT will be implemented in the following way:

- AMRAT will use a combination of two detection mechanisms
- AMRAT will initially split tiles using IRTC detection
- After a goal is found tiles will be split using path detection
- Paths are searched from the goal tile backward toward the start

The transition model will track the number of inter-tile transitions that have occurred and which actions that were performed to cause them. The transition model will then be able to

provide the empirical probability of transitioning from one tile to another dependent on the action performed. Along with the probabilities from the inter-tile transition model the Q-value values will be used to determine the currently held optimal path from an initial tile to a goal tile.

5.2.3 Algorithmic Automation

Automated Mixed Resolution Acyclic Tiling (AMRAT), shown in Algorithm 7, is based on MRAT and the standard tile coding algorithm (see Algorithm 5).

Algorithm 7 AMRAT($S, A, m, \pi_e, \alpha, \gamma$)

Initialise m transition models
Initialise m transition cycle detectors
Initialise m tilings with 2 tiles per state feature
trigger $\leftarrow$ **false**
threshold $\leftarrow 20$
repeat
  repeat
    $s \leftarrow$ current state
    $a \leftarrow \pi_e(s)$ and perform a
    Observe s' and r
    $\Delta \leftarrow r + \gamma \arg\max_{a'} Q(s', a') - Q(s, a)$
    for $i \leftarrow 1$ **to** m **do**
      $Q_i(s, a) \leftarrow \frac{\alpha}{m}\Delta$
      if not trigger **then**
        if IN-CYCLE($\mathrm{t_i}$) **then**
          Increment counter on t_i
          if Counter on t_i ¿= threshold **then**
            SPLIT($\mathrm{t_i}$)
            CLEAR-TM(i)
        else
          Reset counter on t_i
      if $t_i \neq t_i'$ **then**
        UPDATE-TM(i, $\mathrm{t_i}$', a, $\mathrm{t_i}$)
    if IS-TERMINAL(s') **then**
      trigger $\leftarrow$ **true**
      for $i \leftarrow 1$ **to** m **do**
        $route \leftarrow$ GREEDY($\mathrm{t_i}$')
        for Each tile t_r in $route$ **do**
          SPLIT($\mathrm{t_r}$)
        CLEAR-TM(i)
  until s is terminal **or** time expires
until time expires

AMRAT requires the set of states S, the set of possible actions A, the number of tilings to use m, an exploration policy π_e, the learning rate α, and the future discount factor γ. AMRAT begins by initialising m transition models, m transition cycle detectors, m tilings with 2 tiles per state feature, the trigger to false, and the threshold to 20. AMRAT then enters the learning cycle.

The outer repeat loop ensures that if an episode ends a new episode begins, the inner repeat loop ensures the agent learns during episodes. Each learning loop AMRAT gets the current state s, decides on and performs the next action a, and observe the next state s' and reward r. Then it calculates the Bellman error Δ. For each tiling it updates the Q-value of the activated tile t_i of the tiling $Q_i(s, a)$. Next, if the trigger has not been fired, if the agent is in an IRTC in that tiling it increments the counter on the tile t_i and if the threshold was reached splits that tile. If the tile was not in an IRTC the counter on the tile t_i is reset. Then, if the tile activated by s' in tiling i, t'_i, is different to the tile activated by s in tiling i, t_i, then the transition model is updated to include this transition. After looping through all the tilings, if s' is a terminal state then $\mathrm{trigger}$ is set to true and for each tiling the greedy policy estimated optimal route from the tile containing the goal state t'_i is determined, every tile in the route is split, and the transition model of that tiling is cleared.

5.3 Experimentation & Results

In this section we will perform two sets of experiments each with a different focus. The first set will focus on observing the comparative impacts of the unlearning phenomenon on TC and AMRAT. The second set will focus on the evaluating the performance of AMRAT against TC and state-of-the-art competitors.

5.3.1 Comparative impacts of the unlearning phenomenon

Our theoretical and empirical work so far has only considered the impacts of IRTCs where the approximating agent only uses one tiling to learn about the environment. This allows us confidence to state that the unlearning phenomenon is real. However, whilst some algorithms that adapt their TC use a single tiling this is not best practise for TC in general. The literature (Sutton and Barto (1998)) recommends that TC agents use multiple tilings to obtain the best performance.

Our aim in these experiments is to establish where we can observe the unlearning phenomenon caused by IRTCs in agents using TC with multiple tilings. Furthermore, whether that the same phenomenon is observable in AMRAT. In any complex environment it is impractical to study the Q-values of every tile in every tiling during learning, nor is it trivial to predictable which Q-values will impacted by an IRTC beforehand. Instead we take a macro view of the agent's performance and look for signs of the unlearning phenomenon impacting performance. Since the unlearning phenomenon can occur at any point while the agent continues to learn we will only consider times when the agent's performance has stabilised into apparent optimality. When Q-learning is used it is expected that an agent should not deviate from a learnt optimal policy. Therefore, if we observe an agent's performance degrade more than stochastic probability would allow we are indirectly observing the impacts of the unlearning phenomenon.

5.3.1.1 Experiment set up

To compare the impacts of the unlearning phenomenon we will perform experiments in two environments, each under two conditions. The two environments will be a 2D problem, Puddle World, and a 4D problem, Cart Pole. The two conditions are: 1) uniform TC varying the tile size and number of tilings (TC agent); and, 2) adaptive TC using the full AMRAT algorithm (AMRAT agent).

The agents will be evaluated based on their performance in evaluation episodes at set intervals during learning. When an agent's performance is due to be evaluated it is moved from the "learning" environment to an "evaluation" environment. The evaluation environment is identical to the beginning of an episode in the learning environment. During an evaluation episode the agent uses a purely greedy action selection policy and does not learn. At the end of an evaluation episode the agent is moved back into the "learning" environment with the same state as it was previously and continues to learn. The accumulative reward during an evaluation episode is recorded as the agent's performance at that time.

The experiments in the Puddle World environment will be run for a total of $7,000,000$ updates with each episode lasting at most $4,000$ updates. An evaluation episode will be conducted every 1000 updates. The TC agent will be evaluated with the number of tilings, $m \in 1, 2, 3, 4, 5$, and the number tiles per feature, $n \in 2, 5, 10, 20$. The experiments will be repeated 10 times.

The experiments in the Cart Pole environment will be run for a total of $70,000,000$ updates with each episode lasting at most $4,000$ updates. An evaluation episode will be conducted every $10,000$ updates. The TC agent will be evaluated with the number of tilings, $m \in 1, 2, 3, 4, 5$, and the number of tiles per feature, $n \in 10, 20, 30, 40, 50$. The experiments will be repeated 10 times.

The Cart Pole Environment is a 4-dimensional problem of controlling a cart to balance a pole (see 5.24). There are two possible actions: 1. apply a leftward force to the cart (LEFT); or, 2. apply a rightward force to the cart (RIGHT). The agent can perceive the current state of the environment which consists of: 1) the cart's location; 2) the cart's velocity; 3) the angle of the pole on top of the cart; and, 4) the velocity of the pole on top of the cart. Each episode begins with the cart in the centre of the environment with the pole balanced on top. The pole is attached to the top of the cart with a frictionless joint. There is a set distance the cart can move in either direction, left or right, before it reaches the edge of the environment. The agent receives a reward of 1 for every time step the pole remains balanced. An episode ends when either the cart crosses the edge of the environment or the pole becomes unbalanced. The pole is considered unbalanced when the angle between the cart and the pole becomes too large. We use the environment default variable values as specified in the RL-Library, see (RL-Library 2009).

