Learning in a State of Confusion: Employing active perception and reinforcement learning in partially observable worlds

Paul Anthony Crook

Doctor of Philosophy

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

2006

Abstract

In applying reinforcement learning to agents acting in the real world we are often faced with tasks that are non-Markovian in nature. Much work has been done using state estimation algorithms to try to uncover Markovian models of tasks in order to allow the learning of *optimal* solutions using reinforcement learning. Unfortunately these algorithms which attempt to simultaneously learn a Markov model of the world and how to act have proved very brittle. Our focus differs. In considering embodied, embedded and situated agents we have a preference for simple learning algorithms which reliably learn *satisficing* policies. The learning algorithms we consider do not try to uncover the underlying Markovian states, instead they aim to learn successful deterministic reactive policies such that agents actions are based directly upon the observations provided by their sensors.

Existing results have shown that such reactive policies can be arbitrarily worse than a policy that has access to the underlying Markov process and in some cases no satisficing reactive policy can exist. Our first contribution is to show that providing agents with alternative actions and viewpoints on the task through the addition of *active perception* can provide a practical solution in such circumstances. We demonstrate empirically that: *(i)* adding arbitrary active perception actions to agents which can only learn deterministic reactive policies can allow the learning of satisficing policies where none were originally possible; *(ii)* active perception actions allow the learning of better satisficing policies than those that existed previously and *(iii)* our approach converges more reliably to satisficing solutions than existing state estimation algorithms such as U-Tree and the Lion Algorithm.

Our other contributions focus on issues which affect the reliability with which deterministic reactive satisficing policies can be learnt in non-Markovian environments. We show that that greedy action selection may be a necessary condition for the existence of stable deterministic reactive policies on partially observable Markov decision processes (POMDPs). We also set out the concept of Consistent Exploration. This is the idea of estimating state-action values by acting as though the policy has been changed to incorporate the action being explored. We demonstrate that this concept can be used to develop better algorithms for learning reactive policies to POMDPs by presenting a new reinforcement learning algorithm; the Consistent Exploration Q(λ) algorithm (CEQ(λ)). We demonstrate on a significant number of problems that CEQ(λ) is more reliable at learning satisficing solutions than the algorithm currently regarded as the best for learning deterministic reactive policies, that of SARSA(λ).

This thesis is dedicated to “Freddy” Flintoff, Simon Jones, Ashley Giles, Matthew Hoggard, Steve Harmison, Andrew Strauss, Kevin Pietersen, Michael Vaughan, Marcus Trescothick, Geraint Jones, Ian Bell, Paul Collingwood, Gary Pratt, Shane Warne, Brett Lee, Glenn McGrath, Shaun Tait, Justin Langer, Matthew Hayden, Adam Gilchrist, Ricky Ponting, Michael Clarke, Simon Katich, Damien Martyn, Jason Gillespie, Billy Bowden, Rudi Koertzen and everyone else involved in the 2005 Ashes series, for providing entertainment and nail biting drama over the summer whilst I was writing this thesis. To TMS for relaying all the action in their intimate and whimsical style, and who will always conjure up images of idealised English summers in my head. To the idiosyncratic game that is cricket, and to England for winning the Ashes.

Acknowledgements

Many thanks to my supervisors; Gillian Hayes for all her ideas and input, and Bob Fisher for his constructive criticism and advice. Thanks to Jay Bradley, Aaron Crane, Seb Mhatre, Fiona McNeill, and Daniel Winterstein, who read various parts of this thesis. Thanks to Peter Ottery for writing the first version of taskFarmer, a program which was instrumental in spreading the computational load of many experiments in this thesis. Special thanks to my study buddy Alison Pease, who not only read parts of this thesis, but had to put up with my whinging whilst I was writing it. Thanks also to my examiners Sethu Vijayakumar and Eric Postma whose comments and criticisms helped to shape the final version of this thesis. Finally my biggest debt of gratitude is to all my friends and flatmates, without whom this period of research and study in Edinburgh would not have been half as much fun.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Paul Anthony Crook*)

Publications

The material contained in chapter 4 of this thesis has been previously published in conference and workshop proceedings:

- The Sixth European Workshop on Reinforcement Learning (EWRL-6), Nancy, France, September 2003 [Crook and Hayes, 2003a].
- 14th European Conference on Machine Learning (ECML 2003), Cavtat-Dubrovnik, Croatia, September 2003. Published in Springer Verlag Lecture Notes in Artificial Intelligence, volume 2837, pages 72–83 [Crook and Hayes, 2003b].
- Towards Intelligent Mobile Robots (TIMR) 2003, UWE, Bristol, August 2003 [Crook and Hayes, 2003c].

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Any embodied, embedded and situated agent will, by virtue of its body and its location in the world, have limitations on what it can perceive. Often information which would be useful to the agent's decision-making processes can be hidden, but despite this the agent needs to learn how to achieve its goals, and learn in a reliable manner. One way to simplify the agent's task is to enhance its abilities. This could be by providing it with memories of past events; alternatively, as it is embedded in a world, the ability to manipulate its own perceptions may help it find the information it requires.

A popular approach to learning policies with embodied, embedded and situated agents is to use reinforcement learning. The term *reinforcement learning* covers a broad range of learning algorithms which share the common attribute that they are neither left to their own devices to find structure in the data (as in the case of *unsupervised learning*) nor presented with sets of data marked up with the correct answer (as for *supervised learning*). Instead, reinforcement learning algorithms are allowed to attempt some task, and during their attempt they intermittently receive penalties or rewards which indicate the cost of their actions, or their achievement of some goal.

In the field of applying reinforcement learning to agents whose perception of the world is limited, much work has been done with agents enhanced with various forms of memory. These enhancements range from the simple equivalent of being able to tie

a knot in a handkerchief, to sophisticated learning algorithms that analyse an agent's stream of past experiences. We consider an alternative approach inspired by the ability of many creatures to manipulate their own perceptions of the world. We consider agents with *active perception*: the ability of an agent to select actions that manipulate its perception of the world (for more detailed definition see section 2.1). For this type of agent the small amount of work that has been done suggests that little benefit is gained from active perception unless it is coupled with learning algorithms specifically design to handle the active perception actions (or *perceptual actions* as we refer to them in the remainder of this thesis). In this thesis we aim to change this impression by showing that (i) equipping agents with active perception is an effective approach to learning in partially observable worlds, and (ii) the only specialisation required of the learning algorithm is not a need to handle perceptual actions separately, but the ability to converge on *satisficing*[1] policies in partially observable worlds.

The following section motivates our work giving more details of the issues that this thesis addresses. We then set out our aims for this thesis, the contributions made by this work and the organisation of the rest of this thesis.

1.1 Motivation

Research into using reinforcement learning to find optimal solutions to tasks where only partial information is available – partially observable Markov decision processes (POMDPs) – has traditionally focused on augmenting agents or learning algorithms with memory or the ability to build internal models of the world, for example, work by Chrisman [1992b]; McCallum [1993]; Peshkin et al. [1999]; Lanzi [2000]; Aberdeen [2003]. Some work has been undertaken where the agents are augmented with perceptual actions; such work was reviewed by Whitehead and Lin [1995]. The active perception approaches proposed, which are described in more detail in section 3.1, employed

[1] We uses the word 'satisficing' in the sense of achieving the goal, but not necessarily in an optimal fashion. This usage originates from work by Simon [1956] "Evidently, organisms adapt well enough to 'satisfice'; they do not, in general, 'optimize'." Refer to section 2.2.1 for a more detailed definition.

reinforcement learning algorithms specifically created to handle active perception actions. These perceptual actions were clearly segregated from overt or physical actions which change the state of the world, in part because they were conceived of as a different type of action, but mainly because such actions have the potential to create an aliasing of states [Whitehead, 1992]. That is to say, when using perceptual actions, one state of the world can map onto many observations, and one observation can map onto many possible underlying states. If an agent entirely relies on the observation it receives from the world when selecting the next action to execute, then the belief of [Whitehead, 1992; Tan, 1991c] is that the many-to-many mapping of observations to world states caused by perceptual actions had to be handled using specially designed learning algorithms, in order to make policy learning possible.

Such non-bijective mappings between states and observations are, however, not confined to agents with active perception. Any agent with limited perception of its environment suffers from a very similar problem in that many states of the world look identical from its point of view. [Loch and Singh, 1998] showed this many-to-one mapping can be handled successfully by reinforcement learning algorithms that use eligibility traces. An eligibility trace is a record of previous observations and actions as executed by the agent. This trace is used to distribute recent rewards received by the agent, to actions selected several time steps ago. The aim of eligibility traces is to reward actions which at the time of execution received no immediate reward, but may have contributed to the receipt of the current reward. Work by Loch and Singh [1998] showed that the use of eligibility traces allowed reinforcement learning to find optimal policies, within the limits of agents' abilities, in grid world navigation problems where agents are unable to differentiate between many of the grid world locations. These policies consist of learnt associations between the observations the agents receive and the best action to select. Such policies are known variously as *reactive policies* or *memoryless policies* – as in policies learnt by agents with no memory. We generally prefer the term *reactive*, which we consider provides a more accurate description, that is, the next action is selected by the agent's learning algorithm, based purely on the information it is currently receiving.

Our conjecture is that, if using eligibility traces allows agents with limited perceptions

to learn optimal reactive policies, then eligibility traces should also allow agents enhanced with active perception to learn optimal reactive policies, despite changing the many-to-one mapping to a many-to-many mapping. The quality of the optimal reactive policy is governed by the abilities of the agent, that is, the more underlying states an agent can distinguish between, the closer it can get to the optimal solution which would be achieved by an agent which has access to unambiguous perceptions which uniquely identify the current state of the world. Given this, enhancing the abilities of an agent by providing it with active perception should increase the quality of the policies that it can learn.

For several reinforcement learning algorithms, theoretical performance guarantees have been shown for *Markovian* worlds [Watkins and Dayan, 1992; Dayan, 1992; Singh et al., 2000], that is, worlds where the information observed by the agent is sufficient at every step to select the best action. In introducing active perception, tasks will generally become non-Markovian even if they were not before, and as such, only limited theoretical guarantees on convergence have been shown; see section 3.2 for more details. Thus an essential part of this thesis is a study of the reliability with which reinforcement learning algorithms converge to satisficing, reactive policies on partially observable tasks.

Overall our work is of interest in the area of reinforcement learning as:

(i) many problems can be seen as tasks of the form that we are attempting to solve, in that they require the selection of actions based on the observed state of partially observable worlds, for example, unlocking a door or navigating through a building;

(ii) even if you apply techniques that try to learn mappings between the observations an agent receives from a partially observable world and a Markovian representation of that world (for example, algorithms presented in in section 3.1), there are no guarantees that the output of such a mapping will always be Markovian. Thus it is useful to understand the requirement for reliably learning with aliased

states[2] to improve the performance of such techniques;

(iii) any method of state aggregation which is used to reduce the number of states that an agent observes and thus simplify learning (for example, linear state function approximation, state-aggregation or clustering techniques) produces the same problems of aliasing, and thus should be amenable to similar solutions.

1.2 Thesis Aims

1.2.1 First hypothesis

The main hypothesis of this thesis is that **in the field of reinforcement learning, equipping an agent with active perception is an effective approach to learning in partially observable worlds**, where *active perception* is defined as the ability of an agent to select actions that manipulate its perception of the world, and *effective* implies that this approach performs in general no worse[3] than others presented in the literature (assuming that the other methods actually have some positive benefit).

The evidence that we provide in support of the first hypothesis is:

(i) that the addition of active perception can allow an agent to learn satisficing, deterministic, reactive policies for partially observable tasks, where satisficing, stationary, deterministic policies could not have been expressed before,

(ii) a demonstration that enhancing an agent with active perception will allow it to learn satisficing reactive policies that are better, according to some quality measure, for example, the number of physical steps taken to reach the goal, than the optimal reactive policy achievable by an non-enhanced agent,

(iii) the presentation of techniques that improve the reliability of learning satisficing, deterministic, reactive policies in partially observable worlds,

[2] See section 3.1.1 for a definition of *aliased states*.
[3] Including the cost of the added perceptual actions.

(iv) an empirical demonstration that the performance of active perception agents is comparable with other approaches in the field of reinforcement learning.

The techniques mentioned in item (iii) support our hypothesis because we are considering the performance of active perception in the context of other reinforcement learning approaches. Part of our approach relies on the learning of reactive policies in partial observable domains. This means that any improvements we can achieve in this area will aid our active perception approach in comparison to other non-reactive reinforcement learning approaches that have been developed for learning in partial observable domains, for example, approaches which build internal models which attempt to map partially observable tasks onto a Markovian representation. When working with embodied agents, the need to reliably learn *satisficing* policies is emphasised, possibly at the expense of optimal solutions. It is generally more important that most, if not all, agents succeed, than that a few learn optimal policies while others fail.

1.2.2 Second hypothesis

Our second hypothesis is that **there is no requirement to manage perceptual actions separately through the provision of specialised reinforcement learning algorithms**. In other words, provided the reinforcement learning algorithm used performs well in partially observable domains, there is no need to distinguish between what is a perceptual action and, for example, physical actions that move objects in the world.

The evidence provided in support of the second hypothesis is:

(i) in comparing the quality of policies learnt by reinforcement algorithms, no benefit is gained through the employment of an algorithm that distinguishes between perceptual and physical actions,

(ii) that in all experiments in this thesis where agents equipped with active perception are used, that they have learnt good, satisficing, reactive policies without using algorithms that distinguish between the two types of action.

1.3 Contributions

We summarise the contributions of this thesis in the order of the chapters in which they occur. With the exception of the first item listed in section 1.3.2, which is a useful confirmation of a previous result, all the remaining contributions listed below are completely original work by the author.

1.3.1 Chapter 2

- We present a new classification scheme for the extension of agent designs, the three classes that we define are *fixed-SE approach*, *modifiable approach* and *basic approach*. These classes help us distinguish between approaches which assume that a task has to be rendered Markovian before applying reinforcement learning to it, and approaches which assume that satisficing reactive policies can be found despite the non-Markovian nature of the task.

1.3.2 Chapter 4

- We replicate and confirm results observed by Loch and Singh [1998] that in partially observable environments where an agent has limited perception which produces a many-to-one mapping between states and observations, reinforcement learning algorithms which use eligibility traces can learn optimal reactive policies.
- We extent the above research by demonstrating that the results hold for many-to-many mappings between states and observations. Thus showing that reinforcement learning algorithms which use eligibility traces can be applied successfully to agents with active perception systems.
- Using two types of idealised active perception systems we show that it is sufficient for an active perception system to provide a unique reference for each aliased location, and that further information is not necessarily productive.

- We show that reinforcement learning with eligibility traces can be successfully extended to agents enhanced with the addition of a more realistic implementation of active perception. This demonstrates the ability to learn good reactive policies in situations with many-to-many mappings between observations and states of the world. These agents also demonstrate that they are capable of learning better policies than non-enhanced agents.

- Finally we demonstrate that an agent which is enhanced with active perception can learn satisficing, deterministic, reactive policies in partially observable worlds, where for a non-enhanced agent, no satisficing policies were possible.

1.3.3 Chapter 5

- We suggest that the cessation of exploratory behaviour by agents might be a necessary condition for the formation of satisficing, deterministic, reactive policies in POMDP worlds.

1.3.4 Chapter 7

- We generalise the idea of Consistent Exploration from the previous chapter, such that it can be applied to temporal difference (TD) algorithms, and then develop a new TD algorithm, Consistent Exploration Q(λ) ($CEQ(\lambda)$).

- Results in chapter 7 and in the following chapter 8 demonstrate that CEQ(λ) outperforms SARSA(λ), in terms of the reliability with which policies converge to satisficing solutions, on many partially observable problems. This is a significant result as SARSA(λ) is acknowledged as a "high standard to compare to," [Perkins, 2002b] and, in Loch and Singh [1998], "that SARSA(λ) may be hard to beat in problems where there exists a good policy that maps observation space to actions."

1.3.5 Chapter 8

- Reinforcement learning tends to be a field in which there is a proliferation of learning algorithms, either derived to tackle specific issues, or borne out of good theoretical analysis. Empirical evidence, however, tends to be rather scant, and is generally restricted to the one example task. In addition, there is often very little overlap in the choice of tasks used. This section attempts to partially address this lack of empirical evidence by contributing a comparison of four reinforcement learning algorithms across a set of identical problems. We undertaking sufficient runs to offer sound statistical comparisons on their relative performance on these tasks.

- Through empirical comparisons with a range of alternative reinforcement learning approaches, we demonstrate that active perception is an effective approach for learning in partially observable worlds, and easily out performs the state-estimation approaches that were tested.

- Using a simulated robot navigating a set of corridors we demonstrate that both the active perception approach, and the reinforcement learning algorithm CEQ(λ), can be successfully applied to tasks in complex, stochastic environments.

- We show, additionally, that the state-estimation algorithm U-Tree [McCallum, 1995b, 1996] performs poorly in practise.

  (i) On fifty percent of the problems tried, U-Tree failed to form any satisficing policies.

  (ii) U-Tree is designed to learn an internal model in order to distinguish between all utile states. The processes used to extend this model are computationally expensive and the run-time of U-Tree increases supra-linearly, both as it gains experience, and also as it adds to its internal model.

  (iii) Storing every action, observation and reward in an instance chain eventually causes the algorithm to fail due to lack of memory.

(iv) We observe that U-Tree over-extends its internal model when trying to cope with clusters of aliased states, especially where cyclic paths between such states are possible.

The above results are significant as the U-Tree algorithm is often mentioned in the literature as being a good approach to learning in partially observable domains.

1.3.6 Appendix A

- Appendix A provides a summary of the best performing parameter values found for each combination of reinforcement learning algorithm and task that appears in this thesis. It is hoped that this summary will provide a useful guide to others in selecting parameter values.

1.3.7 Reinforcement learning test platform

A practical contribution of this work is the creation of a reinforcement learning test platform which can be used to experiment with alternative agents, worlds and learning algorithms. We have made the code for this test platform publicly available under the terms of the GNU General Public License [Free Software Foundation, 1991]. It can be downloaded from: `http://homepages.inf.ed.ac.uk/s9903543/PhD/code/`

The code is written in a modular fashion which allows different agents, worlds, learning algorithms and parameter settings to be combined with each other for the purpose of running experiments. Experiments are specified and initiated from the command line making it straightforward to run batches of experiments. The structure adopted, combined with object-oriented inheritance, makes the addition of new learning algorithms, grid worlds or agents relatively simple. A separate body of code is provided which integrates the same reinforcement learning algorithms with Webots simulation software [Cyberbotics, 2005] (not provided), allowing reinforcement learning experiments to be run on simulated robotic platforms.

The code was developed exclusively under Linux using Sun's J2SE 1.4.2 version of Java. It should work without any changes on other Unix-like operating systems, such as BSD. Although it is written in Java and thus should run under any suitable Java virtual machine, porting the code to Microsoft's Windows operating system may require some modifications, especially in the file names used.

1.4 Organisation of this Thesis

1.4.1 Main body of thesis

- Chapter 1, **Introduction**, provides our motivations for pursuing this area of work, the hypothesis that we set out to test in this thesis and outlines the contributions achieved.

- Chapter 2, **Background**, provides (i) an introduction to the topics of active perception and reinforcement learning, (ii) clarifies our interpretation of various terms (active perception, direct reinforcement learning, and state estimation), (iii) sets out the definition of three classes which describe different approaches to extending the design of reinforcement learning agents (*fixed-SE*, *modifiable* and *basic approaches*).

- Chapter 3, **Literature Review**, conducts a selective review of previous work undertaken in this area. The review is structured around the approaches to extending agent designs that were identified in chapter 2; namely *fixed-SE*, *modifiable* and *basic approaches*.

- Chapter 4, **Learning with Active Perception**, initially focuses on reproducing existing results from the literature. These results concern the ability of 1-step backup reinforcement learning algorithms and reinforcement learning algorithms which use eligibility traces to learn stable, reactive policies in tasks where a many-to-one mapping exists between states of the world and the agent's

observations. We then extend these results for (i) agents that have access to idealised active perceptual systems (referred to as oracles), and (ii) agents using a more realistic set of active perceptual actions. The results for (ii) demonstrate that the use of eligibility traces to learn good, satisficing reactive policies can be extended to tasks where many-to-many state-to-observation mappings exist.

- Chapter 5, **Effects of Action Selection**, is the first of two chapters that consider the reliability with which reinforcement learning algorithms converge to satisficing, reactive policies on POMDP tasks. This chapter considers the effect of action selection on convergence – specifically, the effect of exploratory actions on convergence.

- Chapter 6, **Monte Carlo Approaches and Consistent Exploration**, contrasts the reliability with which Monte Carlo style algorithms and temporal difference (TD) style algorithms converge on POMDP tasks. This chapter also draws out the idea of Consistent Exploration.

- Chapter 7, **Consistent Exploration Approach**, introduces a new TD algorithm which is developed from Watkins's Q(λ) by applying the idea of Consistent Exploration. The new algorithm, Consistent Exploration Q(λ), or CEQ(λ), is compared for reliability of convergence with one of the best performing algorithms for learning reactive policies on POMDP tasks; SARSA(λ).

- Chapter 8, **Comparison of Approaches**, extends the range of problems over which we compare the reliability of convergence of CEQ(λ) and SARSA(λ) as well as returning to our starting point of considering active perception. This chapter compares the performance of our active perception approach with reinforcement learning approaches that (i) use internal memory bits which are manipulated by the agent, (ii) attempt to build internal models of the environment, specifically U-Tree [McCallum, 1995b, 1996], (iii) explore the perceptual actions available, with the aim of finding a consistent representation for each state [Whitehead, 1992]. We test learning in both deterministic and stochastic environments.

- Chapter 9, **Evaluation and Discussion**, evaluates the results of the previous chapters as well as setting out other, non-experimental arguments in support of our thesis. Specifically it examines (i) the advantages of active perception over other reinforcement learning approaches that have similar aims, (ii) discusses the structure of the active perception agents and reinforcement learning algorithms that were found to be most productive in this thesis, (iii) considers the wider applicability of our results.

- Chapter 10, **Future Work**, sets out many directions in which this work could be pursued. One of the more important ideas arising in this chapter is that the problem of learning good, satisficing policies to partially observable tasks, should be regarded as a search for Markovian observation-action paths through a non Markovian space. This is a viewpoint that we believe to be novel and may suggest new algorithms or new ways of proceeding.

- Chapter 11, **Conclusions**, presents a summary of what has been achieved in this thesis and reviews the initial hypotheses as presented in section 1.2.

1.4.2 Appendices

There are sixteen appendices which provide further details of some the learning algorithms used in this thesis.

- Appendix A, **Best Parameter Values Found**, summaries the best parameter values found for each learning algorithm on the various tasks tested in this thesis.

- Appendix B, **Q-learning**, presents the 1-step backup, reinforcement learning algorithm, Q-learning.

- Appendix C, **SARSA**, presents the 1-step backup, reinforcement learning algorithm, SARSA.

- Appendix D, **SARSA(λ)**, presents the reinforcement learning algorithm SARSA(λ) which is a development of SARSA through the addition of eligibility traces.

- Appendix E, **Watkins's Q(λ)**, presents the reinforcement learning algorithm Watkins's Q(λ) which is a development of Q-learning through the addition of eligibility traces.

- Appendix F, **Modified Watkins's Q(λ)**, presents a modified form of the Watkins's Q(λ) algorithm. The standard version of Watkins's Q(λ) can also be found in this appendix.

- Appendix G, **Supplemental Notes on U-Tree Implementation**, outlines the U-Tree algorithm, and provides supplementary notes aimed at: (i) addressing those parts of the specification [McCallum, 1995b, 1996] which we found to be inadequate, (ii) detailing our implementation.

- Appendix H, **Implementation of Lion Algorithm**, presents the Lion Algorithm along with details of our implementation.

- Appendix I, **Examples of Non-Convergence to a Static Policy for $\varepsilon > 0$**, presents a theoretical analysis which demonstrates that CEQ(λ), CEQ(λ)$^+$ and Monte-Carlo-with-exploring-starts cannot be guaranteed to converge to a single deterministic policy for $\varepsilon > 0$.

- Appendix J, **CEQ(λ)$^+$**, presents a version of CEQ(λ) where the probabilities used in selecting the next action take into account the current modified policy.

- Appendix K, **CE-SARSA(λ)**, presents a Consistent Exploration version of SARSA(λ).

- Appendix L, **Random Grid World Generation**, presents the algorithm used to generate the random grid worlds which were used in chapter 8.

- Appendix M, **Breakdown of figures 5.8(a) and 5.8(b)**, detail behind plots presented in on page 174 in section 5.2.2. Plots are shown for *each* combination of learning algorithm and action selection parameters.

- Appendix N, **Detailed Comparison of Mean Total Steps for Section 7.1.3**, a detailed comparison of the quality of the satisficing policies learnt by CEQ(λ) and SARSA(λ) broken down by *all* parameters combinations tested.

- Appendix O, **Breakdown of Mean Number of Satisficing Policies for Section 8.1**, presents the mean numbers of satisficing policies learnt for *each* combination of parameter values tried.

- Appendix P, **Profile of Sensor Models for Simulated Robot Experiment**, sets out details of the infra-red and sonar sensor models which were used in simulated robot experiment.

Chapter 2

Background

This chapter provides a brief introduction to the topics of active perception and reinforcement learning, including the concepts of Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs). We clarify our interpretation of various terms, such as active perception, direct reinforcement learning, and state estimation, and also introduce other useful terminology.

2.1 Active Perception

The term *active perception*, as used in the Artificial Intelligence (AI) community, has its origins in a paper of that name presented by Bajcsy [1988]. There was growing recognition around the time this paper was written that the old AI assumption of information flowing one way, from the world through passive sensors, was flawed. The then new, active perception paradigm is beautifully summarised by Bajcsy [1988]:

> "It should be axiomatic that perception is not passive, but active. Perceptual activity is exploratory, probing, searching; percepts do not simply fall onto sensors as rain falls onto ground. We do not just see, we look. And in the course, our pupils adjust to the level of illumination, our eyes bring the world into sharp focus, our eyes converge or diverge, we move our heads

> or change our position to get a better view of something, and sometimes we even put on spectacles."

Bajcsy [1988] emphasises the *intelligent control* of sensors to maximise the information obtained while at the same time taking into account the current interpretation of the data and the goals of the task. Thus he defines active perception as "an intelligent data acquisition process." This interpretation is contrasted with that of Aloimonos et al. [1987] who considered active perception (and specifically active vision) from the viewpoint of known motions providing additional views, and thus simplify the data processing problem. The difference in emphasis lies in whether the movement (i) was intelligently directed by the agent in response to its stimulus and goals, or (ii) either was deemed useful *a priori* by the designer of the agent, or just happened to occur and the vision is profiting from this happenstance. We note that Bajcsy [1988] uses the terms *active perception* and *active vision* interchangeably when talking about vision, with active vision being a specialisation of active perception that considers the visual sense.

The claim that vision is not a one way transformation from two dimensional visual data into a description of three dimensional spatiotemporal worlds is developed by Ballard [1991]. Ballard [1991] argues "that vision is more readily understood in the context of the visual behaviors that the system is engaged in, and that these behaviors may not require elaborate categorical representations of the 3-D world." In emphasising the view that a complex internal representation of the world is not required, this paper makes a strong link between the use of active perception in behaviours perceived to be intelligent, and the behaviourist views of Brooks [1986, 1991].

The term *animate vision* is preferred by Ballard [1991] as it clearly distinguishes this paradigm from the term *active vision* which, as mentioned above, has been associated with multi-modal fusion [Aloimonos et al., 1987]. It also reduces confusion with the concept of *active sensing*. Active sensing generally implies use of active sensors; sensors which emit energy to obtain information (for example, radar or laser range finders) in contrast to passive sensors (for example, microphones or video cameras) which simply collect information that falls onto them. If we were to follow Ballard [1991]'s logic

then the area in which our work falls should more correctly be referred to as "animate perception", as we follow the ideas set out by Ballard [1991], but are not restricting ourselves to vision. However, we prefer to adopt the phrase "active perception" as it has already been used in the reinforcement learning literature in the sense that we intend – for example by Whitehead and Ballard [1990a]. To summarise, the concept that we intend to capture by the phrase "active perception" is that of intelligent control toward achieving goals, as expressed by Bajcsy [1988], combined with a behaviourist view of sensing as expressed by Ballard [1991].

Work on active perception has been pursed within and across many disciplinary boundaries:

- **machine vision**, where it is referred to mainly as active vision [Blake and Yuille, 1992; Aloimonos, 1993; Rao and Ballard, 1995],
- **psychology**, for example see Findlay and Gilchrist [2003],
- **robotics**, in the form of visual servoing and navigation. An example of work in this area is provided by Balkenius and Hulth [1999] who developed a robot capable of memorising the relationship between perceptions obtained through control of its camera and physical motions of its body. It was thus possible to teach it visually cued behaviours, such as navigation of an office.

However, the main discipline which is relevant to this thesis is the field of **reinforcement learning**. A review of existing work on combining active perception with reinforcement learning is given in chapter 3.

2.2 Reinforcement Learning

For readers unfamiliar with reinforcement learning, Sutton and Barto [1998] provide a very good introduction to the subject. In brief, reinforcement learning is distinguished

from other machine learning techniques by (i) the requirement that the learning algorithm has to use trial-and-error in order to search out what actions it should select, and (ii) the fact that the rewards may be delayed and thus attributable to earlier actions (or possibly sequences of actions) which the learning algorithm executed. Of necessity, we assume through the remainder of this thesis, that readers are familiar with common terms used in reinforcement learning, such as policy, state, state value, state-action values, *Q*-values, etc.

Much of the formalism adopted for reinforcement learning, as well as some of the theoretical results, can be traced back to the older fields of Operations Research (OR) and Optimal Control. Two useful formulations from these areas are *Markov decision processes* (MDPs) and *partially observable Markov decision processes* (POMDPs). These two formulations, which characterise the worlds in which learners operate, are described in detail in sections 2.2.2 and 2.2.3.

Section 2.2.1 sets out what we mean when we refer to policies learnt by reinforcement learning algorithms as being *optimal*, *satisficing* or *non-satisficing*. Sections 2.2.4 and 2.2.5 discuss two further concepts which are particular to reinforcement learning. The two concepts are:

(i) the distinction between learning algorithms that have access to the full dynamics of the environment (or a perfect model of the environment), and those that do not,

(ii) the distinction between agents that learn internal models of the environment, either to disambiguate states or allow planning, and those agents which do not learn an internal model.

Unfortunately the term *model-free* has been used in conjunction with both the above concepts. Aberdeen and Baxter [2002] uses the terms *model-based* and *model-free* in discussing concept (i), while Sutton and Barto [1998, p.230] uses the term *model-learning* when referring concept (ii), as a result of which *model-free* has been used to refer to agents which do not learn internal models [Singh, 2004]. Fortunately, as this

thesis considers only learning algorithms that do *not* have access to the full dynamics of the environment, we do not need to distinguish between *model-based* and *model-free* as used in the sense of concept (i). With regard to concept (ii), we consider it useful to make finer distinctions than whether or not an agent learns an internal model. The terminology that we use in this case is set out in section 2.2.5.

2.2.1 Policy types

In the main, research into machine learning has focused on the ability of techniques to learn *optimal policies*, that is, policies which achieve the minimisation or maximisation of some mathematical function. Typically in reinforcement learning the function to be optimised represents not just the achievement of some desired outcome, but enumerates the quality of the policy which achieves that outcome. For example, the function considered when training an agent who's aim is to reach a given goal is typically not a binary function which indicates whether or not the agent will eventually reach the goal, but instead will be a function of the path length traversed by the agent in order to reach the goal. Thus, in this case, the optimal policies are those which minimise the path length.

Although attempting to minimise some function, such as path length, may be a useful way of guiding learning, one's primary interest may not be in the path length but in whether or not a goal is achieved. In this case one is not strictly interested in optimal policies but instead in policies that, in the words of Simon [1955], reach some minimal "aspirational level". It is in this sense that we use the label of *satisficing policies*. This definition is in agreement with and based upon the usage of the same phrase in Littman [1994]. It is also worth noting that under this definition *optimal* and *satisficing policies* are not disjunct sets. Optimal policies are in fact a subset of satisficing policies.

We note that there are alternate interpretations of the word satisficing, especially in the cybernetics literature [Kaufman, 1990; Goodrich et al., 1998; Stirling et al., 2002]. In this field satisficing has been defined not in terms of achieving some minimal aspirational level, but in terms of the best which can be achieved given limited computational

resources or in Simon [1956] word's "a bounded rationality" . We dislike this definition as the practical interpretation of "best that can be achieved" is still one of seeking optimality, the only difference is that the function to optimise incorporates addition costs or constrains.

Given our definition of a satisficing policy to mean a policy that achieves a given goal. A *non-satisficing* policy is any policy that fails to achieve the required goal.

2.2.2 Markov decision processes (MDPs)

A Markov decision process (MDP) is a convenient formulation for describing a world that has the Markov property. An agent operating in a Markovian world perceives a series of states each of which contains all the information that the agent needs in order to decide what action to execute next. An example is the game of Go where the current position of the board contains a summary of all the information required to decide subsequent moves. The history of how the board came to be in that state is not required in order to be able to play effectively[1].

An MDP can be represented visually as shown in figure 2.1. Each circle represents the state of the world as perceived at that point. Each arrow represents a transition between the states as a result of selecting an action. For those familiar with *Markov chains*, MDPs can be seen as a simple extension where the transitions in the chain are conditioned on the action selected by the agent. Thus, as Markov chains are special forms of Bayesian Networks [Russell and Norvig, 2003], the observation of the current state (for example, s_2 in figure 2.1) renders the probability of transition to any of the future states (s_3, s_4, ... s_N) independent of the past states (s_0, s_1). That is to say, events that occurred in the past have no effect on the future beyond determining the state that you are currently in. It is this property that allows theoretical guarantees to be given for reinforcement learning on MDPs, for example, convergence to optimal, stationary policies [Watkins and Dayan, 1992; Singh et al., 2000; Dayan, 1992].

[1] The order of play of the stones does contain additional information on the strategy that an opponent is playing. This information might provide guidance as to where to concentrate plays, but it is not essential in determining what move to play next if following a min-max strategy.

Mathematically a MDP is fully described by:

- the set of states that can occur, $\mathcal{S}$,
- the set of actions available to the agent given the current state, $\mathcal{A}(s)$ where $s \in \mathcal{S}$,
- a set of transition probabilities $\mathcal{P}_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ and $s' \in \mathcal{S}$. That is, the probability of the transition from a given state s, to another state s' given a selected action a,
- a set of expected immediate rewards, $\mathcal{R}_{ss'}^{a} = E\{r_{t+1} \mid a_t = a, s_t = s, s_{t+1} = s'\}$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ and $s' \in \mathcal{S}$.

Thus the tuple $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{P}_{ss'}^{a}, \mathcal{R}_{ss'}^{a} \rangle$ fully describes a MDP.

2.2.3 Partially observable Markov decision processes (POMDPs)

A partially observable Markov decision processes (POMDPs) is a formalisation that describes worlds where the perceptions seen by the agent do not fully describe the state of the world. Instead the observations obtained are related probabilistically to the underlying states. A visual representation of a POMDP is shown in figure 2.2.

In the same way that MDPs can be seen as an extension of Markov chains, POMDPs can be seen as an extension of *hidden Markov models* (HMMs) where the transitions are conditioned by the actions selected. POMDPs are inherently more complex models to analyse than MDPs: perceiving the current observation (say o_2 in figure 2.2) does not result in any independence between previous and subsequent states of the world, that is, state s_N remains dependent on state s_0 even when states o_1, o_2, etc. are observed. Unfortunately this means that the convergence guarantees that exist when applying reinforcement learning to MDPs cannot be extended to POMDPs. The few theoretical guarantees that currently exist are discussed further in sections 3.2.2 – 3.2.5 and at the start of chapter 5.