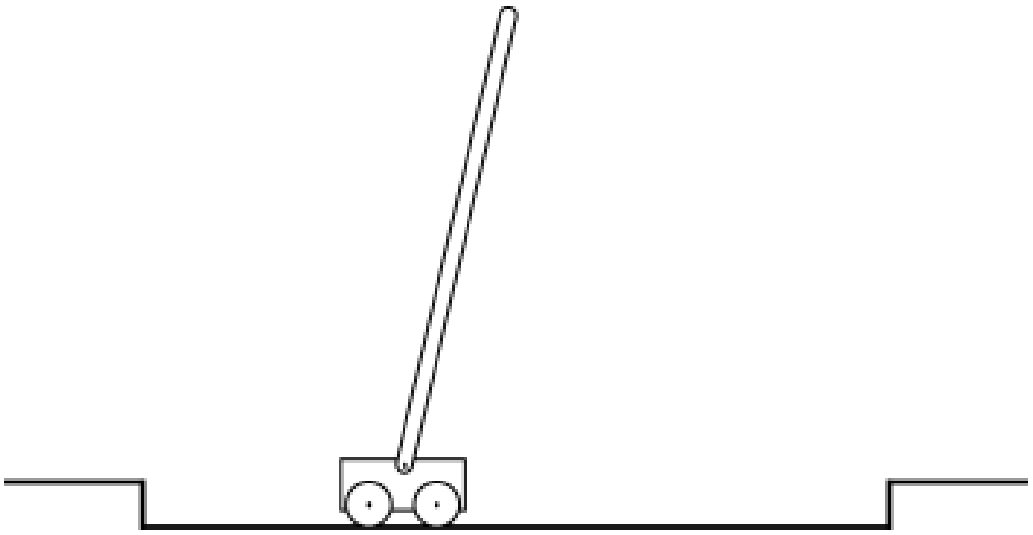


Figure 5.24: Cart Pole Environment. Objective: balance the pole by moving the cart left and right.
Image from: `http://library.rl-community.org/wiki/CartPole_(Java)`

5.3.1.2 Results

In our figures we have omitted some of the results to reduce the amount of visual noise to allow the reader to properly interpret the results. The results display in Figures 5.25 and 5.26 are the average of the 10 runs conducted. The transparent area of the same colour surrounding each line is the standard deviation of the 10 runs. The x-axis of each graph denotes the number of updates the agent has performed and the y-axis of each graph denotes the accumulative reward the agent received on the evaluation run conducted.

In figure 5.25 we can see the results of the AMRAT and TC agents with the number of tilings, m, and the number of tiles per feature, n. We can see that all agents appear to have reached their optimal policy. In this set of results we can see that AMRAT's performance fluctuates a very small amount which is easily accounted for by the stochastic nature of the Puddle World environment. TC with $m = 01, n = 20$ displays a less consist performance than AMRAT with sudden, temporary drops in the evaluation performance. The drops in performance are too great to be accounted for by the stochastic nature of the Puddle World environment. This behaviour is consist with the unlearning phenomenon and matches the behaviour we predicted. TC with $m = 05, n = 20$ displays a further drop in the consistency of its performance from AMRAT and TC with $m = 01, n = 20$. The agent's performance frequently and temporarily drops before improving again. The drops in performance, like the other TC agent, are too great to be accounted for by the stochastic nature of PW.

In figure 5.26 we can see the results of AMRAT and TC agents with the number of tiling, m, and the number of tiles per feature, n. We can see that the majority of the agents appear to have

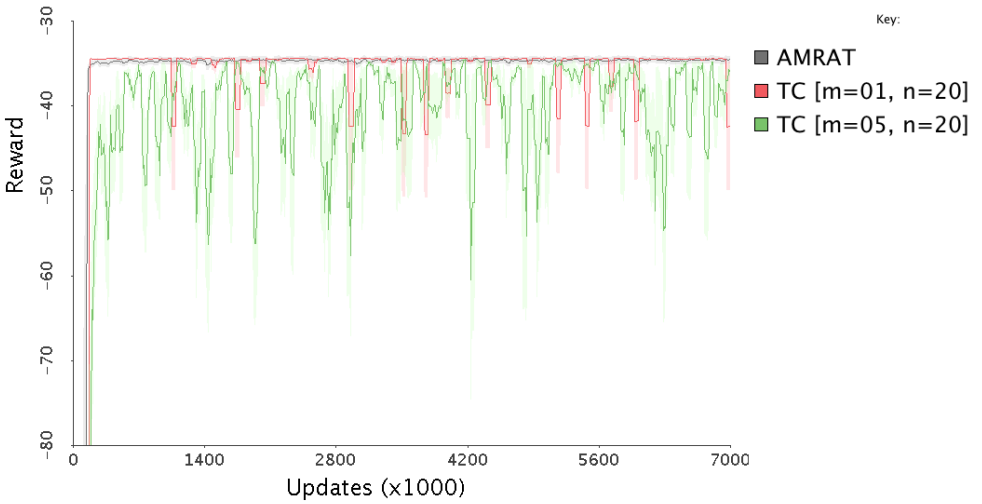


Figure 5.25: Performance of AMRAT and TC in the Puddle World environment.
Where m is the number of tilings and n is the number of tiles per feature.

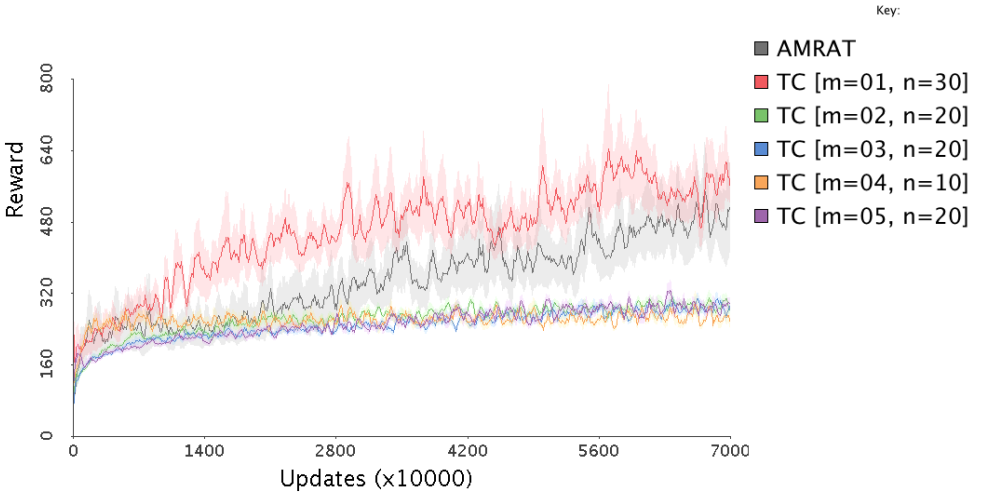


Figure 5.26: Performance of AMRAT and TC in the Cart Pole environment.
Where m is the number of tilings and n is the number of tiles per feature.

reached their optimal policy with the exception of possibly AMRAT and TC $m = 1, n = 30$. In this set of results we can see that none of the agents are able to performance consistently in their evaluation episodes. This is especially prominent in AMRAT and TC $m = 1, n = 30$ and less prominent in the other TC cases. All the agents are frequently and temporarily exhibiting drops in performance before improving again. Since the environment variables used in Cart Pole include no random starts nor transition noise we can safely accredit all the drops in performance to the unlearning phenomenon.

We can also see that in the Cart Pole environment TC with greater than one tilings has a smaller difference between peaks and troughs of performance over AMRAT and TC with one tiling. This is in contrast to Puddle World where the opposite is seen.

5.3.1.3 Conclusions

We can conclude from these results that neither Tile Coding with multiple tilings nor AMRAT is completely immune to the unlearning phenomenon. Our results from Puddle World suggest there are situations where AMRAT is more resistant than TC to the unlearning phenomenon however from these results alone it is not possible to attribute the source of the resistance. The resistance could be limited to only Puddle World or linked the including (solely or in some combination): 1) the low number of dimensions; 2) the state-independent mapping between actions and state changes; 3) etc. Looking ahead to the results from Figure 5.40 in the next section we can see that all but Fixed TC $n = 1, m = 50$ show a resistance to the unlearning phenomenon. We can therefore conclude the resistance to the unlearning phenomenon is not limited to only Puddle World. Furthermore, since in Mountain Car there is a state-dependent mapping between actions and state changes we can say that the low number of dimensions is likely to be the cause of AMRATs resistance to the unlearning phenomenon. There is no apparent explanation as to why range between performance peak and trough is more for greater than one tilings in Puddle World and less of for greater than one tilings in Cart Pole.

5.3.2 Performance Evaluation

We will use the same environments, Puddle World and Mountain Car, to evaluate the performance of AMRAT against the state-of-the-art competitors. We will begin by parameter tuning the competitors for AMRAT.