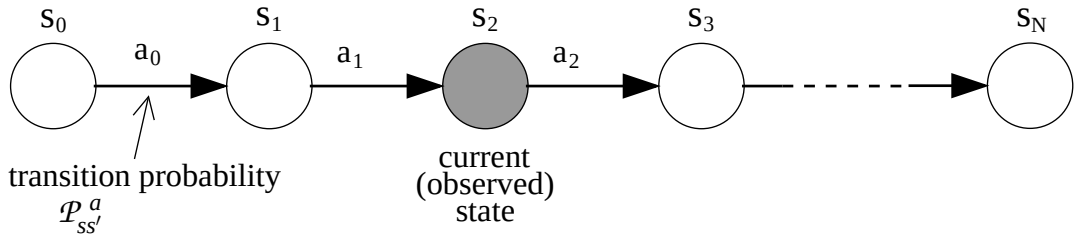


Figure 2.1: Graphical illustration of a Markov decision process (MDP). Direct observation of the current state of the system is possible. In observing a state, for example s_2 (shaded grey), the probability of transition to the following state (s_3) becomes independent of previous states (that is, s_0 or s_1) [Russell and Norvig, 2003]. The process is thus said to be Markovian in that the best action in the current state is independent of past states

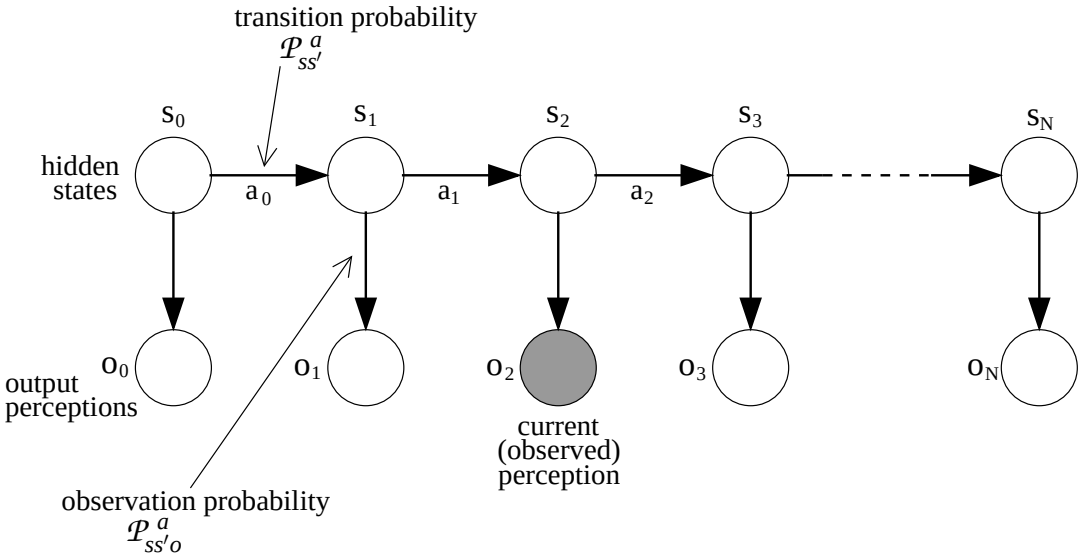


Figure 2.2: Graphical illustration of a partially observable Markov decision process (POMDP). Direct perception of the current state of the system is not possible. It is only possible to perceive the output observation, which is related stochastically to the current state. As the current state is not observed directly, the future states of the system are not independent of preceding states

Mathematically a POMDP is fully described using the same elements as required to describe a MDP but with the addition of a set of observations, Ω, and a set of probabilities of obtaining given observations, $\mathcal{P}^{a}_{ss'o} = \Pr\{o_t = o \mid s_{t-1} = s, s_t = s', a_{t-1} = a\}$ for all observations $o \in \Omega$, $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ and $s' \in \mathcal{S}$. Thus a POMDP is described using the tuple $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{P}^{a}_{ss'}, \mathcal{R}^{a}_{ss'}, \Omega, \mathcal{P}^{a}_{ss'o} \rangle$. In all cases in this thesis the observations are dependent only on the current state so the term $\mathcal{P}^{a}_{ss'o}$ becomes simply $\mathcal{P}_{s'o} = \Pr\{o_t = o \mid s_t = s'\}$.

2.2.4 Learning with knowledge of environment dynamics

In some situations detailed knowledge of the dynamics of an environment is easily accessible, for example, when the task to be learnt takes place in a computer game, or alternatively an accurate model of some real world dynamics may exist (such as those which are often provided for manufacturing plants). Learning algorithms exist which can take advantage of such information to compute solutions to tasks in these environments. For example, the collection of algorithms known as *dynamic programming* can be used to compute optimal policies given a perfect model of the environment described as a MDP [Sutton and Barto, 1998].

The problem of finding optimal policies for POMDPs, even when the full details of the POMDP are available, is PSPACE-complete [Papadimitriou and Tsitsiklis, 1987]. Despite this, approximate methods that use knowledge of POMDP dynamics (that is, $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{P}^{a}_{ss'}, \mathcal{R}^{a}_{ss'}, \Omega, \mathcal{P}^{a}_{ss'o} \rangle$) have been developed. For example, good results have been reported for the policy gradient method GAMP [Aberdeen and Baxter, 2002; Aberdeen, 2003].

Such techniques, however, lie outside the scope of this thesis as we consider learning without access to detailed knowledge of the world's dynamics. In all the examples considered in this thesis the agents have to learn based solely on their experience.

2.2.5 Learning internal models

There are multiple reasons why reinforcement learning algorithms might be designed to learn internal[2] models of their environment:

(i) Models allow for planning or forward prediction, for example, estimating the result of a series of actions prior to their execution.

(ii) Models allow value estimates or policies to be improved, for example, using the current internal model to estimate state values using dynamic programming methods [Sutton, 1990].

(iii) When interacting with POMDP environments, an internal model can provide a useful mechanism for estimating the true state of the world; for example, it can map the observations received to a MDP model to allow optimal action selection. [Chrisman, 1992b; McCallum, 1993; Whitehead and Lin, 1995].

In this thesis we are interested in algorithms that learn internal models for the reason set out in item (iii), that is, in order to render partially observable tasks Markovian.

Given that there are multiple reasons for learning internal models, we avoid the issue of the overused word *model-free* (see start of section 2.2) by insisting on more tightly defined terms. A review of literature provides the term *state-estimation* [Singh et al., 1994] which captures the function of the internal model in item (iii) above. The opposite type of algorithm, where the observations and rewards from the world are used to directly update the estimated state-values or policy is referred to as *direct reinforcement learning* [Sutton and Barto, 1998, p.230].

This thesis, for the most part, contrasts the performance of learning algorithms using *state-estimation* against *direct reinforcement learning* algorithms. However, the agents

[2]The word *internal* is used to emphasis that the models are learnt through the learning algorithms' experience of their worlds, and are not provided externally as was the case with the algorithms described in section 2.2.4. The models are also internal in the sense that they are generally not exposed to examination or modification except by the learning algorithm.

through which the direct reinforcement learning algorithms operate are enhanced such that the learning algorithms can modify the observations they receive from the world. It is useful, therefore, to consider the interaction between learning algorithms and the agents they are embodied in. Part of this involves clearly delineating what constitutes the functionality of the learning algorithm and what are external resources provided to the learning algorithm by the agent. In considering this, the following section introduces terms which allow us to classify types of agent, and from this we can then classify the agent-algorithm combinations that this thesis investigates.

2.3 Modifiable, Fixed (Fixed-SE) and Basic Approaches

Our thesis is motivated by our interest in agents learning to carry out tasks in the real world. To this end it is useful to consider the relationship between learning algorithms, embodied agents, and the world in which the agents are embedded and situated. Our concept of the relationship between these elements is set out in figure 2.3. An embodied agent acts as an interface between the world and the learning algorithm. The agent's abilities and limitations affect what the learning algorithm can observe about the world and what actions are available for the learning algorithm to select.

In this thesis we are interested in how agents' abilities can be extended in order to improve performance in partially observable domains. In examining this we believe that it is useful to divide the possible approaches into three categories. We therefore propose a new classification scheme which divides the extension of agents' abilities into three broad classes (i) *modifiable approach*, (ii) *fixed approach* and (iii) *basic approach*.

In order to formally define the distinction between these approaches, it is necessary to define exactly what we mean by the *core-learning-algorithm* as shown in figure 2.3. That is to say we need to draw an exact boundary between what abilities we designate as being part of the agent and what abilities we designate as being part of the learning algorithm.

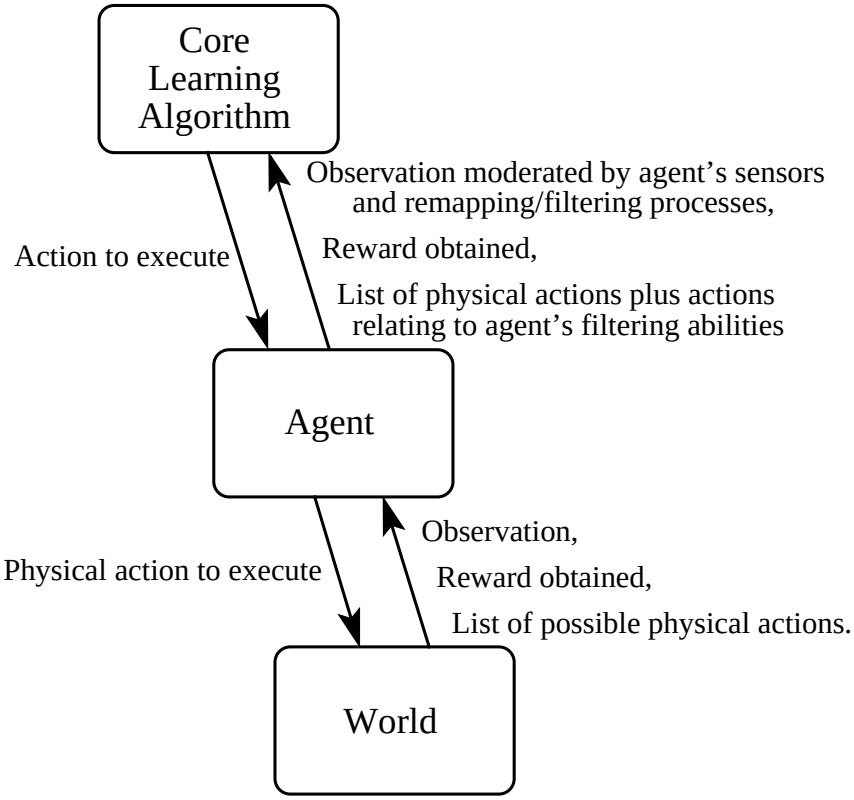


Figure 2.3: Relationship between learning algorithm, embodied agent and world

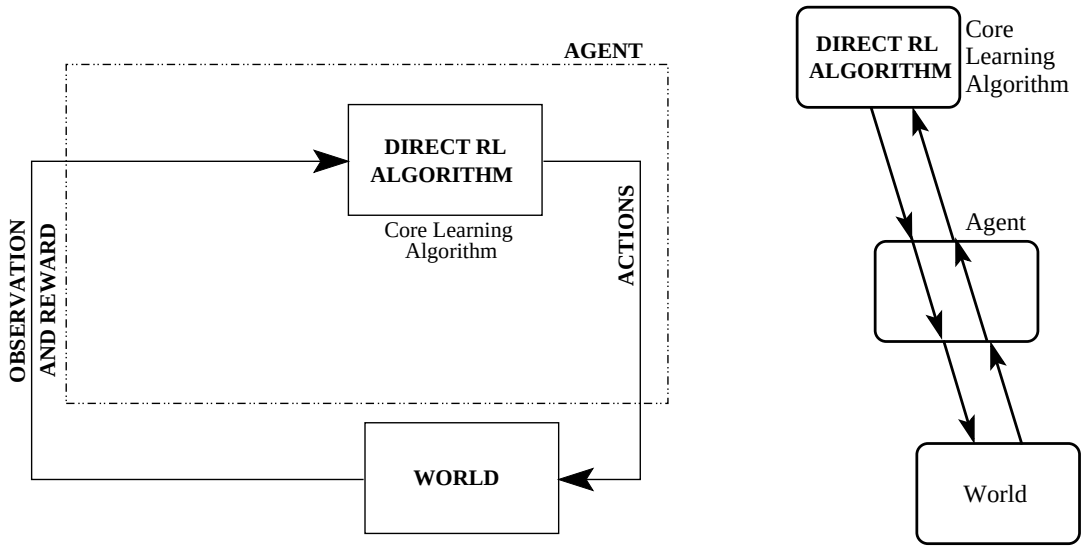


Figure 2.4: Simplest agent-learning algorithm design. Left hand side shows world, agent and algorithm using a layout common to reinforcement learning (RL) literature. Right hand side shows the same design in terms of the agent acting as buffer between the world and learning algorithm (as set out previously in figure 2.3). In this case the agent simply passes through observations which it obtains from the world

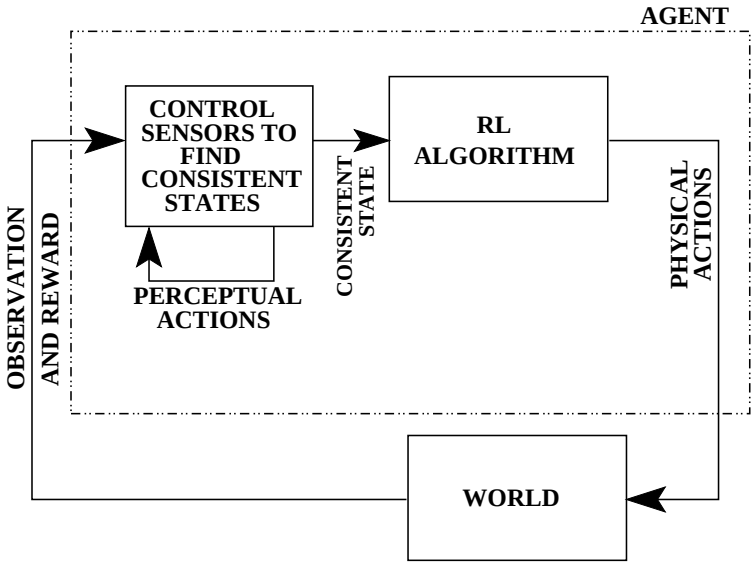
Our starting point is the simplest possible reinforcement learning design as shown in figure 2.4. This consists of an environment and a direct learning algorithm that does no more than learn profitable associations between the states which it observes and the actions it can choose. The learning algorithm is very simple, it has no memory of past events, it simply learns a mapping between the currently observed state and next action to select. A number of simple reinforcement learning algorithms exist which have been shown to successfully learn such associations, e.g. Q-learning, SARSA, SARSA(λ) and Watkins's Q(λ) (see appendices B–E). Such simple learning algorithms which learn direct associations and are indivisible are what we deem to be *core-learning-algorithms*.

More complex reinforcement learning algorithms that have been proposed generally retain at their heart a simple core which learns to associate the main actions afforded by the agent with the state representations which that core receives. As an example, consider the generic form of either Consistent Representation[3] (CR) or State Estimation (SE) algorithms as shown in figure 2.5. In comparison to the simplest reinforcement learning design, both of these design have some process that sits between the incoming observations and the reinforcement learning algorithm. There can be a chain of such processes, some of which may learn and others which may control specific actions afforded by the agent, but it is the simple core learning element at the end of that chain which we label as the *core-learning-algorithm*.

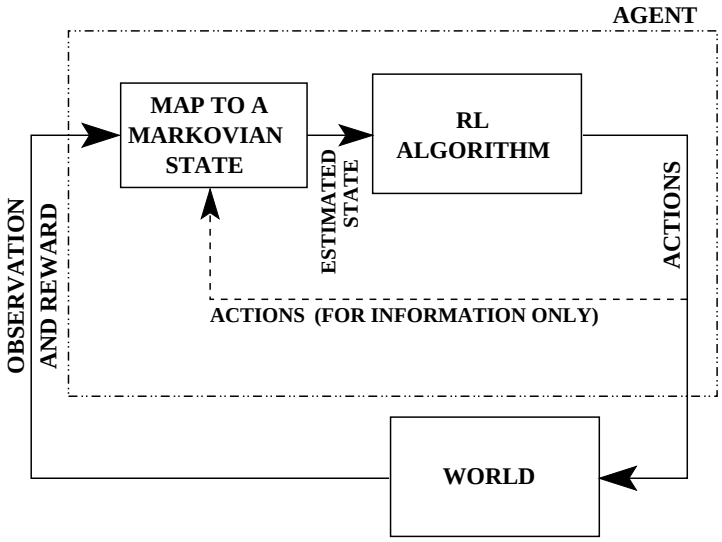
Let us consider three concrete examples to illustrate the point.

(i) Tabular SARSA(λ), appendix D, maintains a table of state-action values (Q values) which estimate the value of taking each of the actions over which it has control. SARSA(λ) has the form of the simplest agent-algorithm design as shown in figure 2.4. It cannot be subdivided, is a *direct* learning algorithm, in that it forms a direct association between the states with which it is presented and the actions selected. Thus we consider the SARSA(λ) algorithm as the *core-learning-algorithm*.

[3] A description of the design principles of CR algorithms can be found in section 3.1.1.



(a) Generic form of a Consistent Representation (CR) algorithm



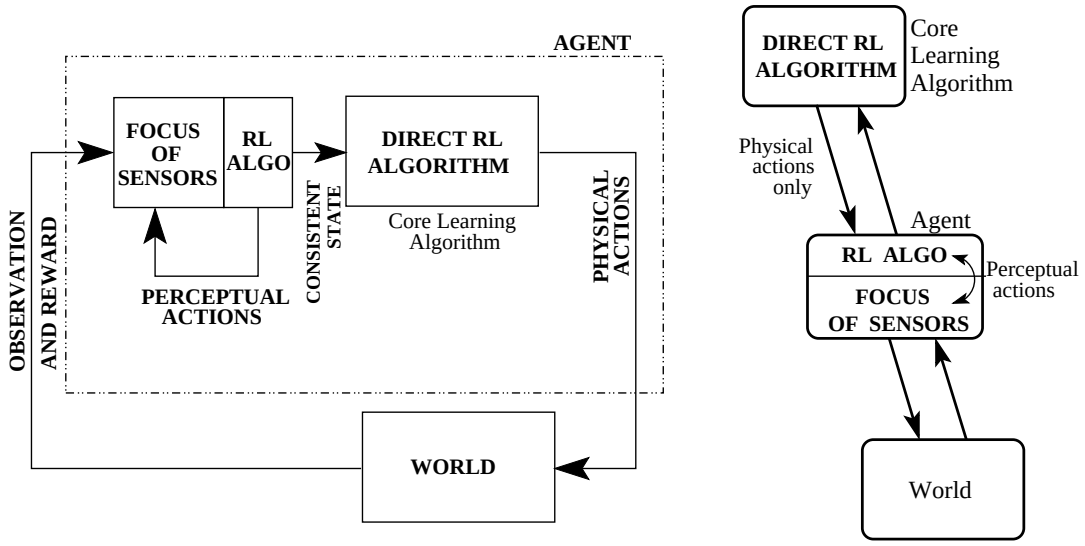
(b) Generic form of a State Estimation (SE) algorithm

Figure 2.5: Generic form of Consistent Representation (CR) and State Estimation (SE) algorithms

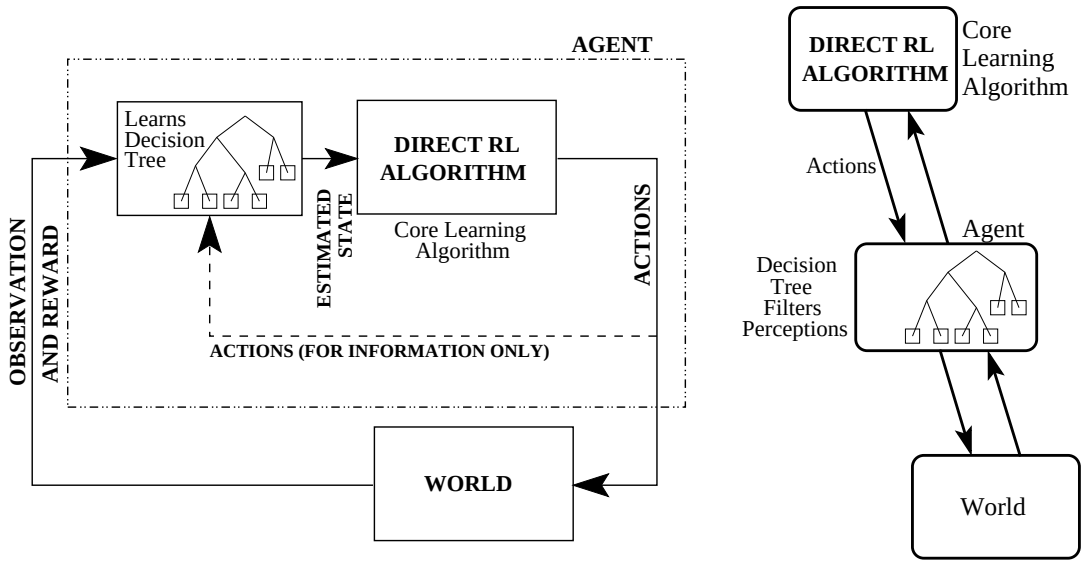
(ii) The Lion Algorithm as proposed by Whitehead and Ballard [1991] and Whitehead [1992] is a practical implementation of a CR algorithm. It consists of two reinforcement learning algorithms that work in conjunction as depicted in figure 2.6(a). The first reinforcement learning algorithm attempts to learn which are the best perceptual actions to select. During each iteration of the Lion algorithm the first reinforcement learning algorithm selects a series of perceptual actions which allows it to obtain different views of the current state. The second learning algorithm is then presented with this list of different views of the current state and selects the physical action that the agent then executes based upon the direct associations that it has learnt between the state views as provided by the first algorithm and the actions available to it. In this case we say that the second learning algorithm is the *core-learning-algorithm*. It is the core part which learns associations between the main physical actions that the agent can execute and the states presented to it. We regard the first learning algorithm as a filter and thus part of the agent's internal architecture using the breakdown we set out in figure 2.3.

(iii) A third example is the U-Tree algorithm [McCallum, 1995b, 1996] which is a SE type algorithm. As shown in figure 2.6(b), this algorithm builds a decision tree based on the actions taken by the agent and the resulting observations and rewards obtained from the world. Each leaf of this tree is the estimated state of the agent. Associated against each leaf are state-action values (Q values) which are updated by sweeps of dynamic programming. For U-Tree we regard the regular sweeps of dynamic programming as the *core-learning-algorithm*. Dynamic programming learns the direct association between the leaf states and the action to be selected in that state. The part of the U-Tree learning algorithm that builds the tree is regarded as a filtering process and as such we deem it to be part of the agent's internal architecture.

Having established what we mean by a *core-learning-algorithm*, we can now define the difference between modifiable, fixed and basic approaches in extending an agent's abilities.



(a) Lion algorithm. Perceptions seen by the core-learning-algorithm are selected by another reinforcement learning (RL) process which attempts to find consistent representations for each state



(b) U-Tree. Agent filters the observation passed to the core-learning-algorithm via the decision tree. This decision tree is learnt on-line

Figure 2.6: Outline of Lion and U-Tree algorithms

- **An extension which constitutes a *fixed approach* is one which provides the core-learning-algorithm with no additional actions, specifically no additional actions which affect its sensory input.** Typically, additional filtering or remapping stages are introduced which alter the state as perceived by the core-learning-algorithm, but the core learning algorithm cannot exert direct influence over these additional stages. The view it receives may have changed due to extending the agent's abilities but the control it has remains *fixed*, possibly even reduced with some actions being controlled by other stages.

- **An extension which is classified under our scheme as a *modifiable approach* adds to the existing actions that the core-learning-algorithm has control over. These additional actions being of the form which allows the core-learning-algorithm to directly *modify* the state which it perceives.** This modification of the perceived state is affected through the control of the sensors and/or the filtering/mapping processes applied to the sensory input. Typically the state space as originally perceived by the core-learning-algorithm remains intact with additional viewpoints now available at the request of the core-learning algorithm.

- **It is possible to modify the design of an agent-algorithm combination without extending the abilities of the agent. This occurs when the core-learning-algorithm is simply replaced by another core-learning-algorithm, for example Q-learning is replaced by SARSA(λ). We use the label *basic approach* to refer to this class of changes.**

Note that all three approaches are defined in terms of extending arbitrary agent-algorithm designs, not in relation to an absolute design. This is necessary as the simplest agent-algorithm design shown in figure 2.4 is an ideal that does not exist in reality, there is always some filtering of observations as shown in figure 2.7. This may occur implicitly due to the interface between the agent and the world or in the case of real world scenarios the observation will in all probability be explicitly filtered in order to constrain the volume of information obtained. Given that the simplest agent-algorithm design is an unrealisable ideal, it is thus necessary to define these approaches in relation to an arbitrary starting design in order to make their definition meaningful.

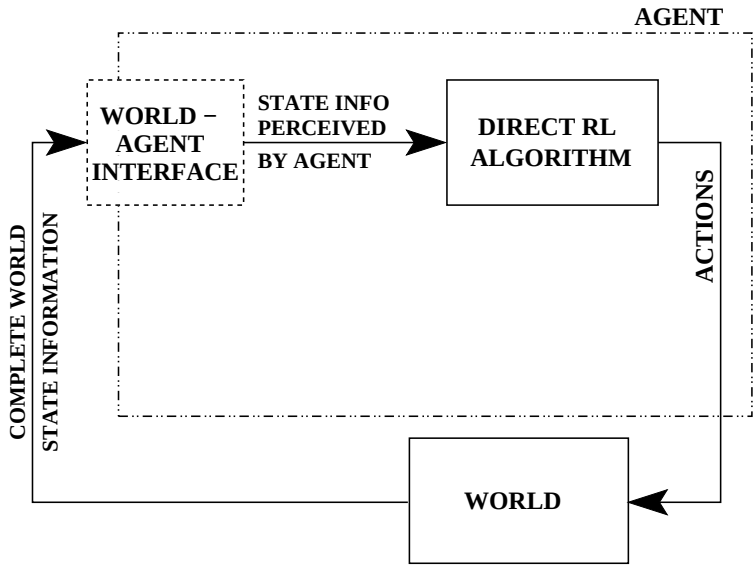


Figure 2.7: Realistic simple agent-learning algorithm design. It is not possible to have an agent-algorithm design that does not involve some implicit filtering of observations due to the nature of the interface between the agent and the world

The typical form of a *fixed-approach* extension is that illustrated by the Consistent Representation or State Estimation designs outlined in figure 2.5. The typical form of a *modifiable-approach* to extending an agent's abilities is set out in figure 2.8. Table 2.1 sets out how we would classify various reinforcement learning algorithms found in the literature using our new scheme. The classification of each algorithm is based on the assumption that it represents an extension of the simplest agent-algorithm design as depicted in figure 2.4.

We observe that the *fixed approaches* identified in table 2.1 consist exclusively of approaches which involve some form of state estimation.We include the consistent representation (CR) learning algorithms in the category of state estimation approaches. In the CR approach, the details of which are set out in section 3.1.1 and illustrated in figure 2.5(a), the process of seeking a consistent state representation among the possible input observations can be seen as a state estimation process due to the fact that the perceptual actions are not under the control of the core-learning-algorithm. From this viewpoint there is a strong correspondence between figures 2.5(a) and 2.5(b).

Lion Algorithm G−algorithm CS−QL Window−Q recurrent−Q recurrent−model k−order Markov methods Perceptual Distinctions approach (PDA) Utile Distinction Memory (UDM) Nearest Sequence Memory (NSM) Utile Suffix Memory (USM) U−Tree PSR / OOMs	fixed approaches
eligibility traces, *i.e.* SARSA(λ) MCESP−CD gradient decent approaches, *i.e.* VAPS	basic approaches
Active Perception Agents Internal Memory Agents	modifiable approaches

Table 2.1: Classification of various reinforcement learning algorithms or modifications of agent designs. The three *approaches* are those devised in this chapter for the classification of the extension of an agent's abilities. The classifications shown assume a common starting point of a simple agent-algorithm design as shown figure 2.4.

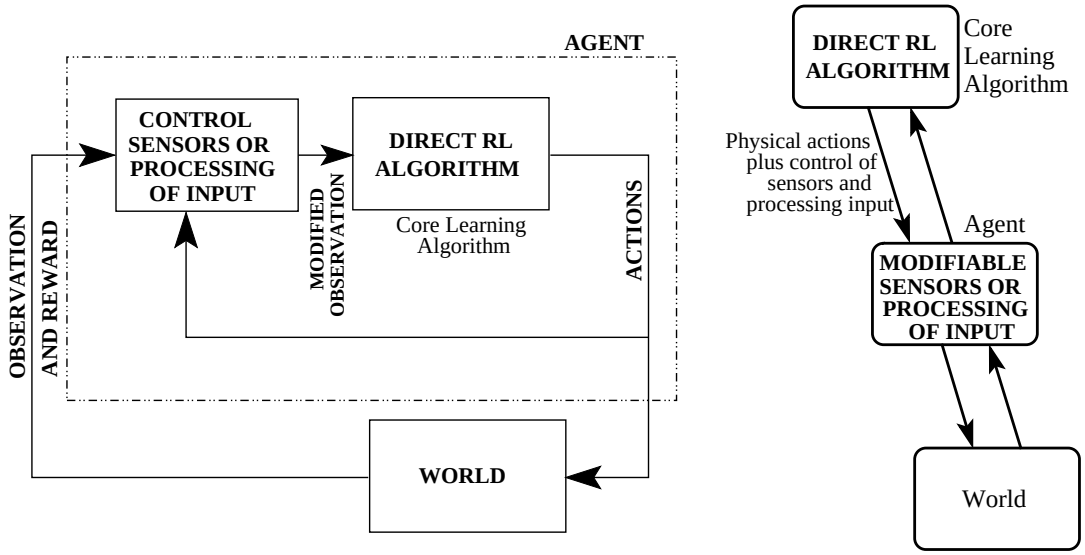


Figure 2.8: Modifiable approach

In a further refinement of our notation, we observe that all the *fixed approaches* that we consider in this thesis are also state estimation style algorithms. Thus we use the term ***fixed-SE approach*** to indicate *fixed approach* extensions to an agent's abilities which result in a state estimation (SE) design.

Active perception agents are an example of a *modifiable approach*, as the agent exposes additional perceptual actions that the core-learning-algorithm can select. The additional perceptual actions allow modification of the behaviour of the agent's sensors, or how the sensory information is processed, and thus affect the observations that the learning algorithm receives. A second design of agent-learning algorithms which fits the *modifiable approach* class, is that in which the agent has some form of memory which is modified through the actions selected by the core-learning-algorithm. Typical examples are agents which have a memory register that contains an arbitrary number of bits, such as the agent depicted in figure 2.9. The contents of the agent's register are modified by additional actions that are made available to the core-learning-algorithm and the core-learning-algorithm perceives the states of the memory register as part of the observation it obtains from the agent's sensors. Thus these additional actions allow it to modify its perception. Such agents have been proposed by Peshkin et al. [1999]

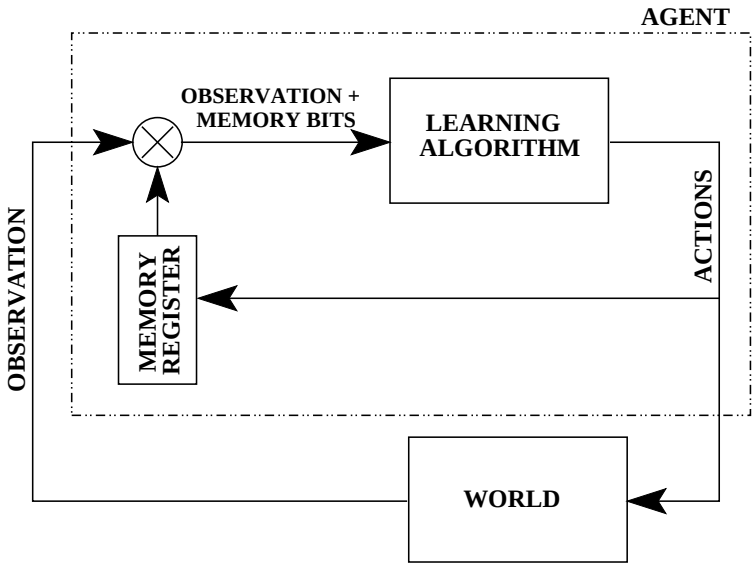


Figure 2.9: Modifiable agent with memory register

and Lanzi [2000][4].

Having identified these three approaches we can see that the modifiable approach involves applying direct reinforcement learning to (potentially) non-Markovian observations. Thus, any improvement that can be achieved in the performance of direct reinforcement learning algorithms on non-Markovian problems will not only benefit the simplest agent-algorithm design as shown in figure 2.4, but additionally learning with internal memory bits as proposed by Peshkin et al. [1999] and Lanzi [2000] and our active perception approach. Looking for ways to improve direct reinforcement learning on non Markovian tasks motivates our work in chapters 5, 6 and 7.

2.3.1 Why define these approaches?

We define these three approaches — fixed, modifiable and basic — because they reflect different underlying assumptions about the policies it is possible to learn for non-Markovian tasks.

[4]We note that Peshkin et al. [1999] refers to these memory bits as "external", whilst Lanzi [2000] refers to them as "internal". Both are actually correct, as Peshkin et al. [1999] considers them external to the *learning algorithm*, while Lanzi [2000] considers them an internal part of the *agent*.

The fixed-SE approach assumes that because static, deterministic policies can be arbitrarily worse than the optimal policy for a partially observable task [Singh et al., 1994] and that no convergence guarantees have yet been shown on such tasks, then it is necessary to transform the states observed into a Markov decision process. Whether a fixed-SE approach succeeds or fails is dependent on whether the transformation process can learn a suitable mapping for the task.

The modifiable (and basic) approach assumes that although the task is non-Markovian, a sequence of deterministic actions exist that will reach the goal even if such a policy is less than optimal. Although this can never be guaranteed to be the case (see section 4.4) we believe for any significantly complex world that this assumption can be made to hold through the extension of the agent's abilities. Further, in adding to the range of actions which can be selected and the perceptions that can be viewed, the number and quality of such solutions will increase making it easier to converge on a satisficing solution.

We expect that these underlying assumptions should lead to a marked difference in performance between fixed-SE and modifiable approaches. We also expect that agent extensions which fall within the same approach, for example all extensions that are modifiable, should exhibit similar levels of performance.

2.4 Chapter Summary

In this chapter we provide a brief introduction to the topics of active perception and reinforcement learning and set out our interpretation of the terms active perception, direct reinforcement learning and state estimation.

We also introduce and define three approaches to the extension of embodied agent abilities: that is the *fixed-SE*; *modifiable* and *basic approaches*. This new classification scheme is intended to capture a difference in approach to non-Markovian tasks which we believe will lead to significant differences in performance when applied to the same task. We use these three agent extension categories to structure the literature review which follows.

Chapter 3

Literature Review

This literature review is not intended as a comprehensive review of work in the area of applying reinforcement learning to partially observable tasks. Good reviews of the field are provided by, among others, Whitehead and Lin [1995] and Kaelbling et al. [1996]. Instead, our aim is to highlight the key ideas and contributions that shaped the current view of this field.

This review is structured using the classification the scheme proposed in section 2.3 which defined three different approaches to extending agents' abilities, that is *fixed-SE*, *modifiable* and *basic* approaches. We focus on these three categories as each represents a different approach to the general problem that we are interested in, that of an embodied embedded agent (for example a robot) learning to carry out tasks in a world where some of the information it requires to perform optimally is obscured (such as the real world).