In Figures 5.27, 5.28, 5.29, and 5.30 we perform parameter tuning of fixed TC in the Puddle World environment. We conducted our tuning trying different numbers of tilings, $m \in \{1, 5, 10\}$, and different numbers of tiles per state feature, $n \in \{5, 10, 15, 20, 25, 30\}$. Our results showed that the best of 1, 5, and 10 tilings were 20, 25, and 25 respectively.

In Figures 5.31, 5.32, 5.33, and 5.34 we perform parameter tuning of fixed TC in the Mountain Car environment. We conducted our tuning trying different numbers of tilings, $m \in \{1, 5, 10\}$, and different numbers of tiles per state feature, $n \in \{5, 10, 25, 50, 100\}$. Our results showed that the best of 1, 5, and 10 tilings were 50, 50, and 25 respectively. The different configurations of fixed TC are labelled with their settings in square brackets; the first number is the number of tilings used, the second is the number of tiles per state feature.

In Figures 5.35 and 5.36 we perform parameter tuning of ATC in the Puddle World and Mountain Car environments. We conducted our tuning trying different settings for $p \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100\}$. We only show a subset of each to save confusion on the graph. Our results showed that in Puddle World $p = 50$ performed marginally better than the others and in Mountain Car $p = 40$ performed marginally better than

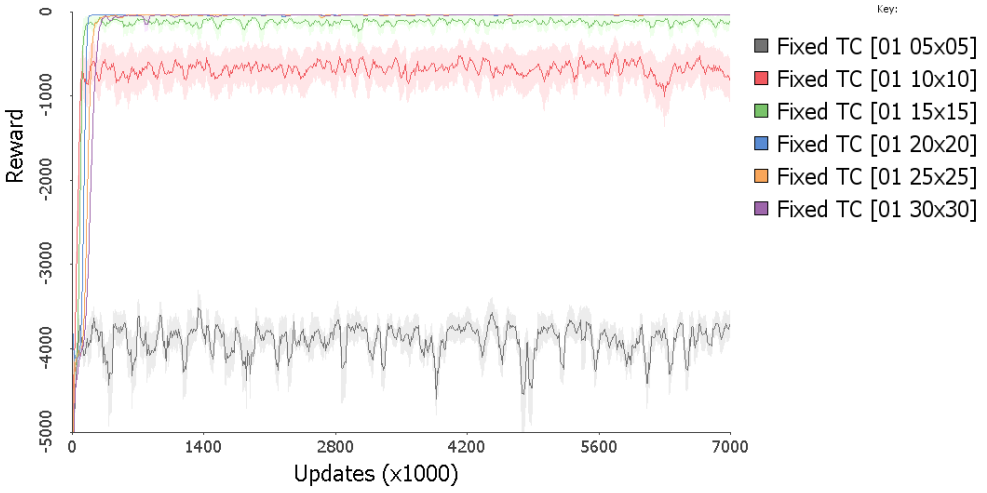


Figure 5.27: Puddle World. Parameter tuning for fixed Tile Coding.
Where $m = 1, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

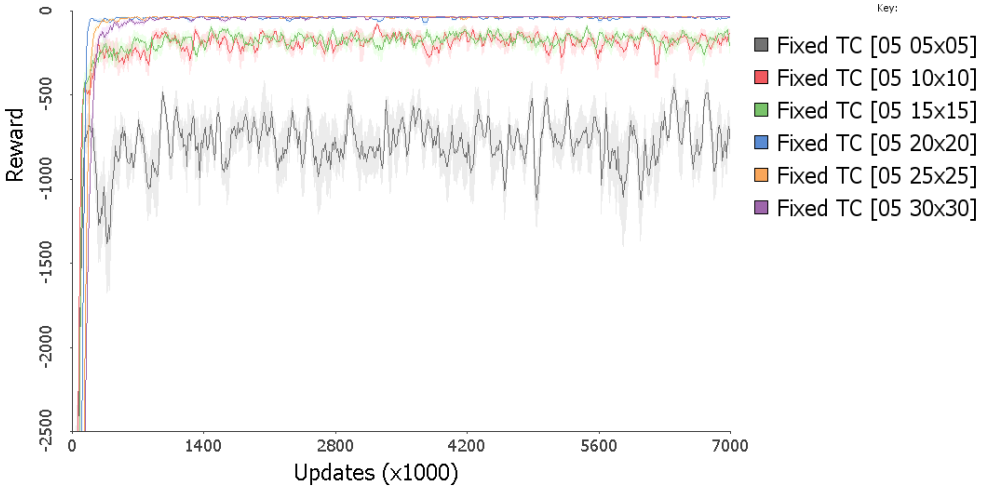


Figure 5.28: Puddle World. Parameter tuning for fixed Tile Coding.
Where $m = 5, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

the others. However, it can be seen in the graphs there was no predictability of performance based on the value of p. The different configurations of ATC are labelled with their settings in square brackets; the number is the value of p and Difference means that the difference criterion was used.

We only used the Difference criterion on our results as the ATC implementation received from the authors learnt the value function for each tile rather than the Q-value. The ATC implementation also used the transition function in action selection to generate the Q-values by sampling

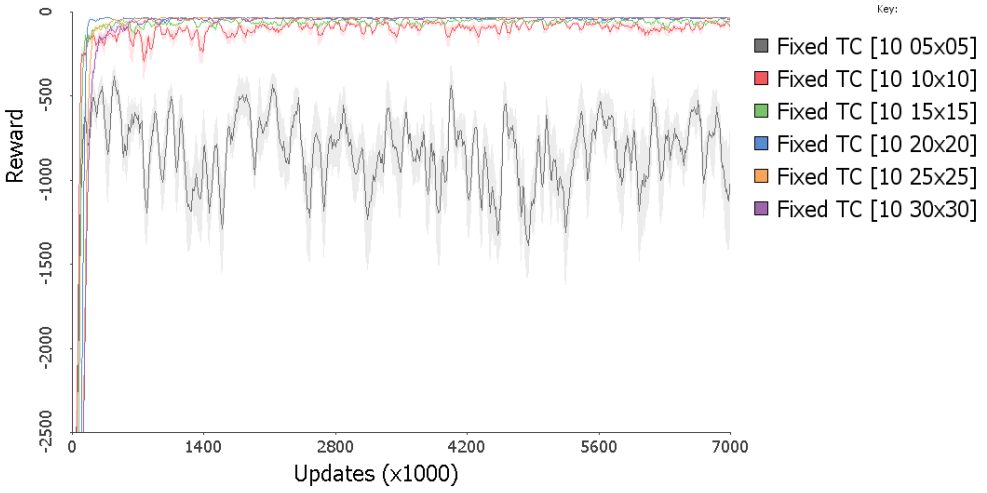


Figure 5.29: Puddle World. Parameter tuning for fixed Tile Coding.
Where $m = 10, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

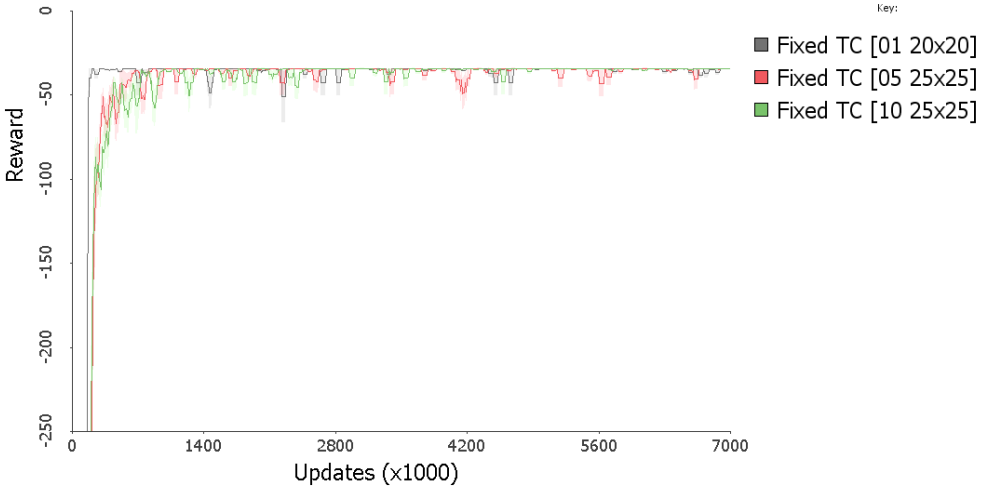


Figure 5.30: Puddle World. Parameter tuning for fixed Tile Coding. Comparison of the best.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

each action once in the environment and using the sampled transition reward and state value. Furthermore, the learning algorithm sampled random states of the environment instead of playing out episodes. Finally, the policy criterion requires the transition function to gain access to the number of possible successor states. For these reasons the implementation given had to be altered to remove the need for knowledge about the transition function. These alterations included: playing out episodes, learning Q-values and using them in action selection, and modifying one of the