3.1 Fixed-SE Approach

The *fixed-SE approach*, as shown diagrammatically in figure 3.1, consists of a combination of a state estimation (SE) method and a separate direct reinforcement learning

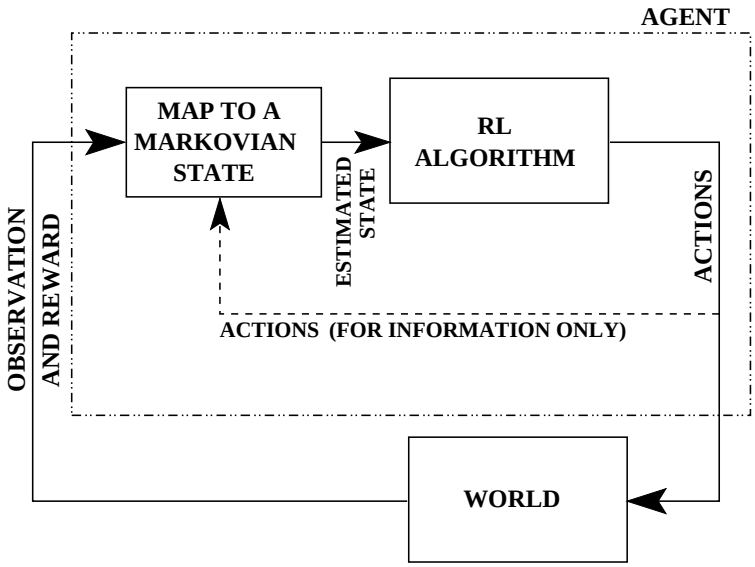


Figure 3.1: The fixed-SE approach

algorithm. The main feature of this approach is that there exists a separate process which attempts to find a mapping from the observations obtained, onto a Markovian representation of the task. The reinforcement learning algorithm learns its policy using the representation provided by the mapping process and crucially, as far as our definition in section 2.3 is concerned, has no direct control over the mapping process.

In this section we look at the contributions of authors who developed reinforcement learning algorithms that fit the fixed-SE approach. The work is mainly grouped under author names with the exception of the last section that reviews the recent Predictive State Representations (PSR) approach.

3.1.1 Whitehead and Ballard

Whitehead and Ballard [1990b, 1991]; Whitehead [1992] were interested in the more realistic approach that active perception represented when compared to the prevailing approaches in the field of Artificial Intelligence (AI) at that time. “It is common to assume that a decoupled (often implicit) sensory system provides the decision system with an internal representation that completely and accurately describes the state of the

external world. [. . .] The large amount of information encoded in these representations is difficult to deal with, but more importantly, most of it is irrelevant to the immediate task facing the agent." By assuming that only a subset of the possible information available from the world is needed for each decision, active perception promises to cut the computation resources required in sensing, and reduce the number of states of the world that need to be represented. Introducing active perception raises the question of control and Whitehead and Ballard [1991]; Whitehead [1992] examined the use of 1-step Q-learning (see appendix B) as the control mechanism, the aim of the control mechanism being to select parts of the environment that are relevant to the current decision. Based on the failure of Q-learning to learn to control an active perception system in order to solve a blocks world problem, and analysis of applying 1-step Q-learning to a simple one dimensional world [Whitehead and Ballard, 1991; Whitehead, 1992], Whitehead developed;

(i) the concept of *inconsistent* internal states,

(ii) the concept of *perceptual aliasing*, and

(iii) an adaptive control technique called Consistent Representation.

Inconsistent internal states arise when, due to improper control, an active perception system "generates a single internal state that represents two or more functionally different situations in the external world." Perceptual aliasing is introduced to refer to this undesirable overloading of internal states[1]. Based on the analysis of a simple one dimensional world Whitehead [1992] concludes that "1-step Q-learning cannot learn the optimal policy for this task." To overcome this problem the Consistent Representation (CR) approach is devised. "In the CR-method control is accomplished in two distinct phases: a state identification phase, followed by an overt control phase. During state identification, the system executes sensory-control actions [that is perceptual actions] in an attempt to generate an internal representation that accurately identifies the state of the external world with respect to the task." In other words achieves a consistent

[1]The definition is slightly different in Whitehead and Ballard [1991] where perceptual aliasing is broadened to include single external states that map to multiple internal states.

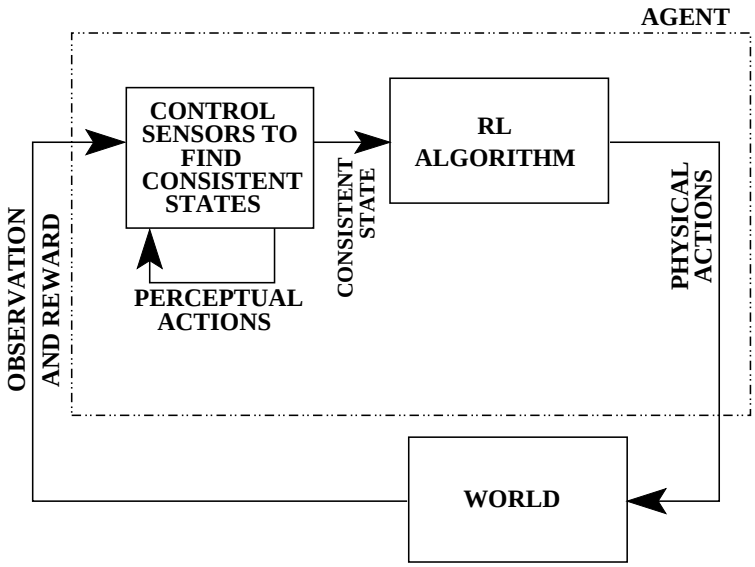


Figure 3.2: Consistent Representation (CR) approach

Markovian representation. "Next, during overt control, the internal representation is used to generate the overt [physical] actions needed to perform task." Figure 3.2 shows a simple outline of the CR approach.

As discussed in section 2.3 , we consider that although the CR approach is designed with active perception agents in mind, the combination effectively constitutes a fixed-SE approach. Our argument hinges on the fact that the process that attempts to find the CR mappings is not under the direct control of the reinforcement learning algorithm which learns the agent's policy, and neither are the perceptual actions of the agent. The agent's active perception abilities are manipulated solely by the mapping process which effectively carries out state-estimation.

Although the terminology introduced by Whitehead [Whitehead and Ballard, 1991; Whitehead, 1992] was possibly done without the realisation that the active perception robotic arm task could be described as a POMDP, the main symptoms were accurately diagnosed, and as such the concepts of *inconsistent internal states* and *perceptual aliasing* remain useful ideas when discussing the issues that arise in POMDP tasks. An additional useful phrase that arises from this work is the concept of *perceptually aliased states*. This phrase refers to a state whose observation, from the point of view

of the learning algorithm, is shared with other states in the environment, and which, for the purposes of learning a satisficing or optimal policy for a task, it would be advantageous to distinguish between. In this thesis we often shorten *perceptually aliased states* to just *aliased states*. With reference to the the simple one dimensional world setup to establish that "1-step Q-learning cannot learn the optimal policy for this task", we disagree with this statement and show in chapter 4 that under appropriate (and not particularly limiting) conditions it can converge to the optimal policy on this problem (though not necessarily the optimal value function).

Whitehead [1992] proposed the Lion algorithm, as detailed in appendix H, as a CR algorithm and demonstrated it learning with reasonable success on a simulated robotic arm, which used active perception to solve a blocks world problem.

The Lion algorithm's main limitations are that:

(i) The effects of actions must be deterministic.

(ii) Sensing must be noiseless.

(iii) The value of rewards must also be deterministic.

(iv) Only non-negative rewards are allowed.

(v) The number of states and actions are finite.

(vi) For each external state there must exist at least one configuration of the sensory system that generates an internal state that is Markovian.

The Lion algorithm assumes that the value of a consistent state will never decrease, hence the requirement that points (i), (ii) and (iii) hold. This same assumption also creates the requirement of point (v). A continuous state or action space will require, for example, a function approximation approach in order to represent Q values over the space. In employing such approaches, even if (i), (ii) and (iii) still hold, there can be no guarantee that state-action values for consistent states will rise monotonically.

Give the Lion algorithm's assumption, any decrease in state values due to function approximation would cause consistent states to be mistaken for aliased states.

Most of the above points are identified in Whitehead and Lin [1995]. We note that point (iv) can be addressed by a simple modification of the algorithm, requiring it to keep a record of the lowest state-action value seen and use this, instead of zero, as the value to which inconsistent states are suppressed. For more details see appendix H.

In addition to the limitations identified above, the Lion algorithm requires a division between perceptual actions which "change the configuration of the sensorimotor system (for example, shift the direction of gaze) but do not affect the state of the external world" [Whitehead and Ballard, 1990b] and overt actions, when in reality no such clear division exists. An overt action such as moving left will have a perceptual effect, possibly revealing previously obscured visual information. Likewise a perceptual action such as changing the direction of gaze can have an effect on the state of the world, for example, alerting observers to your intentions. Lastly the Lion algorithm imposes an artificial structure on the ordering and selection of actions. The agent always executes a state identification phase followed by an overt control phase. The state identification phase consists solely of selecting perceptual actions. The overt control phase consists solely of selecting an overt action. This ordering is artificial. In some situations it might be better to take an overt action in order to either gain a better view, or simply because it is possible (and maybe timely) to proceed in a ballistic manner ignoring perceptions. We consider that algorithms that dispense with the distinction between the types of actions and their ordering would be more natural. Such algorithms would be more akin to typical reinforcement learning algorithms like *Q*-learning, SARSA, etc.

3.1.2 Chapman and Kaelbling

Chapman and Kaelbling [1991] considered an agent learning in a video game environment where the observations provided by an active perceptual system consisted of a long binary vector, containing approximately 100 bits. The problems considered in Chapman and Kaelbling [1991] are not those of state aliasing, as with Whitehead

[1992] above, but problems caused by having to sample an enormous state space. That is to say, when such a long binary vector is used as the learning algorithm's state, that the number of possible values this vector could take resulted in an enormous state space. In addition each of these states would occur so rarely that the system would take an extremely long time to accumulate sufficient experience to be able to gauge their correct values.

Their solution was to develop the G-algorithm, an algorithm that incrementally grows a binary decision tree in order to select relevant binary features from the long binary observation vector. The G-algorithm initially lumps all the possible values that the binary vector could take together in a single node at the root of the tree. It assumes this single node represents an accurate generalisation for the expected reward value of all the possible observation vectors. Statistics are collected for each bit and Student's t-test is used to detect if any single bit splits the data into two distinct distributions of rewards. Once a bit which splits the rewards is found, the tree is divided into two leaf nodes using this bit's value, and the process repeated. The G-algorithm learns on-line, compiling the statistics as experience of the domain is gained. The nodes of the tree it builds are used as the internal states of the agent.

The G-algorithm can be seen as providing a form of state-estimation, though in the context that it is presented, it is more akin to a method of *state aggregation*, by which we mean a method for the clustering of observations[2]. We include the G-algorithm in this review because, as was observed by Whitehead and Lin [1995], its statistical approach provides useful ideas on the way that the active perception algorithms can be adapted to stochastic domains. An example of this is the U-Tree algorithm [McCallum, 1995b] which we present in section 3.1.8.

3.1.3 Ming Tan

[2] State aggregation is used to reduce the size of the state space as seen by the learning algorithm. This is advantageous as it reduces the time taken for policies, learnt by reinforcement learning algorithms, to converge. State aggregation can, however, cause similar problems to perceptual aliasing by obscuring details required to form an optimal policy.

Tan [1991b], like Whitehead [1992], considered the assumption "that a decoupled sensory system automatically provides the learning agent sufficient information to disambiguate states of the external world" as unwarranted. Tan and Schlimmer [1990] and Tan [1991b,c,a, 1993] pursued a more realistic approach to sensing. This approach focused on the cost of detecting the features of an object with which the robot wishes to interact. Tan [1991b] assumes that every sensing action that a robot can execute has an associated cost. This cost is generally expressed in terms of time. The approach proposed, *cost-sensitive classification learning*, aims to minimise the overall cost of evaluating features during both the construction (learning) of class descriptions and their subsequent use for the classification of objects [Tan and Schlimmer, 1990]. This is achieved by classifying objects using the most informative and least costly of the features which can be detected by the robot's sensors.

Objects are classified into groups which relate to the robot's task. Typical classifications could for example correspond to different gripping procedures, that is, grip, pinch and wrap. The main cost-sensitive learning approach proposed [Tan, 1991b] is partially decision tree based but differed from prevailing decision tree approaches in that it is incremental, that is, new objects, new classes of objects and new object features can be added incrementally to an agent's existing knowledge as it explores its world. This is achieved by storing a list of exemplars based on every different object seen. Then, for each object it needs to classify, it dynamically builds a new decision tree from these exemplars. This tree is used to decide what sensory features to check in order to classify the object. If the object is new, that is, the cost-sensitive learning approach finds a conjunction of features that does not match any of the existing exemplars, then the object is added to the list of exemplars.

The majority of the work in Tan [1991b] considers cost-sensitive learning in the context of a robot collecting items scattered on the floor of the lab, this work is also presented in Tan [1993]. In this context cost-sensitive learning forms one component of a control system with responsibility for identifying the grasping procedure in order to pick up an object. Tan [1991a,c] and Tan [1991b, chapter 5] considered combining cost-sensitive learning with Q-learning in order to relax the Markovian assumption of this reinforcement learning algorithm. In cost-sensitive Q-learning (CS-QL), each time the

state of the world changes, the cost-sensitive learning component builds descriptions of the current world state by selecting among the perceptual actions available to the agent. Each action is used to test and record features of the environment, and the actions selected are determined by a tree of tests and results which the cost-sensitive component has learnt, and which, theoretically, represent the least costly number of tests required to identify every significant state of the world. The state descriptions formed by these test are used as the internal states by the Q-learning component of the algorithm, against which it learns the agent's policy. Two assumptions of CS-QL are (i) that the agent's world and the rewards returned by the world are deterministic, and (ii) each action has a non-negative reward. Using these assumptions it is possible to detect state descriptions that are aliased. When aliased state descriptions are detected, the cost-sensitive learning component searches for the least costly perceptual action that it could apply in order to differentiate between the aliased state descriptions. This test is then added to the tree of tests applied by the cost-sensitive component. In this way, state descriptions are mapped onto consistent internal states. "A state description is defined as *consistent* if and only if its utility and best actions are the same as those of *all* the states that it represents" [Tan, 1991b].

Cost-sensitive learning can be trained off-line on examples with known classifications, or alternatively, can learn incrementally on-line from experience. The latter approach is adopted with CS-QL. Learning incrementally introduces three main issues, the first two of which require design decisions to be taken when implementing the CS-QL algorithm:

(i) The set of state descriptions against which *Q*-learning learns changes whenever a state-description is found to be inconsistent and a new test is added to the tree. This change will, due to the bucket brigade style updates of *Q*-learning, cause adjustments to the values learnt for consistent state descriptions. To avoid consistent state descriptions being mis-identified as inconsistent following a change in the state description space some heuristic has to be adopted; (a) reset all state description values to zero and recommence learning the policy from scratch, (b) add new state descriptions sparingly and ignore variations in state description

values immediately following the addition of a new state, or (c) after adding a new state description execute a series of random exploratory actions of nearby state descriptions (assuming you can identify and reach the locations/experiences that correspond to nearby state descriptions). The design trade-off of these three options, as demonstrated by Tan [1991a,b], is between the compactness of the internal representation and the time taken for Q-values to converge. The resetting strategy produces the most compact internal representation but takes the longest to converge as partially learnt policies, represented by the state description values, are reset several times.

(ii) Dealing with unknown feature values. As the features that make up a state-description are only measured as and when necessary, there is no guarantee that a particular feature will have been measured for a given state-description example. CS-QL has to therefore deal with examples containing unknowns. This raises three problems; (a) how to select between alternative perceptual actions based on cost when the candidate state descriptions that the algorithm is seeking to differentiate between have differing unknown features, (b) when partitioning examples using a given feature (as part of growing the discrimination tree), how to treat examples with an unknown value of that feature, (c) in attempting to separate aliased state-descriptions which are known to describe different states, how to proceed when tests on all the known common feature values have been exhausted without finding any differences. Tan [1991b] proposes two alternative heuristics for (a) and (b), a conservative heuristic that distributes the unknown examples evenly across the possible sub-divisions, and a liberal heuristic that ignores examples with unknown feature values. The former is preferable when the number of examples is small and each example is informative, the latter when there are many examples available and the loss of information is compensated by the speed of of classification. Tan [1991b] does not give exact details of how to deal with (c). One approach is to select the next cheapest feature and measure it on the example that is currently available. At some future time, when an example which is known to be aliased with the previous example occurs, then test this same feature, if this feature fails to separate the two state descriptions

then add yet another feature. Depending on the depth of similarity between the aliased states, this process could take a large number of trials to resolve the ambiguity. The time taken to distinguish between the state-descriptions will also be dependent on the frequency with which the agent visits each aliased states.

(iii) Early on, CS-QL will have an empty or sparsely populated library of example state-descriptions. This makes it impossible to accurately select the most cost effective[3] perceptual action as the calculation of information gain across a limited number of examples will, if possible at all, be inaccurate. The resulting decision trees used to select perceptual actions will be sub-optimal. The perceptual actions selected by these sub-optimal trees will affect the state-descriptions stored which will have a knock on effect on future decision trees. The situation improves as more examples are added at the cost of rebuilding the decision tree dynamically for each new example, however once all consistent states are identified cost-sensitive learning will not attempt to improve its classification efficiency and will continue to discriminate using what may be sub-optimal features. Tan [1991b] suggests a possible extension where feature exploration continues in the absence of state classification errors to allow cost-sensitive learning to shift to more efficient state-descriptions.

Tan [1991b,a] criticises the two reinforcement learning approaches reviewed in sections 3.1.1 and 3.1.2, that is, Chapman and Kaelbling [1991] and Whitehead and Ballard [1990a], for being "neither completely adaptive nor cost-sensitive." This criticism is based on the algorithms putting "certain limits on robot sensory capabilities (for example, restricting the number of sensors or the types of sensing features) and ignoring the costs of applying sensors (for example, the speed of a sensor and the time of processing sensory data)." Chapman and Kaelbling [1991]'s algorithm "tests all sensing features (which could be expensive) and then selects *individually relevant* sensing features (which could be insufficient) using statistical tests." The Lion algorithm presented in Whitehead [1992] ignores the cost of perceptual actions. In addition the sensing process of the lion algorithm limits the agent to determining its current state

[3]Cost effectiveness is computed as a ratio of information gain over cost.

from the information obtained from only *one* of the perceptual actions tried. This is in contrast to CS-QL where many perceptual actions can be chained together by the cost-sensitive learner in order to determine the current state. Further CS-QL does not look for a statistically significant *individual* feature to distinguish between states, but will continue to add features to state descriptions until all consistent states are identified.

CS-QL is limited [Tan, 1991b,a] in that it assumes that:

(i) The environment and rewards are deterministic,

(ii) Reward values are always positive,

(iii) The number of states and actions are finite,

(iv) Sensing is noiseless,

(v) Non-equivalent states *are* distinguishable – that is they differ on at least one detectable feature,

(vi) State descriptions required to classify consistent states are finite.

CS-QL shares limitations (i) – (iv) with the Lion algorithm [Whitehead, 1992] as the measure used to detect inconsistent state-descriptions (aliased states) is the same. Item (iii) raises two issues:

- As with the Lion algorithm, the approximation of a continuous state action space using some function will break the assumption that action-values for consistent states rise monotonically during learning (Baird [1995] shows that the behaviour of state values cannot be relied upon when look up state value tables are replaced with state value function-approximation). This could result in consistent states being mistaken for aliased states and the discrimination tree will therefore be grown needlessly,
- A further restriction is that CS-QL stores an example of every state-description encountered. This representation requires changing to deal with infinite state spaces.

Point (iv) is important as noise or errors in identifying state-descriptions could lead to consistent states being identified as inconsistent, and will definitely lead to over complex internal representation of the problem. Points (v) and (vi) arise as the algorithm will continue to add features to state descriptions until it can distinguish between them.

In addition to the points above, CS-QL, similar to the Lion algorithm, separates actions into two distinct groupings; perceptual actions executed by the cost-sensitive learning algorithm to discern the current state and physical actions (that change the state of the world) selected by *Q*-learning. As with the Lion algorithm, see section 3.1.1, we argue that this distinction is arbitrary and unjustified; physical actions can constitute perceptual actions and vice versa. Note this is not a general feature of the cost-sensitive learning approach, but a feature of the CS-QL algorithm in that the two components, cost-sensitive learning and Q-learning, perform two distinct tasks. The original design of cost-sensitive learning (which is as a supervised learning approach, not a reinforcement learning algorithm) makes no distinction between movement of the robot and other perceptual tests [Tan, 1991b].

3.1.4 Whitehead and Lin (review)

Whitehead and Lin [1995] is a review paper which summarises the above work. It presents a review of the Lion algorithm, CS-QL and G-algorithm, as well as considering three memory based approaches; window-Q, recurrent-Q and recurrent-model. All three memory based approaches use a neural network, as a function approximation approach to learn the *Q* values of the problem. In window-Q the input to the neural network is a sliding (temporal) window of the current sensation from the world coupled with the N most recent sensations and N most recent actions. In the recurrent-Q approach the neural network is a recurrent network [Elman, 1991] which can encode a compressed representation of relevant information which is arbitrarily deep in the past. The recurrent-model architecture has a one step prediction model that attempts to learn a model of the environment that predicts the next sensation. The predictive model's internal state is then used to provide an enhanced state representation against

which Q values are learnt. All three of these memory based processes perform state estimation through the use of memories of previous observations, the duration of those memories being fixed in the case of Window-Q, but potentially infinite in length for the two recurrent networks. As the internal state of a recurrent network is continuous the neural networks also effectively performs action-value function approximation. This is explicit in the case of the recurrent-model but implicit in the case of recurrent-Q.

Whitehead and Lin [1995] identifies two types of control problems that resist Markov modelling; (i) agents which have a high degree of control over the information collected by their sensors, (ii) agents that have a limited set of sensors which do not always provide adequate information about the current state of the environment (such systems are discussed in section 3.2). Both types represent *hidden state* or *aliased state* tasks "since they arise when information relevant to identifying the state of the environment is *hidden* (or missing) from the agent's immediate sensation."

By casting the three algorithms Lion algorithm, CS-QL and G-algorithm, into the common Consistent Representation (CR) approach, the paper suggests useful areas of cross fertilisation. These include using the G-algorithm's statistical approach in order to apply the Lion algorithm (or CS-QL) to non-deterministic domains. Further interesting suggestions are put forward for hybrid CR and memory based approaches, for example, allowing CR 'perceptual actions' that act on memory, thereby allowing the selection of a memory of a perception from N steps ago. Alternatively they suggest that by providing CR with memory it can avoid having to rediscover the current state at every time step.

Whitehead and Lin [1995] identified many of the problem outlined above with the CR approach to solving hidden state problems. One of the key limitations is the assumption that "by appropriate control of perception, the external states can be identified at each point in time from the immediate sensory inputs."

A key assumption for *any* memory based approach is also raised in this paper. "Note that the reason for restricting the possible initial states is to avoid perceptual aliasing at the onset of the trial when no contextual information is available." That is to say,

when testing such memory based approaches, account has to be taken of the disorientation that will occur when such agents are restarted in new locations. This will affect approaches which keep a history of past events, such as window-Q or recurrent-Q, and those that manipulate their own internal memory, as reviewed in section 3.3.2. It will, however, not affect active perception approaches. This is an important assumption often overlooked when comparing memory based techniques with alternative approaches.

3.1.5 Hochreiter, Bakker and Schmidhumber

A problem with the recurrent-Q network reviewed above can be the failure to retain memories over long time intervals. This is caused by the memories held in the recurrent circuit being swamped by new experiences. To improve the performance of recurrent neural networks on problems where memories of distance events are important, Hochreiter and Schmidhuber [1997] introduced a method called Long Short-Term Memory (LSTM). LSTM is implemented using recurrent neurons that have gated input and output. This gating helps protect memories of previous events. LSTM has been successfully applied to a number of POMDP tasks including robot navigation [Bakker et al., 2002, 2003].

3.1.6 Loch and Singh

Another form of state estimation which uses memory but in a more direct form than say Window-Q above, is the approach of storing previous observations and using them to augment the internal state representation. One such method, as proposed by Loch and Singh [1998], involves augmenting the immediate observation with the past $k-1$ observations to form an estimated-state, and then learn a policy that maps estimated-states to actions. Such methods are known as *k-order Markov methods* as the assumption is that k observations are sufficient for the environment to behave in a Markovian manner.

Implementing this approach necessitates a very small change to algorithms like SARSA, *Q*-learning, or SARSA(λ) in that the estimated state (that is, the vector of k observations $\langle o_1, o_2, \ldots, o_{k-1}, o_k \rangle$) is substituted in place of the state term s. Loch and Singh [1998] report good results with $k = 2$ and $k = 3$ on a simple grid world problem (Parr and Russell's maze).

The main criticisms of this approach are that k has to be fixed beforehand by the designer. The larger the value of k, the larger the estimated state space that the agent has to sample, so the slower learning will be, but if k is too small, significant connections between past and current events will be missed. Dividing up histories of observations using a fixed window is akin to dividing up a building plan using a fixed size grid. In some locations there is little detail and there will be lots of empty grid squares, in other places the grid squares will not be fine enough to pick out the required level of detail. Just as it is better to vary the grid size to fit the detail of a building, similarly it is better to vary the history length to match the detail required to disambiguate aliased states. Some algorithms that divide up history using variable length memories are presented by McCallum [1995b] and are reviewed in section 3.1.8 below.

3.1.7 Chrisman

Chrisman [1992a] introduces the concept of using POMDPs as a general representation for agents executing physical and perceptual actions in an uncertain environment. For a description of POMDPs see section 2.2.3. The primary focus of the paper is on planning with known POMDP models. To this end Chrisman [1992a] proposes that analysis of POMDPs can be simplified by exploiting the inherent properties of the agent's situated action and sensing, and the regularities of the real world. The two exploits that are proposed are directly transferable from planning with detailed knowledge of the POMDP dynamics to reinforcement learning approaches that do not have full access to the details of the environments dynamics. The two approaches are:

(i) Limit the plans developed such that the policies explored are restricted to those that are almost purely reactive; "the choice of action is (almost) a function of

current percepts". This approach corresponds to that which we seek to explore in this thesis. That of applying *direct* reinforcement learning to POMDP tasks in conjunction with a simple agent-algorithm design, such as figure 2.4, or a simple design to which the *modifiable approach* has been applied. Though in both these cases, the policies are *strictly* reactive policies.

(ii) Given that results of actions are in the main predictable and largely deterministic, and that the observability of world state is high, then an agent can maintain with some degree of certainty a belief about the current world state. This belief is represented probabilistically using the POMDP structure of the task with the agent maintaining a set of probabilities that represent the probability of occupying each *state*, that is, s_1, s_2, etc. in figure 2.2, of the world. These probabilities are then used to weight the selection of actions. For reinforcement learning, in the absence of access to the detailed dynamics of the environment, the agent can attempt to learn a POMDP model of the world based on the observations it sees, and then maintain belief states over this learnt model. This is the approach taken in Chrisman [1992b], see below, and McCallum [1995b]'s Utile Distinction Memory (UDM) algorithm described in section 3.1.8.

Chrisman [1992b] proposed the Perceptual Distinctions approach (PDA) for agents that have an incomplete perception of the environment. In earlier work Chrisman and Simmons [1991] and Chrisman [1992a] considered agents which can vary their perceptions via perceptual actions. However, when they outline the Perceptual Distinctions approach Chrisman [1992b] only considers an agent with fixed incomplete perception of the world. Perceptual actions could be used with the Perceptual Distinction approach but would probably over-complicate the internal model of the world that the algorithm builds as it would attempt to capture the many-to-many mapping that could occur between locations in the world and observations selected. The agent builds an internal POMDP model over which it maintains a belief about the state of the world in line with point (ii) above. It determines whether the model it has built so far is adequate by attempting to predict the next perception the agent will observe, based upon its current observation and the chosen action. The agent adds states to this model until

it can successfully predict the next observation.

If the world can be represented using a POMDP, then storing a complete memory trace of actions and observations would allow learning algorithms to completely distinguish between every state that the agent reaches. The disadvantages of such an approach are that the agent sees every state as unique and cannot generalise actions between states. Therefore, if we believe the dynamics of the world fit those of a POMDP, then attempting to learn a POMDP model of it offers advantages of (i) a compact representation, (ii) removes the need to maintain a potentially infinite long memory trace – instead we maintain a belief state which can be represented as a vector of probabilities over the number of states in the model, (iii) a more intuitive way of being able to generalise than methods that attempt to generalise between memory traces.

The disadvantage of using a POMDP is that the number of internal states used needs to match the complexity of the task such that it can sufficiently distinguish between aliased states. Insufficient internal states in the POMDP will lead to the internal states containing an aliased representation of the agent's current state. Too many internal states will result in slower learning. In answer to this issue the Perceptual Distinctions approach adapts the number of states used in the POMDP such that the number is sufficient for the POMDP to predict the next perceptual state that the agent will observe.

The PDA clearly fits the classification used in this thesis of a *fixed-SE* approach. The POMDP model that the agent learns, carries out state-estimation (SE) by providing a mapping from the sequence of observations received to a belief state that, if the POMDP model is accurate, is Markovian. The belief state however poses a problem in itself in that it represents a continuous state space. The occasions when the system believes 100% that it is in one state of the POMDP will be very rare, belief will generally be spread in a continuous manner across a number of states. There is therefore a need to discretize the state represented by the belief state or use function approximation to represent Q-values. Unfortunately either of these approaches will introduce approximation errors in the learnt state values [Thrun and Schwartz, 1993] (interestingly, these errors are akin to those induced by the aliasing of states; for futher discussion of this point see section 9.6). PDA uses a simple function approximation approach, with the

estimated probability of occupying each individual state being counted as weighted vote towards that state's preferred action. The executed action is that which wins this poll.

Other criticisms of PDA are:

- That it attempts to match every perception that the agent perceives, whether these are relevant to the task at hand or not. Thus the POMDP model it develops may be more complex than is actually required to learn the task. This is an issue considered by McCallum [1995b], see section 3.1.8.
- Learning of the POMDP is generally not optimal in that it generally creates more states than are strictly necessary (this is, in part, equivalent to over-fitting in machine learning [Russell and Norvig, 2003] as it tries to build a model that fits the noise in the data).
- It maintains a trace of observation-action pairs in order to build the POMDP model which will significantly increase the memory requirements of any machine utilising this approach.
- The statistical approaches used to build the POMDP model are computationally intensive and thus time consuming.

3.1.8 McCallum

McCallum [1995b] argued against the arbitrary partitioning of perceptual actions and physical actions as used by Whitehead [1992]. McCallum [1995b] proposes that actions form a continuum "from actions that are primarily manipulative of world states, such as lifting a box, to actions that primarily change perceptual processing, such as shifting concentration from the words written on a piece of paper to the color of the paper." He further argues that it is impossible to draw firm boundaries between the types of actions, "even at the extremes of either side of the continuum, we can see

elements of the other side," for example, lifting a box is not just a physical action it is also perceptual in that it uncovers what was underneath.

McCallum [1995b] also identifies that the two problems of *too much* and *too little* sensory data are intimately related. Too much sensory data poses an issue for generalisation, too little sensory data results in states that, from the agent's viewpoint, are aliased. The proposal of his thesis is that the agents must be able to both select sensory data, that is, apply selective attention, in order to generalise, and use memories of past actions and observations to resolve aliased states. It is then argued that resolving aliased states using short-term memory, can be seen as a problem of applying selective attention to memory – "the agent cannot possibly afford to remember everything; it must *select* what to remember and what to forget."

McCallum [1995b] also introduces the useful idea of focusing on the *property* of a model that we wish to be Markovian, for example, Markov with respect to property X.

> Traditionally, a model is said to have the *Markov property* when it is "Markov with respect to the model state," that is, a model has the Markov property if future state is conditionally independent of past states (and actions), given the current state (and action). We write this as follows: $\Pr\{s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots s_0\} = \Pr\{s_{t+1} \mid s_t, a_t\}$ where s_t is the model state at time t, and a_t is the action chosen at time t, and subscripts $t-1$ and $t+1$ indicate steps in the past and future.
>
> However, we can also talk about the model being "Markov with respect to reward," or "Markov with respect to perception" or "Markov with respect to a certain feature of perception."
>
> A model is "Markov with respect to reward" if and only if future reward is conditionally independent of past states and actions, given the current state and action. We write: $\Pr\{r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots s_0\} = \Pr\{r_{t+1} \mid s_t, a_t\}$ where $r_t + 1$ is the expected future discounted reward at time $t+1$.

The objective in reinforcement learning is to maximise some reward function, normally the future discounted reward. In order to successfully learn a task a reinforcement learning agent therefore does not need to distinguish between every state of the world; it is sufficient for it to distinguish between states that are Markovian with respect to

reward. McCallum [1995b] refers to such states as *utile* aliased states, the word utile coming from "utility", which is a term often used in reinforcement learning literature to mean "expected future reward". An agent which has established a set of internal estimated states that are Markovian with respect to reward, that is, a set of utile internal states, has sufficiently resolved its understanding of the world such that it can learn the reinforcement learning task.

Based on the above principles McCallum [1995b] presents four algorithms for learning with hidden states; Utile Distinction Memory (UDM), Nearest Sequence Memory (NSM), Utile Suffix Memory (USM) and U-Tree. Details of these algorithms can also be found in McCallum [1993, 1994, 1995a, 1996] respectively.

3.1.8.1 Utile distinction memory

Utile Distinction Memory (UDM) is a algorithm that uses a partially observable Markov decision process (POMDP) to represent its internal state in order to learn tasks containing aliased states – for a description of POMDPs see section 2.2.3. In comparison to previous approaches, for example, Chrisman [1992b]'s Perceptual Distinctions approach, the key difference of UDM is that it uses the *utile distinction test* to determine when extra internal states should be added to its POMDP model of the task. McCallum [1995b] defines the utile distinction test:

> The utile distinction test distinguishes states that have different policy actions or different utilities, and merges states that have the same policy action and same utility.

Using statistics collected on future discounted rewards, or *return*, the UDM algorithm applies the utile distinction test to determine if it should add states to its POMDP model of the world. If a state in the POMDP is Markovian with respect to reward then the return associated with each incoming transition should be similar. If, however, there are statistical differences in the returns, then the model's state is non-Markovian with respect to reward, and the algorithm attempts to improve its predictive power by splitting that state. When a model's state is split, the incoming transitions are partitioned

into sets, such that those in a set share statistically similar returns, and between sets the returns differ. Then sufficient new states are created such that there is one for each of the sets of incoming transitions. The assumption in this splitting process is similar to that for *k*-order Markov methods, that is, knowledge of the preceding states will render the current state Markovian (with respect to reward). In this case knowledge of preceding states is captured by changing the model such that different sequences of preceding states now arrive at different (split) states. It should be noted that the preceding states are not grounded in the environment but are states of the learnt POMDP model.

UDM has to be initiated with a POMDP model which contains a state for each possible perception that the agent could experience. UDM has the ability to split states based on the history of preceding states. It lacks, however, a mechanism for splitting states based on the current observation, thus the internal POMDP model has to be initialised with an internal state for each possible percept. This contrasts with the Perceptual Distinction approach (Chrisman [1992b], as reviewed in section 3.1.7) which starts with a single internal state that represents its initial view of the world (that all states are the same) and then splits this single state to build a model of the world.