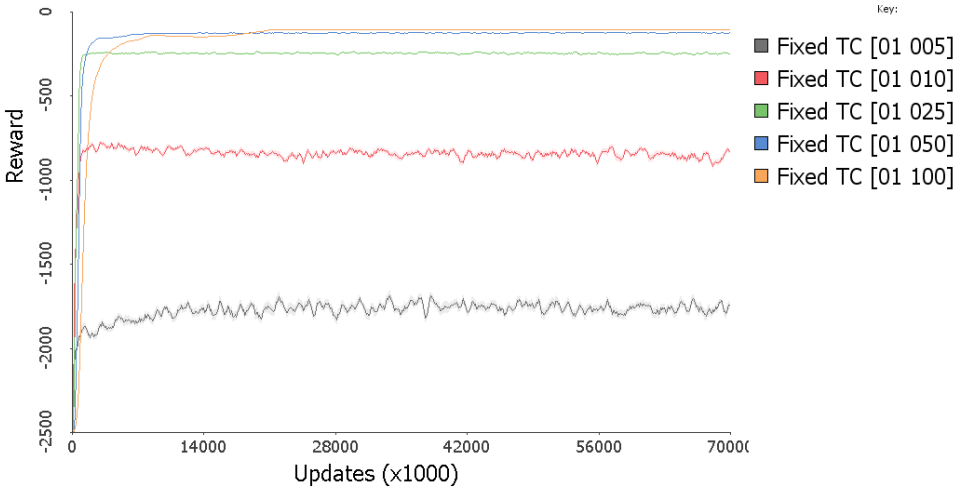


Figure 5.31: Mountain Car. Parameter tuning for fixed Tile Coding.
Where $m = 1, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

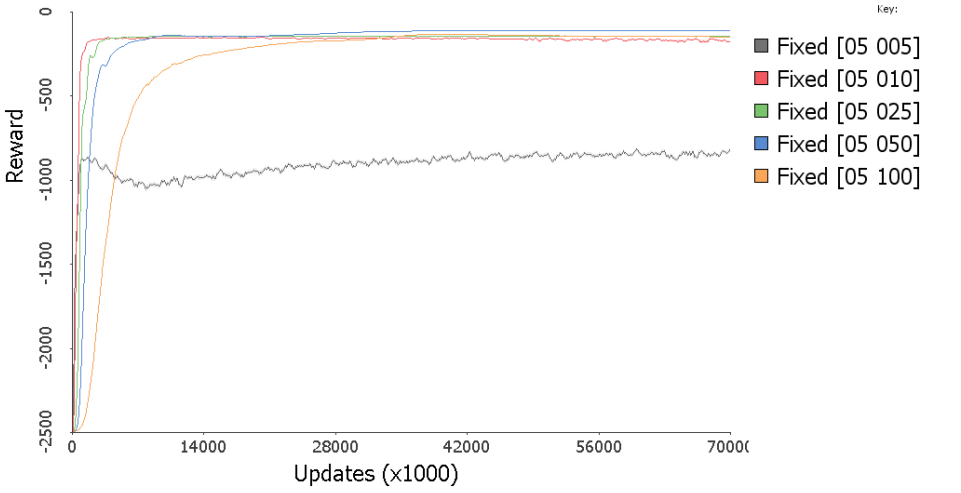


Figure 5.32: Mountain Car. Parameter tuning for fixed Tile Coding.
Where $m = 5, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

tile splitting criterion[3].

5.3.2.1 Results of AMRAT

Puddle World and Mountain Car were used to compare the performance of AMRAT against fixed uniform TC and ATC (Whiteson et al. 2007). The implementation used in the evaluation of ATC used does not require the transition function. We use the best parameters as found in our

[3]Specifically, the policy criterion was altered to increment the change counter when a sub-tile's Q-values would result in a possible change in policy

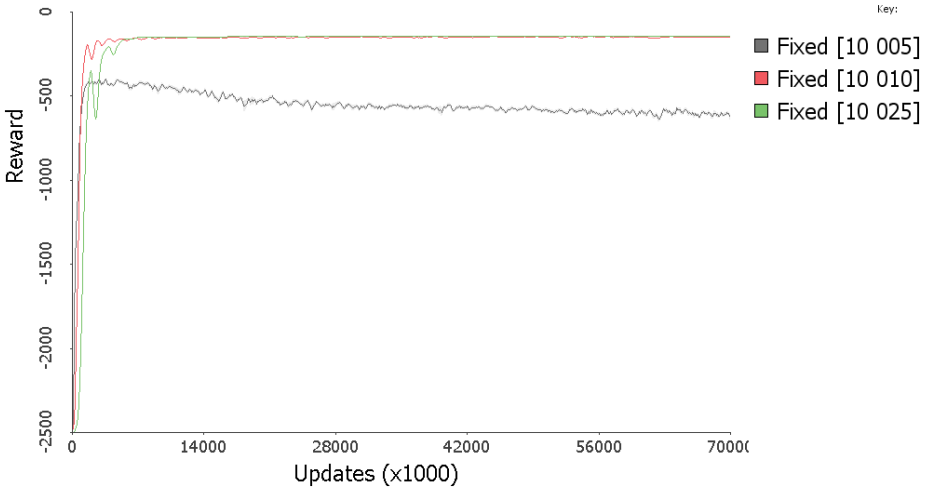


Figure 5.33: Mountain Car. Parameter tuning for fixed Tile Coding.
Where $m = 10, \alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

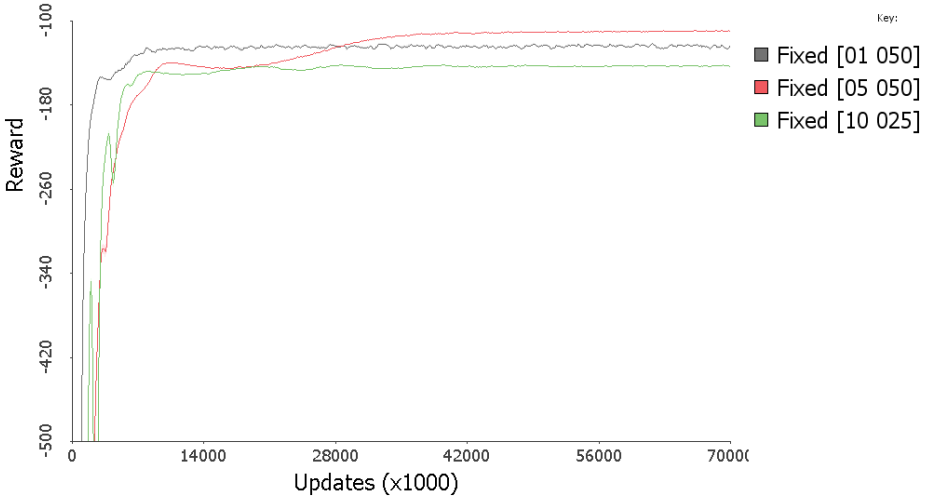


Figure 5.34: Mountain Car. Parameter tuning for fixed Tile Coding. Comparison of the best.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

parameter tuning of each state-of-the-art algorithm.

Q-learning in the Puddle World Environment was set up as follows: $\alpha = 0.4$ and $\gamma = 0.9$, and ϵ-Greedy exploration was used with $\epsilon = 0.4$ without linear decay. Experiments were repeated 10 times and the average reward after each update is shown the results (shown in Figures 5.37 and 5.38). The y-axis is the reward and the x-axis is the number of updates. The lightly coloured areas surrounding each line represents the standard error allowing us to be confident that these results are statistically significant.