That UDM needs to be initialised with all possible agent perceptions seems to be a weakness of this algorithm. It either requires that the designer build into the agent his prior knowledge of all the perceptions that will be encountered whilst attempting to learn the task, or possibly worse, if the designer has no knowledge of the perceptions that will be encountered, UDM would have to be initialised with a model that contains as many states as there are combinations of features allowed by the agent's sensors. The number of feature combinations, which is dependent on the design of the agent's perceptual system, might be very much larger than the actual perceptions that will be encountered in the world. Thus the designer could burden UDM with an overly complex internal state representation.

As identified in a subsequent paper, McCallum [1995b], UDM learns extremely slowly. This is due to the complexity of updating the POMDP model. Separate to the procedure of testing and splitting states, the transition probabilities of the UDM's POMDP model are updated using the Baum-Welch procedure [Rabiner, 1989]. The Baum-Welch pro-

cedure alternates between performing trials to collect statistics on the current model, and updating the transition probabilities of the model based on those statistics. In addition, a large number of steps is required in each trial to make the experiences statistically significant.

UDM also has difficulties in discovering the utility of memories longer than one time step since the statistical test only examines the immediate previous state. This problem was identified by both McCallum [1995b] and Hartley [2004].

Hartley [2004] undertook a thorough analysis of UDM and attempted to improve its theoretical soundness. Although he showed that it is possible to simultaneously grow a POMDP model of the environment and learn the control policy to provide optimal behaviours, the complexity of the process tends to result in a large degree of instability. Convergence can not be guaranteed, and the performance of the system, when it moves away from a solution, does not degrade gracefully but tends to collapse completely. Based on these results he suggests that the use of any global optimisation technique to simultaneously learn an environment model and control policy for a partially observable environments is not sensible. The techniques are too powerful and the results will tend to be fragile due to the circular coupling between model, policy and observations.

Further evidence of the problems inherent with approaches like UDM and PDA is provided by Shatkay and Kaelbling [1997]. They demonstrate that processes which attempt to learn of POMDP models from observation data are prone to becoming trapped in local minima.

3.1.8.2 Nearest sequence memory

Nearest Sequence Memory (NSM) addresses two of the drawbacks of UDM, namely the speed of learning and the ability to discover multi-step memories. NSM is an instance-based technique. Instance-based techniques are distinguished from other approaches by maintaining a set of raw instances across which learning occurs. In other

fields, such as theorem proving [Baumgarter and Stenz, 2004], the set of instances are generally known in advance. In the case of reinforcement learning, the instances are collected as the agent explores the world. An instance based reinforcement learning agent stores the raw sequence of actions, observations and rewards. The trade off in using memory to store the agent's raw experience, is that when the agent's internal model of the world changes, for example, when it adds extra internal states, then the agent's original raw experience is still available to re-populate the new model. In contrast a non-instance-based method tends to incorporates experience by averaging it into its current flawed internal model, with the result that some experience will be attributed to the wrong model state, and thus the experience is wasted [McCallum, 1995b].

NSM closely resembles the *nearest neighbours* algorithm [Russell and Norvig, 2003] which classifies new examples based on existing labelled data. In nearest neighbours, a new data point is classified by seeking out its nearest k neighbours (where k is fixed) and checking the classification of these neighbouring points. In this way known, pre-labelled data points are used to sub-divide the space they occupy and can be used to label new data. In NSM the data points are the stored instances of actions, observations and rewards, that is, $\langle a, o, e \rangle$. The distance metric used to determine the proximity of neighbouring instances is based upon matching the current instance's actions, observations and rewards, and also matching as many as possible of those instances that lead up to the current instance. The measure is defined such that the closest neighbours are those with the longest sequences of preceding actions, observations and rewards that match. Stored instances are labelled with Q values which identify the value of taking each possible action from that instance. To find the next action to select given some new instance, the nearest k neighbours to the new instance are found. The neighbours' Q values are then used to determine the action the agent should take, following which an adapted version of Q-learning is used to update the 'labels' of the k neighbours.

Experimental evidence in McCallum [1995b] demonstrates that NSM indeed does learn substantially quicker than UDM and also Chrisman [1992b]'s Perceptual Distinction approach. It is also capable of finding solutions to tasks on which UDM fails.

NSM has no concept of hidden states or utile distinctions. It finds the most closely

matching neighbours it can, even if such tight matching is not necessary. As McCallum [1995b] observes, the specificity with which NSM identifies a point in the instance space depends not on requirements related to the task, but on the sample density available for that region of space. This tight matching will have a detrimental effect on NSM's ability to generalise, for example consider an agent reaching the end of a given corridor, if the start position along the corridor can vary, for example, exiting from two different offices, and the difference is internally captured by the agent as a different number of 'move forward' instances, then the agent may have to learn twice what action to execute at the end of the corridor, as the two sequences may not match closely enough to allow generalisation between them. Related to this tight matching of sequences is the absence of any test to determine if the state distinctions are utile. In not testing for utile distinctions, NSM also fails to check for aliased states. It basically hopes that in achieving the longest matches possible it has avoided the aliasing of any states, something that will not necessarily be true. This tight matching also means that NSM handles noise badly as it is unable to distinguish between changes in state-action-reward sequences that are due to noise and those that are significant for predicting future reward.

A further criticism of the NSM algorithm is that although it stores a long sequence of action, observation and rewards it does not take full advantage of the information it has collected. For example, rather than doing a 1-step update of Q values it should be possible to do sweeps of dynamic programming updates (see section 2.2.4) which would improve the calculation of the returns associated with each instance.

3.1.8.3 Utile suffix memory

Utile Suffix Memory (USM) combines instance-based learning, as used in NSM, with a utile distinction test. This produces an algorithm that can capture utile distinctions, as in UDM, but learns quickly. USM uses suffix trees (also known as decision trees [Russell and Norvig, 2003]) to capture models of the world and thus estimate the agent's current state. The style of tree learnt is shown in figure 3.3; each layer of the suffix tree

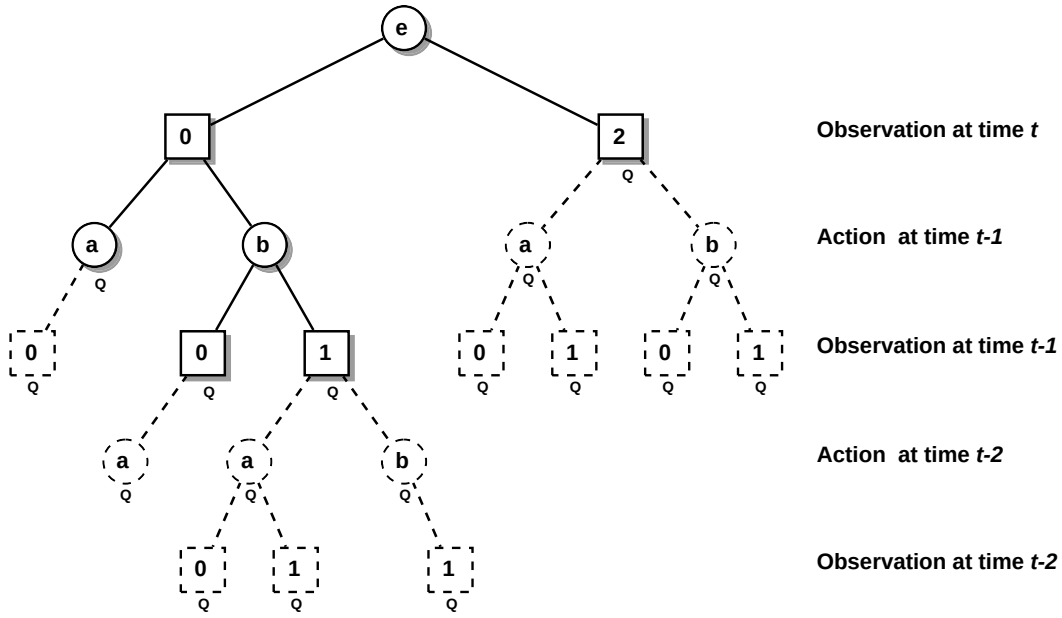


Figure 3.3: A USM-style suffix tree. Perceptions are indicated by integers, actions by letters. The fringe nodes are drawn in dashed lines. Nodes labelled with Q are nodes which hold *Q*-values. Diagram reproduced from McCallum [1995a]

alternately corresponding to observations and actions. The deeper a node, the further back in time the action or observation occurred, giving USM variable length memories.

USM uses the utile distinction test to determine explicitly how much memory is required for predicting future rewards. It does this by maintaining a fringe beyond the existing leaves of the suffix tree. This fringe is grown using additional past distinctions that have not yet been taken into account. The Kolmogorov-Smirnov test is then applied across the set of fringe nodes to discover if returns for one fringe leaf node significantly differ from its parent's, which is a leaf node of the tree proper. If a significant difference is found, it implies that this fringe node will help predict future reward. Fringe nodes are thus promoted to form part of the proper tree. USM is initiated with a suffix tree that contains no history information, just the agent's current perception; that is, a root node with one branch for each possible observation. To this tree USM adds nodes, as described above, until each of the leaves of the suffix tree are Markovian

with respect to reward. USM can be viewed as building an k-order Markov model (as discussed in section 3.1.6) of the state space, where k is variable across the space.

Given a set of leaves that are Markovian with respect to reward, a reinforcement learning algorithm can then learn to evaluate an agent's actions using the leaves of the suffix tree as internal states. Q values are associated with each leaf and are updated using one step of dynamic programming for each step the agent takes in the world.

The statistical test used makes USM more robust than NSM with respect to noise, as it will tend to differentiate between structure and noise in the data.

In three experimental tests [McCallum, 1995b] USM was reported to: (i) successfully separating noise and structure in the environment and only building as much memory as needed to perform the task; (ii) learning quicker than NSM and much quicker than the Perceptual Distinction approach [Chrisman and Simmons, 1991]. However it struggles to learn tasks with large sensory spaces, the suffix-tree making as many branches at each observation layer of the tree as there are possible observations. For such tasks the provision of a knowledgeable teacher to guiding USM's exploration improved it ability to learn [McCallum, 1997].

One of the objectives stated for USM was to combine the best features of UDM and NSM. In one area, that of discovering multi-step memories, there is a significant compromise. NSM can discover significant multi-step memories of arbitrary length. It is difficult for USM to discover multi-step memories that are any deeper than the set fringe depth. This parameter has to be set appropriately by the designer, and cannot be adapted to the task by the learning algorithm if the wrong depth is selected.

It would appear to be more principled when deciding to extend the tree to carry out the statistical comparison between each related node of the fringe, rather than comparing them to their parent. This should result in better identification of significant splits. However it would also result in significantly more tests, that is, n^2 tests compared to just n. Finally the ordering of branches is fixed, with each layer of the tree representing alternately actions and observations, and each pair of layers a single step back in time.

It would, in general, be possible to optimise the tree by allowing the decision points to be reordered.

3.1.8.4 U-Tree

The U-Tree algorithm is very similar to the Utile Suffix Memory algorithm above. Both carry out utile distinction tests to build trees whose leaves are Markovian with respect to reward. To do this both maintain fringes below the current official leaves of the tree. Both learn Q values using the leaves of the tree as the internal state of the agent, and both are instance-based algorithms that maintain a chain of previous raw experience which can be used to re-populate the algorithm's tree as it grows.

As indicated at the start of section 3.1.8, McCallum [1995b] argues that the two issues of too little and too much sensory data are closely interrelated. To this end the "key idea behind U-Tree is that the same tree structure used to represent state distinctions based on memory, can also be used to represent state distinctions based on features of perception." U-Tree, like G-algorithm [Chapman and Kaelbling, 1991], treats the observations obtained by the agent at any one time step as a vector of features. This contrasts with many of the previous algorithms that treat each observation as an indivisible whole. By treating each observation as a vector of features it is possible to test and branch on selected individual features. This can (i) reduce the branching factor when compared to branching on complete observation vectors, and (ii) improve the ability of the leaves (internal states) to generalise.

See figure 3.4 for an example of the kind of tree built by the U-Tree algorithm. Each decision point is indexed by two parameters: a perceptual dimension indicating which feature of the observation to test, and a history index indicating whether it is a feature from the current observation that is to be tested (history index equal to zero) or some previous observational feature (non-zero history index). By forming branches based on current/previous features the U-Tree is able to form memory chains such that it can distinguish between states. Note in this context the features include actions executed by the agent.

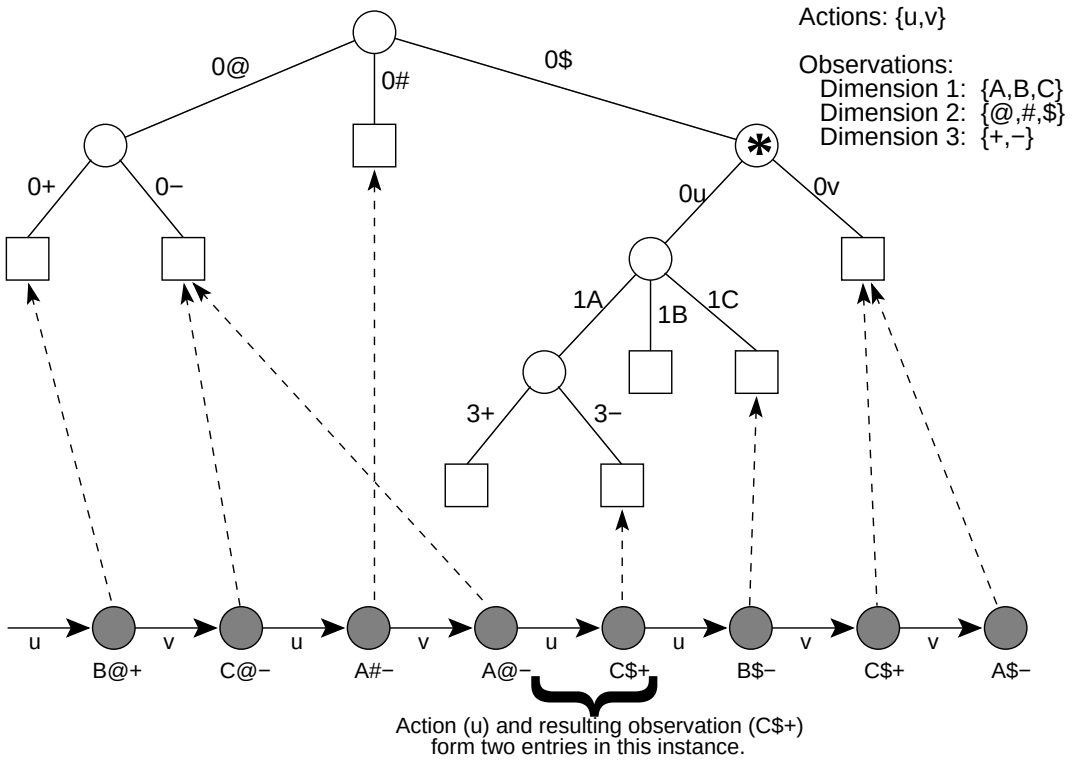


Figure 3.4: Example of U-Tree instance-chain and tree. The instance chain is shown at the bottom of the figure, with each instance represented by a grey circle and the chain of events that preceded it. Every circle in the instance chain is labelled by the observation received at that time step, and every arrow is labelled with the action taken to make that transition. Each branch of the tree, top part of the figure, is labelled by a history index and perceptual dimension. History indexes enumerated backwards through the instance chain with 0 being the current instance, 1 being one instance ago, etc.. The perceptual dimension is either a sub-division of the sensory space or the action executed. Tree leaves, drawn as squares, are the agent's internal states. The dashed-arrows indicate the agent's "state" given that it just has experienced the instance where the dashed arrow originates. The dash-arrows do not indicate the leaf where the instance is stored, this is found by considering the previous instance. The diagram is modified from the one presented in McCallum [1995b] and McCallum [1996] to be consistent with our understanding of the algorithm, see appendix G for more details

The use of features, over observations, makes it possible for the leaves of the U-Tree (which are the estimated internal states) to generalise across large parts of the observation space. If a U-Tree model achieved this, the resulting compact internal state representation would make it easier for the reinforcement learning algorithm to learn the correct policy. However, the attempt to simplify the task of the reinforcement learning algorithm results in an increase in complexity for the model building part of U-Tree, as the possible ways in which the fringe can be extended grows. If each observation is split into n features then the number of possible ways of growing just one extra layer of the fringe is $n+1$ (including the previous action). In addition, unlike USM, there is no fixed ordering over the feature that should be tested, previous actions and observations can be tested before more current observations. Although this allows the possibility of more compact trees being developed, this also increases the number of possible fringe expansions, resulting in an enormous possible number of fringe expansions. The only limitations are the manual settings that constrain the depth that the fringe is extended to, and limits on how far back in time the history index can reach. In practice, to reduce the computational load, artificial biases are generally imposed to limit the possible fringe expansions which are generated (such as placing an artificial constraint on the order in which feature tests are combined). In addition the fringe is not tested after every time step, but after some fixed number of steps, for example, every 1000 time steps.

McCallum [1995b] demonstrates U-Tree learning using a simulated driving task where an agent attempts to steer a car along a highway, safely overtaking slower vehicles and moving out of the way of faster vehicles. The agent has control of its perception, choosing whether to look ahead or behind and also which lane of traffic it wishes to regard. Its perception is limited to the direction and lane on which its gaze is currently focused. In addition to the aliased states created by this active perception system there is also redundant information that the agent perceives, for example, the colour of the cars, road surface and hard shoulder, all of which can vary randomly. The design of this agent's perception is such that trying to use observations directly and not break them down into individual features would result in a large input space and cause problems with generalisation.