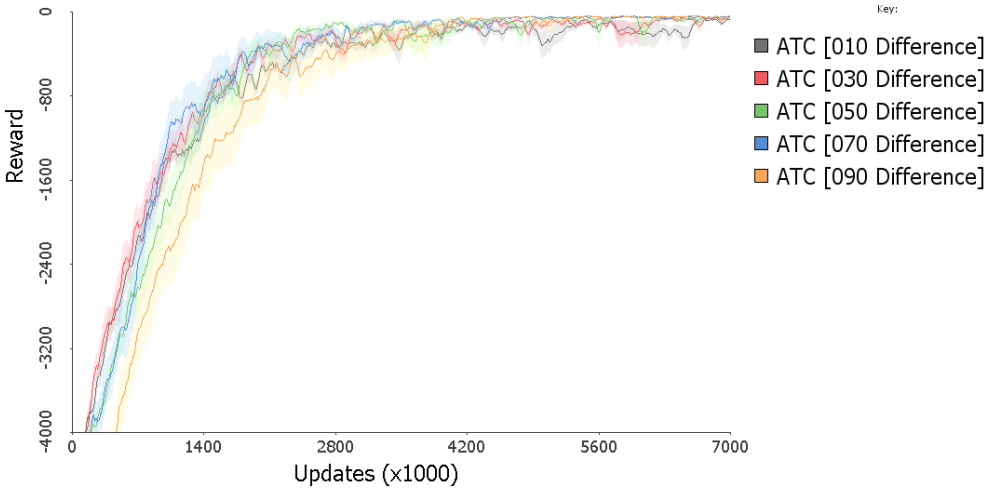


Figure 5.35: Puddle World. Parameter tuning for Adaptive Tile Coding.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

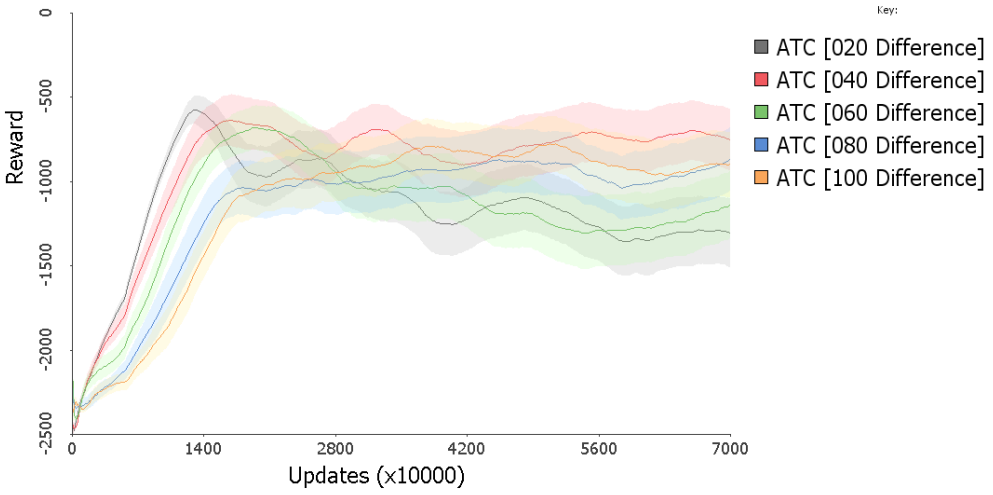


Figure 5.36: Mountain Car. Parameter tuning for Adaptive Tile Coding.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

We can see in left-hand side graph of the results shown in Figure that ATC is outperformed by TC and AMRAT and so in the right-hand side graph ATC is excluded. We can see that AMRAT both learns the fastest than any of its competitors and the final policy is more stable.

Next we performed similar experiments in the Mountain Car environment to confirm that AMRAT can match or exceed the performance of fixed uniform TC in non-navigation environments.

Q-learning in the Mountain Car environment was set up as follows: $\alpha = 0.4$ and $\gamma = 0.999$,

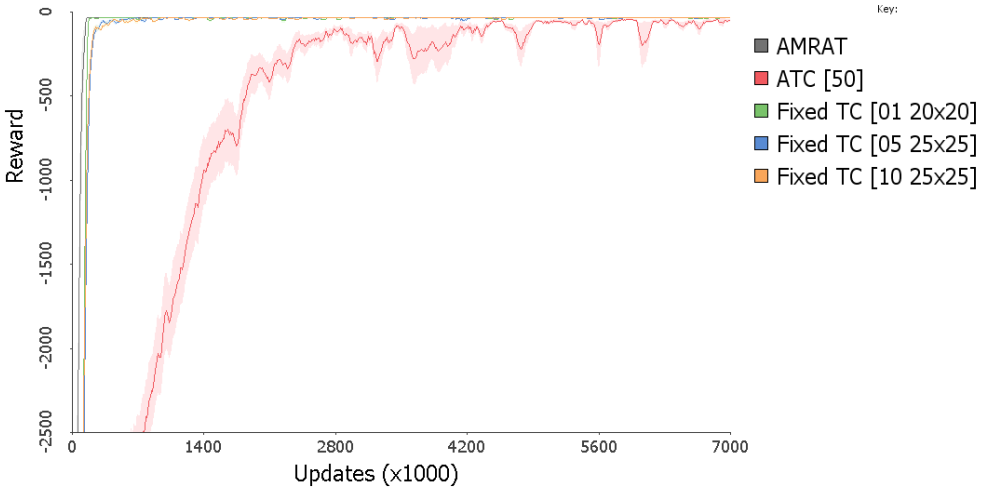


Figure 5.37: Puddle World. AMRAT state-of-the-art comparison.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

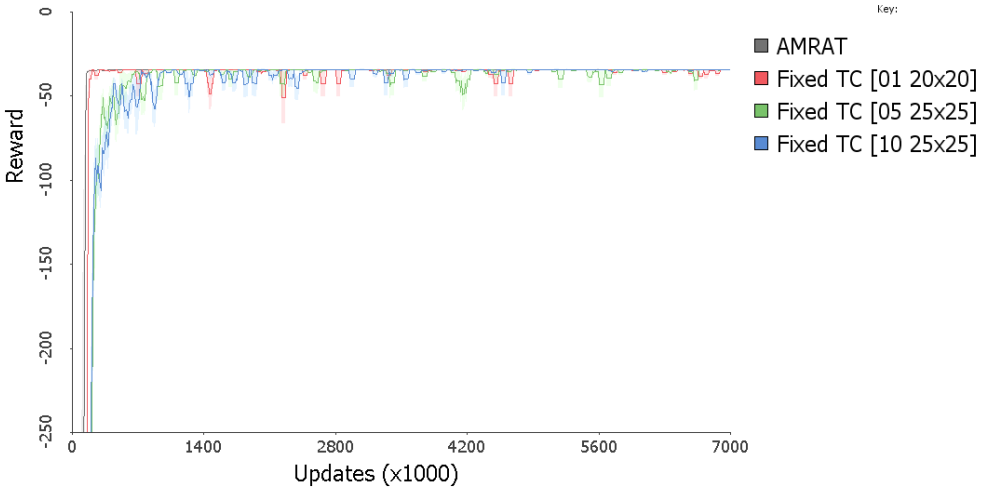


Figure 5.38: Puddle World. Vs Tile Coding state-of-the-art comparison.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

and ϵ-Greedy exploration was used with $\epsilon = 0.4$ without linear decay. Experiments were repeated 10 times and the average reward after each update is shown the results (shown in Figures 5.39 and 5.40). The y-axis is the reward and the x-axis is the number of updates. The lightly coloured areas surrounding each line represents the standard error allowing us to be confident that these results are statistically significant.

We can see in left-hand side graph of the results shown in Figure that ATC is outperformed by TC and AMRAT and so in the right-hand side graph ATC is excluded. We can see that initially

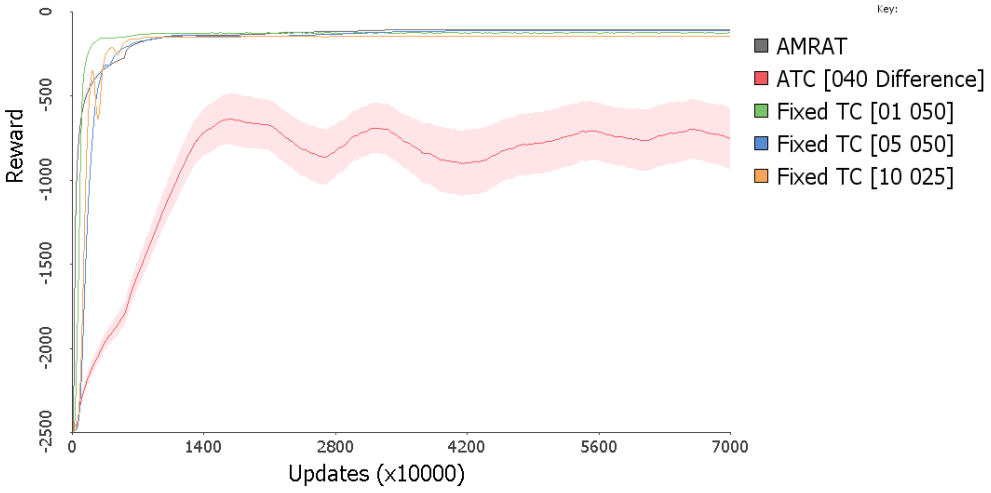


Figure 5.39: Mountain Car. AMRAT state-of-the-art comparison.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

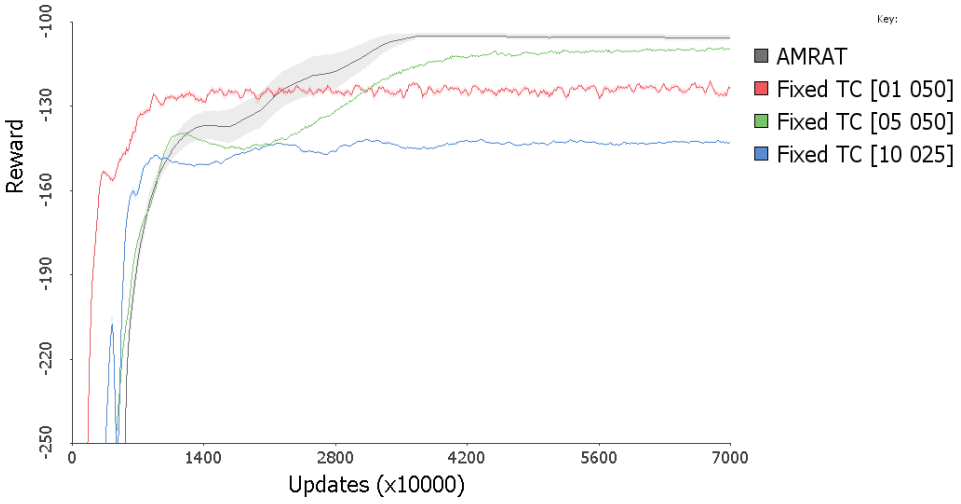


Figure 5.40: Mountain Car. Vs Tile Coding state-of-the-art comparison.
Where $\alpha = 0.4, \gamma = 0.9, \epsilon = 0.4$

fixed TC with 1 tiling and 50 tiles per state feature learns the fastest but settles on its best policy just after fixed TC with 10 tiling and 25 tiles per state feature. We can also see that fixed TC with 1 tiling and 50 tiles per state feature takes longer to learn but do eventually learn better final policies. Moreover, AMRAT statistically outperforms TC by the end of learning.

5.4 Conclusions

5.4.1 Discussion

This chapter has presented a novel algorithm AMRAT, an automated version of the manual tile placement strategy MRAT. AMRAT outperforms all fixed uniform TC representations we trialled without any prior knowledge of the environment or trial and error guesses of different numbers of tilings and tiles per feature. AMRAT achieves this by automatically altering tiles during learning to better match the environment. One of the appeals of TC is its typically speed of execution, however a user of TC is required to perform parameter tuning. AMRAT does not require parameter tuning and outperforms TC.

CHAPTER 6

Conclusion and Future Work

This chapter concludes the thesis by stating the contributions, discussing their limitations and stating possible future work.

This chapter begins by stating the contributions of the thesis. Starting with the theoretical properties of transition cycles (Chapter 3), followed by the application to tile coding (Chapter 4), and the state-of-the-art algorithm Automated Mixed Resolution Acyclic Tiling (Chapter 5). Then this chapter discusses the limitations of the contributions and possible future work.

6.1 Contributions and Conclusions

Theoretical Properties of Transition Cycles

In chapter 3 we uncovered new theoretical properties regarding transition cycles with identical transition rewards. Specifically, when an agent uses SARSA or Q-Learning it can temporarily diverge from learning the true optimal policy when interacting with identical reward transition cycles. There is no limit to how many times or when this can occur during learning as long as the agent continues to learn. Therefore, poorly timed decisions to halt learning or evaluate the agent's performance will reflect this temporary divergence. This can only be counteracted by reducing the rate at which Q-values converge or by reducing the probability that an agent will perform consecutive loops through identical reward transition cycles.

Application to Tile Coding

In chapter 4 we applied these theoretical properties to Tile Coding. We see that tiles can induce identical reward transition cycles where there are identical reward paths through the states in a

tile. These induced identical reward transition cycles tend to be short and therefore the agent's estimated optimal policy is at greater risk of temporarily diverging. The chapter then conducted an empirical study on whether it was beneficial to bias more significant state features. While the results showed that there are benefits for bias, incorrect bias can result in extremely hindered learning. Therefore it is ill-advised to bias one state feature over another without being highly confident of the relative significance of each state feature. Finally, heuristics were derived from the theory and empirical studies: tiles should not be too large and tiles near or on the optimal route through the MDP should be small. These heuristics were put together to create a tile placement strategy called Mixed Resolution Acyclic Tiling (MRAT). MRAT was then verified empirically in the Mud World environment.

Automated Mixed Resolution Acyclic Tiling

In chapter 5 of this thesis we discussed in detail the different implementation options available for our new algorithm Automated Mixed Resolution Acyclic Tiling (AMRAT). We then conducted an empirical study of the performance of the different implementations individually and in promising combinations. Finally, we compared AMRAT to its state-of-the-art competitors. AMRAT begins with a uniform coarse tile coding and over time moves toward an MRAT tile coding. AMRAT achieves this through two detection mechanisms: 1) IRTC detection; and, 2) optimal path detection. AMRAT exclusively uses IRTC detection until a goal state is found, then it switches to exclusively using optimal path detection. The IRTC detection mechanism ensures that the optimal path detection mechanism has a good platform to work from. AMRAT outperforms all of its competitors and does not have any parameters.

6.2 Limitations and Future Work

Theoretical Properties of Transition Cycles

In our work on theoretical properties assumptions 1 and 2 are fundamental and therefore cannot be relaxed. The theory is based around the concept of Identical Reward Transition Cycles; there must exist transition cycles in the MDP (Assumption 1) and the reward for each transition in the cycle must be identical (Assumption 2). Furthermore, in the extension to Q-Learning the IRTC must be an $\arg\max_a$ IRTC. This left Assumption 3 as the only assumption that could be relaxed. Assumption 3 states that number values are stored approximately. Although the main prognosis of the theory does not require this assumption, it is used to gain further understanding as to how the parameters and initial Q-values impact the rate of convergence.

Further work could include extending the theory to cover other RL algorithms. Lemma 1 does not provide a full mathematical proof for the common recurrence relation lemma in the general case. Rather it proves the cases where the length of the cycle is 1 and 2, then argues that this can be extended to a transition cycle of length n. Whilst this is more than sufficient for the work in this thesis, future work could produce a fully mathematical proof to the general case of transition

cycles of length n. Lemma 2 only used the recurrence relation of transition cycle length 1 in its proofs. This is the most useful case as the most common length of TC induced IRTC is length 1. However, this means there are nuances in cases where the length of the transition cycle is greater than 1 that are not fully mathematically described. Further work could use the general case formula for a recurrence relation from a transition cycle of length n in the proof alongside the case of a transition cycle length 1. Lemma 3 describes the cases where the policy will change thereby proving that it is possible to happen. Further work could research in more detail how these situations arise and how they are caused. Further work into Lemma 4 could explore exactly how the different parameters impact convergence, especially in the cases where the length of the transition cycle is greater than 1 and how having multiple values to update leads to more complex behaviours. Further work into Theorem 1 could provide mathematical models of how exactly consecutive loops through an IRTC correlate to policy change. Also, how frequent but not consecutive loops impact the probability of policy change.

Application to Tile Coding

The empirical study into the impacts of tile shape was conducted in one environment and whilst this was insightful further work could conduct similar experiments in different types of environments.

The manual tile placement algorithm, MRAT, being based on heuristics is limited in its effectiveness to environments where the heuristics apply. MRAT will be especially effective in environments where tile coding can induce many IRTCs; conversely MRAT may be less effective in environments where this is not the case. However, because these tiles can induce IRTCs and the induced IRTCs tend to be short, its effectiveness is widespread. MRAT is also at risk to hinder learning if it is provided with incorrect or sub-optimal path(s) when placing tiles but this is inherent to many heuristic based solutions. There are also other possible heuristics that could be derived from the theory presented in this thesis.

Automated Mixed Resolution Acyclic Tiling

Whilst we performed an extensive study into different possible implementations of ARMAT further work into the automation of MRAT could explore more methods of implementing the two detection mechanisms that AMRAT uses. Moreover, other techniques could be incorporated or used in conjunction with AMRAT. For example, Chow and Tsitsiklis (1991) work on how to compute the tile width of a fixed uniform tiling necessary to learn an approximate optimal policy could be used to set a minimum tile size for AMRAT.

Further work into the analysis of AMRAT could include more environments. For example, environments with many more actions or continuous actions.

References

Akchurina, N. (2009). Multiagent Reinforcement Learning: Algorithm Converging to Nash Equilibrium in General-Sum Discounted Stochastic Games. In *In Proceedings of the Eighth International Conference on Autonmous Agents and Multiagent Systems (AAMAS)*.

Albus, J. S. (1981). *Brains, behavior, and robotics*. Byte books Peterborough, NH.

Ammar, H. B., Tuyls, K., Taylor, M. E., Driessens, K., and Weiss, G. (2012). Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems.

Andreae, J. (1963). STELLA: A scheme for a learning machine. In *In Proceedings of the 2nd IFAC Congress, Basle*, pages 497–506.

Babes, M., de Cote, E., and Littman, M. (2008). Social reward shaping in the prisoner's dilemma. In *In Proceedings of the Seventh Annual International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1389–1392.

Baroglio, C. (1995). Teaching by shaping. In *In Proceedings of the ICML-95, Workshop on Learning by Induction vs. Learning by Demonstration, Tahoe City, CA, USA*.

Basar, T. and Olsder, G. (1995). *Dynamic Noncooperative Game Theory*, volume 200. SIAM. Series in Classics in Applied Mathematics.

Bellman, R. E. (1957a). A Markov decision process. In *Journal of Mathematical Mechanics*, volume 6, pages 679–684.

Bellman, R. E. (1957b). Dynamic Programming. In *Princeton University Press*.

Bernstein, A. and Shimkin, N. (2010). Adaptive-resolution reinforcement learning with polynomial exploration in deterministic domains. *Machine learning*, 81(3):359–397.

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

Bowling, M. (2000). Convergence problems of general-sum multiagent reinforcement learning. In *Seventeenth International Conference on Machine Learning*, pages 89–94.

Bowling, M. (2004). Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17 (NIPS-04), Vancouver, Canada*, pages 209–216.

Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250.

Boyan, J. and Moore, A. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376.

Bradtke, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. In *Advances of Neural Information Processing 5, Morgan Kaufmann*, pages 295–302.

Brafman, R. I. and Tennenholtz, M. (2003). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231.

Brown, G. (1951). Iterative Solutions of Games by Fictitious Play. In *Activity Analysis of Production and Allocation, T.C. Koopmans (Ed.)*. Wiley.

Brys, T., De Hauwere, Y.-M., Nowé, A., and Vrancx, P. (2011). Local Coordination in Distributed Constraint Optimization Problems. In *The 9th European Workshop on Multi-Agent Systems, Maastricht, The Netherlands*.

Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015a). Ensembles of shapings. In *Proceedings of the Multi-Disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*.

Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015b). Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188. International Foundation for Autonomous Agents and Multiagent Systems.

Brys, T., Nowé, A., Kudenko, D., and Taylor, M. E. (2014). Combining multiple correlated reward and shaping signals by measuring confidence. In *AAAI*, pages 1687–1693.

Busoniu, L., Babuska, R., and De Schutter, B. (2008). A Comprehensive Survey of Multi-Agent Reinforcement Learning. In *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions*, volume 3. issue 2.

Chow, C.-S. and Tsitsiklis, J. (1991). An optimal one-way multigrid algorithm for discrete-time stochastic control. *Automatic Control, IEEE Transactions on*, 36(8):898–914.

Claus, C. and Boutilier, C. (1998). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *National Conference on Artificial Intelligence (AAAI-98)*, pages 746–752.

De Hauwere, Y.-M., Devlin, S., Kudenko, D., and Nowé, A. (2012). Context sensitive reward shaping in a loosely coupled multi-agent system. In *The 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*.

De Hauwere, Y.-M., Vrancx, P., and Nowé, A. (2010). Solving delayed coordination problems

in mas (extended abstract). In *The 10th International Conference on Autonomous Agents and Multiagent Systems, Toronto, Canada*, pages 1115–1116.

De Hauwere, Y.-M., Vrancx, P., and Nowé, A. (2011). Future Sparse Interactions: A MARL Approach. In *the European Workshop on Reinforcement Learning*, Athens, Greece.

Devlin, S. and Kudenko, D. (2011). Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The Tenth Annual International Conference on Autonomous Agents and Multiagent Systems*.

Devlin, S. and Kudenko, D. (2012). Dynamic Potential-Based Reward Shaping. In *The 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*.

Efthymiadis, K., Devlin, S., and Kudenko, D. (2012). Plan-Based Reward Shaping for Multi-Agent Reinforcement Learning. In *The 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*.

Farley, B. G. and Clark, W. A. (1954). Simulation of self-organizing systems by digital computer. In *IRE Transactions on Information Theory*, volume 4, pages 76–84.

Fernández, F. and Borrajo, D. (2008). Two steps reinforcement learning. *International Journal of Intelligent Systems*, 23(2):213–245.

Fudenberg, D. and Levine, D. K. (1996). The Theory of Learning in Games. Levine's Working Paper Archive 624, David K. Levine.

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

Grzes, M. and Kudenko, D. (2008b). Plan-based reward shaping for reinforcement learning. In *The 4th IEEE International Conference on Intelligent Systems (IS'08), IEEE*, pages 22–29.

Grzes, M. and Kudenko, D. (2009). Reinforcement Learning with Reward Shaping and Mixed Resolution Function Approximation. *International Journal of Agent Technologies and Systems (IJATS)*, 1(2):36–54.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

Harutyunyan, A., Brys, T., Vrancx, P., and Nowé, A. (2015). Off-policy reward shaping with ensembles. *CoRR*, abs/1502.03248.

Hennes, D., Kaisers, M., and Tuyls, K. (2010). RESQ-learning in Stochastic Games. In *The AAMAS 2010 Workshop on Adaptive Learning Agents and Multi-Agent Systems (ALA 2010), Toronto, Canada*.

Hu, J. and Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *The Fifteenth International Conference on Machine Learning*, volume 242, page 250.

Jin, Z., Liu, W., and Jin, J. (2009). Partitioning the state space by critical states. In *Bio-Inspired Computing, 2009. BIC-TA'09. Fourth International Conference on*, pages 1–7. IEEE.

Jung, T. and Stone, P. (2010). Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. *Machine Learning and Knowledge Discovery in Databases*, pages 601–616.

Kakade, S., Kearns, M., and Langford, J. (2003). Exploration in metric state spaces. In *ICML*, volume 3, pages 306–312.

Kapetanakis, S. and Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. In *In Proceedings of the National Conference on Artificial Intelligence*, pages 326–331.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232.

Klopf, A. H. (1972). Brain function and adaptive systems - A heterostatic theory. In *Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA*. A summary appears in Proceedings of the International Conference on Systems, Man, and Cybernetics. IEEE Systems, Man, and Cybernetics Society, Dallas, TX (1974).

Klopf, A. H. (1975). A comparison of natural and artificial intelligence. In *SIGART Newsletter*, volume 56, pages 11–13.

Klopf, A. H. (1982). The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence. In *Hemisphere, Washington, DC*.

Lau, Q. P., Lee, M. L., and Hsu, W. (2012). Coordination guided reinforcement learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 215–222. International Foundation for Autonomous Agents and Multiagent Systems.

Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163.

Littman, M. L. (2001a). Friend-or-Foe Q-Learning in General-Sum Games. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328.

Littman, M. L. (2001b). Value-function reinforcement learning in Markov games. *Journal of Cognitive Systems Research*, 2(1):55–66.

Littman, M. L. and Szepesvari, C. (1996). A generalized reinforcement-learning model: Convergence and applications. In *In Proceedings of the Thirteenth International Conference on Machine Learning*, pages 310–318.

Liu, C., Xu, X., and Hu, D. (2015). Multiobjective reinforcement learning: A comprehensive overview. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 45(3):385–398.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *In Proceedings*

of the 24th International Conference on Machine Learning, ACM, page 608.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *In Proceedings of the 11th ICML, Rutgers Univ., New Brunswick, NJ, USA*.

Mataric, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83.

Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, pages 664–671. ACM.

Minsky, M. L. (1954). *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem*. PhD thesis, Princeton University.

Minsky, M. L. (1961). Steps toward artificial intelligence. In *Proceedings of the Institute of Radio Engineers*, volume 49, pages 8–30. Reprinted in E. A. Feigenbaum and J. Feldman (eds.), Computers and Thought, pp. 406-450. McGraw-Hill, New York (1963).

Munos, R. and Moore, A. (2002). Variable resolution discretization in optimal control. In *Machine Learning*, volume 49, pages 291–323.

Nash, J. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49.

Nash, J. (1951). Non-Cooperative Games. *The 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 *In Proceedings of the 16th International Conference on Machine Learning*, pages 278–287.

Nouri, A. and Littman, M. L. (2009a). Multi-resolution exploration in continuous spaces. *Advances in neural information processing systems*, pages 1209–1216.

Nouri, A. and Littman, M. L. (2009b). Multi-resolution exploration in continuous spaces. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1209–1216. Curran Associates, Inc.

Nouri, A. and Littman, M. L. (2010). Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning*, 81(1):85–98.

Nunes, L. and Oliveira, E. (2002). On Learning by Exchanging Advice. In *In Proceedings of the First Symposium on Adaptive Agents and Multi-Agent Systems (AISB'02), Imperial College, London*.

Nunes, L. and Oliveira, E. (2003). Cooperative Learning Using Advice Exchange. In *Adaptive Agents and Multiagent Systems*, volume 2636, pages 33–48.

OED-Online (2014). artificial intelligence, n. Accessed: 2014-11-28. Available online at `http://www.oed.com/view/Entry/271625`.

Oxford-Dictionaries (2014). artificial intelligence. Accessed: 2014-11-28. Available online at `http://www.oxforddictionaries.com/definition/english/artificial-intelligence`.

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

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

RL-Library (2009). Cartpole (java). Accessed: 2016-05-07. Available online at `http://library.rl-community.org/wiki/CartPole_(Java)`.

Robinson, J. (1951). An Iterative Method of Solving a Game. In *Annals of Mathematics*, volume 54, pages 296–301.

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. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2nd edition.

Scopes, P. and Kudenko, D. (2014). Theoretical properties and heuristics for tile coding. In *ALA Workshop, AAMAS 2014*.

Seiler, M. C. and Seiler, F. A. (1989). Numerical recipes in c: the art of scientific computing. *Risk Analysis*, 9(3):415–416.

Sen, S., Sekaran, M., and Hale, J. (1994). Learning to coordinate without sharing information. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431.

Sherstov, E. A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In *of Lecture Notes in Artificial Intelligence*, pages 194–205. Springer Verlag.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent reinforcement learning: A critical survey, . In *Computer Science Dept., Stanford University, CA, US, Tech Rep.*

Singh, S., Kearns, M., and Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. In *In Proceedings 16th Conference on Uncertainty in Artificial Intelligence (UAI-00), San Francisco, US*, pages 541–548.

Skinner, B. F. (1938). The behavior of organisms: an experimental analysis. In *New York: D. Appleton Century*.

Smith, A. J. (2002). Dynamic generalisation of continuous action spaces in reinforcement learning: A neurally inspired approach.

Stone, P. and Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. In *In Proceedings of the third annual conference on Autonomous Agents*, pages 206–212.

Strehl, A. L., Li, L., and Littman, M. L. (2009). Reinforcement learning in finite mdps: Pac analysis. *The Journal of Machine Learning Research*, 10:2413–2444.

Strutz, T. (2010). *Data fitting and uncertainty: a practical introduction to weighted least squares and beyond*. Vieweg and Teubner.

Sutton, R. S. (1978a). Learning theory support for a single channel theory of the brain. Unpublished report.

Sutton, R. S. (1978b). Single channel theory: A neuronal theory of learning. In *Brain Theory Newsletter*, volume 4, pages 72–75. Center for Systems Neuroscience, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1978c). A unified theory of expectation in classical and instrumental conditioning. Bachelors thesis, Stanford University.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. In *Machine Learning*, volume 3, pages 9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044.

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

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

Taylor, M. E., Kuhlmann, G., and Stone, P. (2008). Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems.

Taylor, M. E., Kulis, B., and Sha, F. (2011). Metric learning for reinforcement learning agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 777–784. International Foundation for Autonomous Agents and Multiagent Systems.

Taylor, M. E. and Stone, P. (2004). Speeding up reinforcement learning with behavior transfer. In *AAAI 2004 Fall Symposium on Real-life Reinforcement Learning*.

Taylor, M. E. and Stone, P. (2005). Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59. ACM.

Taylor, M. E., Stone, P., and Liu, Y. (2005). Value functions for rl-based behavior transfer: A comparative study. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 880. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Thorndike, E. L. (1911). *Animal Intelligence*. Hafner, Darien, CT.

Turing, A. M. (1950). Computing machinery and intelligence. In *Mind*, volume 59, pages 433–460. Reprinted in E. A. Feigenbaum and J. Feldman (eds), Computers and Thought, pp. 11-35. McGraw-Hill, New York (1963).

van Hasselt, H. (2012). *Reinforcement Learning in Continuous State and Action Spaces*, pages 207–251. Volume 12 of Wiering and van Otterlo (2012).

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University.

Whitehead, S. D. (1991). A Complexity Analysis of Cooperative Mechanisms in Reinforcement Learning. In *In AAAI-91 Proceedings*, pages 607–613.

Whiteson, S., Taylor, M. E., and Stone, P. (2007). Adaptive Tile Coding for Value Function Approximation. Technical report, University of Texas at Austin. Technical Report AI-TR-07-339.

Wiering, M. and van Otterlo, M., editors (2012). *Reinforcement Learning: State-of-the-art*, volume 12. Springer.

Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19(1):205–208.

Wingate, D. (2004). Solving large mdps quickly with partitioned value iteration.

Witten, I. H. (1977). An adaptive optimal controller for discrete-time Markov environments. In *Information and Control*, volume 34, pages 286–295.

Wolfe, J. (1996). How to write a phd thesis. Accessed: June 2013. Available online at `http://www.phys.unsw.edu.au/~jw/thesis.html`.

Zhu, Y., Zhao, D., and He, H. (2014). An high-efficient online reinforcement learning algorithm for continuous-state systems. In *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, pages 581–586. IEEE.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In *In Proceedings Twentieth International Conference on Machine Learning (ICML-03)*, pages 928–936, Washington, US.

Ziv, O. and Shimkin, N. (2005). Multigrid methods for policy evaluation and reinforcement learning. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 1391–1396. IEEE.