U-Tree algorithm is reported as learning a reasonable policy on this task, however its performance is not perfect. This is attributed to a lack of good exploration. McCallum [1995b] reports the ordering of the splits used in the final learnt tree as often bizarre and certainly not optimal. Although each split can be shown to be utile, there is no procedure that ensures that they occur in any particular order. At what point they are added to the tree will depend on the agent's experience, and its model of the world. There is a circular problem that exploration depends in part on the agent's policy, which in turn depends on the current state estimation model, that is, the existing U-Tree model. Thus if a U-Tree's initial experience causes it to poorly divide up the environment, this may handicap both future exploration, and the remainder of the tree's development. As the U-Tree algorithm is currently designed there is no mechanism which would correct structural errors once they are introduced into a tree.

As mentioned at the start of this section (section 3.1.8), McCallum [1995b] argues for a continuum of actions: "Although the terms "manipulative actions," "perceptual action," "overt attention" and "covert attention" are useful, we cannot draw firm boundaries between them." However, we consider that even U-Tree makes a distinction between actions used to render the space Markovian and those used to execute tasks. U-Tree, in common with all fixed-SE approaches, has two stages, the first of which attempts to render the internal state Markovian (a state-estimation stage) and a second stage that associates actions with the estimated internal states. The first stage uses perceptual actions which select among the features of the agent's observations and memories. These actions remain distinct and separate from the largely physical actions available to the second, reinforcement learning stage of U-Tree.

Criticisms of U-Tree are:

(i) the need to set additional parameters such as fringe depth, maximum history index and statistical significance. The fewer parameters a system designer has to manage the better,

(ii) like USM it is difficult for U-Tree to un-alias states that require linking more memories than the maximum depth of the fringe expansion. This parameter has

to be set appropriately by the designer, and can not be adapted to the task by the learning algorithm if the wrong depth is selected. Further, setting too large a value significantly slows learning due to the exponential growth in the number of possible fringe expansions.

(iii) it would appear to be more principled when deciding whether to extend the tree to carry out statistical comparisons between each related node of the fringe, rather than comparing them to their parent. This should result in better identification of significant splits, however it would result in significantly more tests, that is, n^2 tests compared to just n,

(iv) in practical tests, see chapter 8, we find that the growth of the tree is possibly unbounded. When presented with a cluster of aliased states that can be visited in a cyclic fashion, the tree appears to grow wildly trying to capture some perceived difference in utility between what actually are identical states.

3.1.9 Predictive state representations

The most recent development in the area of state-estimation is the Predictive State Representation (PSR) modelling approach [Littman et al., 2001; Singh, 2004]. PSRs, like POMDPs provide a generative model of an environment, but unlike POMDPs its representations are grounded in data, which gives it some of the simplicity of k-order Markov methods. The key idea behind PSRs and the closely related Observable Operator Models (OOMs) [Jaeger, 2000] is to use of predictions of observable outcomes of tests which could be performed on the world as state representations. An example test could be the execution of action $a1$ followed by action $a2$ from the current state. The PSR would then estimate the outcomes of such actions in terms of observations and thus produce a vector $\langle a1, o1, a2, o2 \rangle$. This vector captures information on PSR's estimated state of the world. If an agent can discover the appropriate tests to apply in order to uncover the states of the world, it can use a PSR to provide state-estimates against which actions can be learnt.

PSRs can be used to represent a larger class of dynamical systems than POMDPs, yet at the same time holds the promise of a more compact representation [Singh, 2004]. The major hope of PSRs is that learning a PSR model from observation data should be easier and less prone to the issues that afflict the learn of POMDP models (see section 3.1.8.1). However, at this stage algorithms for PSR learning are still in their infancy and limited to a sub class of task which have a *reset* action (which resets the world state) [Singh et al., 2003; James and Singh, 2004].

3.2 Basic Approach

This section looks at basic approaches that apply *direct* reinforcement learning algorithms with simple agents (of the form outlined in figure 2.4 on page 29). This type of agent associates actions directly with the observations that it receives. This is in contrast to the approaches described previously which used state-estimation techniques which attempted to map the agent's observations on some internal, hopefully Markovian, state representation. This section considers only basic approaches, section 3.3 considers applying direct reinforcement learning with modifiable approaches.

We structure our review by looking in turn at authors whose work has focused on applying basic approaches to non-Markovian problems.

3.2.1 Littman, Loch and Singh

Littman [1994] considered learning reactive policies without memory in partially observable environments. He introduced the useful concepts of *satisficing* and *optimal memoryless policies*. An optimal memoryless policy, or in the terminology adopted in this thesis, an optimal reactive policy is the *optimal* policy than can be achieved by a reactive agent. Using hill climbing and branch and bound techniques Littman [1994] showed that it is possible to find optimal memoryless policies for various grid world navigation problems.

Loch and Singh [1998] then showed that reinforcement learning using eligibility traces, specifically SARSA(λ), could learn such optimal memoryless policies in partially observable grid world navigation problems. It was this, and other promising, results which inspired the investigations of Singh et al. [1994]; Pendrith and McGarity [1998, 1997]; Perkins [2001, 2002b]; Perkins and Pendrith [2002] into the limitations of applying direct reinforcement learning to non Markovian tasks.

3.2.2 Singh, Jaakkola and Jordan

Singh et al. [1994] examines what limitations might exist when applying a simple agent using direct reinforcement learning to POMDP tasks. The paper demonstrates that hopes of a graceful degradation of the performance of reinforcement learning algorithms as the degree of non-Markovianness increases are misplaced. A simple example is constructed to show that the aliasing of just two states in a POMDP can lead to arbitrarily high absolute loss in the discounted future reward received by the agent.

Singh et al. [1994] considers the types of policy that exist and through theoretical examination of example POMDP problems establishes a ranking over the types of policies. The ranking from best to worst is:

(i) optimal reactive policies based on perceiving the *states* of the POMDP, that is, obtaining access to the underlying MDP,

(ii) non-stationary, reactive policies in a POMDP,

(iii) stationary, stochastic, reactive policies in a POMDP,

(iv) stationary, deterministic, reactive policy in a POMDP.

Having established this ordering Singh et al. [1994] look at the issues involved in learning stationary, stochastic, reactive policies for POMDPs, such policies being possibly better than deterministic policies, but hopefully less computationally expensive than searching the space of non-stationary policies. The underlying states of the POMDP

are considered inaccessible to the agent so it is not possible to consider learning the optimal reactive policy for the underlying MDP.

In attempting to learn stationary, stochastic, reactive policies Singh et al. [1994] considers what is meant by an optimal policy. "In a discounted[4] MDP, an optimal policy is simply one that maximises the value of each state simultaneously. Unfortunately, in discounted POMDPs it is no longer possible to define optimal policies in the same way." Two examples demonstrate that for the class of POMDPs considered in this paper there need not be a stationary policy that maximises the value of each *observation* simultaneously or maximises the value of each *state* in the underlying MDP simultaneously. To overcome this the value of a policy is defined by converting the value of each observation under the policy into some weighed score for that policy, that is, $\sum_{X \in \chi} P_X V_\pi(X)$ where P_X is some weight or measure of the importance of the observation X drawn from the set of possible observations χ. Possible choices for P_X are the probability of starting in observation X or the probability of receiving observation X at a given point in time. Singh et al. [1994] considers the former pays too much attention to the starting observation and instead considers the second option which is equivalent to maximising the average payoff per time step. Using this definition of optimal policy Singh et al. [1994] set out a framework for assigning values to observations, which does not involve the Markov assumption.

The above work, Singh et al. [1994], is then built on in Jaakkola et al. [1995] which presents a theoretical description and analysis of a Monte Carlo based algorithm that uses time averaged rewards to learn stationary, reactive policies on POMDPs. The batch version of this algorithm is shown to converge to a local maximum. The paper focuses on the theoretical analysis of this algorithm and no empirical evidence of its abilities is presented.

[4] Conventional reinforcement learning framework where future rewards are discounted.

3.2.3 Pendrith and McGarity

Pendrith and McGarity [1998, 1997] carry out mathematical analysis of the stability of *optimal* reactive policies under various commonly used direct reinforcement learning algorithms.

Two useful properties that exist when applying reinforcement learning to Markov decision processes (MDPs) are that:

(i) there are no local maxima that learning can get trapped in (see convergence proofs for reinforcement learning, Watkins and Dayan [1992]; Dayan [1992]; Singh et al. [2000]),

(ii) "an optimal policy has the property that all state values are maximal" [Pendrith and McGarity, 1997].

The second property is used in the definition of an optimal policy π^*, for example, from Sutton and Barto [1998][p.75], "although there may be more than one [optimal policy] they share the same state-value function, called the optimal state-value function, denoted V^*, and defined as $V^*(s) = \max_\pi V^\pi(s)$ for all $s \in S$ [where S is the set of all possible states]." "In MDPs, a policy is optimal if and only if it is greedy with respect to its action values" [Perkins, 2002b]. Given the above properties of MDPs it can be seen that acting to maximise the local reward, that is, using the policy update function $\pi \leftarrow \arg\max_a Q(s,a)$ eventually results in the optimal policy. However, when considering a non-Markov decision process (NMDP) the above properties no longer hold.

To show the first point Pendrith and McGarity [1998, 1997] demonstrate by use of a simple example that local maximum policies are possible in NMDPs. The two papers demonstrate the failure of the second property for NMDP by adopting a definition of optimality that differs from Sutton and Barto [1998][p.75] but appears to be reasonable. An optimal policy π^* is defined as a policy which generates the maximum expected return, that is, $\pi^* \in \arg\max_\pi J(\pi)$, the expected return $J(\pi)$ being calculated as

the expected discounted reward over all the possible paths that exist under this policy, weighted by the probability of each path occurring. Based on this definition both papers show through the use of examples that for NMDPs the maximisation of individual state-values can result in a non optimal reactive policy.

Having demonstrated that the conventional definition of an optimal policy as used with MDPs does not hold for NMDPs, Pendrith and McGarity [1998, 1997] introduce the concept of learning equilibria. Being a learning equilibria is a property that optimal MDP policies possess and would be desirable for optimal policies in NMDPs; "A learning equilibrium has the property that if you replace the current state [or state-action pair] value estimates with the expected value of those estimates given the current policy and the learning method being used, then the policy remains unchanged." For MDPs it is relatively easy to see that the optimal policy is also a learning equilibrium given the *policy update* rule $\pi \leftarrow \arg\max_a Q(s,a)$ and, for example, the *value update* used in in the Q-learning algorithm, that is, $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$.

Having demonstrated that there exist NMDPs where the *individual* state-action values are not necessarily maximised under the optimal reactive policy, Pendrith and McGarity [1998, 1997], further show, that for

- temporal difference learning with eligibility traces (TD(λ)) where $\lambda < 1$,
- 1-step Q-learning algorithm,
- 1-step SARSA,
- any algorithm using discounted rewards, that is, where $\gamma < 1$

that there exist NMDPs where the optimal, reactive policy is not guaranteed to be a learning equilibrium. Thus there can be no guarantee that these learning algorithms will converge to the optimal reactive policy for a NMDP.

They do show, however, that optimal reactive policies are a learning equilibrium on NMDPs for the first-visit Monte Carlo algorithm when using undiscounted returns, that

is, $\gamma = 1$. Note that this result does not imply that these optimal reactive policies will be the only maxima; local maxima will still exist in NMDPs when using first-visit Monte Carlo updates. In addition, the existence of optimal policies as learning equilibria is a necessary but not sufficient condition to ensure convergence. The guarantee is, that if the algorithm happens to learn state-action values that describe an optimal reactive policy, then undiscounted first-visit Monte Carlo updates will ensure that the state-action values will remain at this equilibrium point.

An interesting question is whether this guarantee extends to TD(λ) when $\lambda = 1$ and the discount rate γ is also 1. Although TD(1) has equivalence to Monte Carlo approaches its update is not directly equivalent to first-visit Monte Carlo methods. Though as noted by Singh and Sutton [1996] the use of *replacement* eligibility traces [Sutton and Barto, 1998, pp.186–189] cause the updates to more closely resemble *first-visit* Monte Carlo as opposed to *every-visit* Monte Carlo methods. We also note that Pendrith and McGarity [1998, 1997] assume a policy update of the form $\pi \leftarrow \arg\max_a Q(s,a)$. Different policy update rules would not necessarily have the same problems in NMDPs.

Questions raised by this work are:

(i) How well does first-visit Monte Carlo perform learning policies on NMDPs?

(ii) What exploration strategies can we utilise with the first-visit Monte Carlo algorithm and still hope for convergence to learning equilibria? We note the definition of a learning equilibrium requires that the learning algorithm follow the learnt policy. This requires all exploration to have been curtailed.

(iii) This work specifically considers optimal reactive policies. If would be useful to investigate the properties of satisficing reactive policies on NMDPS. For example, what is the relationship between satisficing policies and learning equilibria? Are all learning equilibria also satisficing policies? Alternatively, are satisficing policies learning equilibria?

(iv) Given a policy update rule and value update formula, is it possible to split NMDP tasks into those for which the optimal reactive policy is a learning equilibrium and those for which it is not?

(v) Can similar partitions be found for satisficing policies?

(vi) Is it possible to change the policy update such that it maximises the global policy?

Chapter 5 examines items (i) and (ii) above, the remainder are left as future work.

3.2.4 Perkins

Perkins [2001, 2002b] develops the work presented in Pendrith and McGarity [1998, 1997] by observing that the conditions for which an optimal policy is a learning equilibrium can be extended to the case where discounted rewards are used, provided that the state value calculation $V^{\pi}(s)$ is subtly redefined.

As with Pendrith and McGarity [1998, 1997] the key to achieving the stability of the optimal policy is to ensure that the policy value is calculated in such a way that it is consistent with the assumption made in using the policy update equation $\pi \leftarrow \arg\max_a Q(s,a)$. The assumption being that the maximum valued policy is achieved by maximising the value of every state. In the case of Pendrith and McGarity [1998, 1997] this was achieved by using first-visit Monte Carlo updates with undiscounted $\gamma = 1$ returns. In the case of Perkins [2001, 2002b] first-visit Monte Carlo is again used but with discounted returns $\gamma < 1$. To ensure consistency between the value of each state (or state-action pair) and its contribution to the value of the policy, the value of a state in Perkins [2001] is redefined such that it is discounted using the *same* factor that will be applied when calculating its contribution to the policy value. That is, given observation *o*, "the observation-action value is the portion of the *policy's* expected discounted return that follows *o*, not the discounted return following *o* itself" Perkins [2002b]. Perkins [2001, 2002b] present a proof that the optimal policy is a learning equilibrium when their modified calculation of state values (or state-action values) is used. The result of this proof can also be interpreted as showing that a "policy is locally optimal if and only if it is greedy with respect to its action values (as we [Perkins [2002b]] have defined them)."

Based on the above arguments Perkins [2001, 2002b] present the Monte Carlo Exploring Starts for POMDPs (MCESP) algorithm, this is a modified form of the Monte Carlo algorithm. Perkins [2001] claims that if MCESP "converges to a policy π, (that is, if π is the current greedy policy at some time and remains the greedy policy forever after) and if standard stochastic approximations conditions hold [...] then π is locally optimal with probability 1." Further, Perkins [2002b] claims "that special cases of our algorithms can achieve probability one convergence to locally optimal policies in the limit."

The MCESP algorithm as presented in Perkins [2002b] allows for various design choices to be made. Based on ideas from reinforcement learning and stochastic optimisation three possible variations are implemented and compared experimentally: (i) MCESP-SAA (Sample Average Approximation); (ii) MCESP-PALO using Greiner [1996]'s PALO (PAC hill-climbing) algorithm; (iii) MCESP-CD (constant delta[5]). For the third version, MCESP-CD, Perkins [2002b] claims convergence to δ_0-locally optimal policies (where δ_0 is some very small value) with probability 1, given suitable decay on the learning rate α. A policy is δ-locally optimal if and only if $Q^{\pi}_{o,\pi(o)} + \delta \geq Q^{\pi}_{o,a}$ for all o and a. The proof of this theorem is unfortunately not included in the paper.

The experimental results presented are for a single grid world problem, Parr and Russell's grid world. MCESP-CD proved to be the best of the three MCESP algorithms tried. It converged to the globally optimal policy 995 times out of 1000. (The policy space is such that it consists of a plateau of minimal-valued "bad" policies some of which are locally optimal and a single peak corresponding to the unique globally optimal policy.) Interestingly SARSA(0.9) performs better reaching the globally optimal policy 100% of the time. However Perkins [2002b] report that SARSA(λ)'s behaviour was very sensitive to exploration and only the "exact design" given in Loch and Singh [1998] behaved well.

In reporting these results Perkins [2002b] note that further experimental work is required to examine how well the performance of MCESP-CD generalises. We report

[5] Perkins [2002b] uses the variable epsilon, that is, "constant epsilon" and the abbreviation MCESP-CE. We change this to *delta* in this thesis to avoid confusion with the unrelated exploration parameter epsilon.

experimental results, in this area, in section 6.

Appendix B of Perkins [2001] contains an interesting result using a counter example to show that there is no guarantee of convergence of SARSA(λ) on POMDPs for $\lambda \in [0,1]$. This is an important result as previously it had only been shown for $\lambda \in [0,1)$ leading to conjectures that for any POMDP, SARSA(λ) might be convergent for λ sufficiently close to 1. The example is shown to hold for both accumulating and replacement traces when using ε-greedy as the exploration policy. This example raises interesting questions; (i) does it also hold for Watkins's Q(λ), or, if not, can a similar non convergent example be found? (ii) what about convergence when using action-select approaches other than ε-greedy? In answer to the second point we observe that the proof relies on the fact that ε-greedy exploration is discontinuous. Perkins and Pendrith [2002], which is reviewed below, provides an analysis which shows that convergence is possible when continuous action selection methods, such as *softmax* [Sutton and Barto, 1998, pp.30–31], are employed. In addition, in chapter 5, we present an analysis of a specific example which suggests that there may exist a guarantee of convergence when an agent is acting greedily.

3.2.5 Perkins and Pendrith

Perkins and Pendrith [2002] presents mathematical proofs that for episodic tasks and assuming ergodic POMDPs, that if an agent uses a continuous exploration strategy (that is, softmax) with Q-learning or SARSA updates, then stochastic fixed points in the space of observation-action value functions are guaranteed to exist. No guarantees are given about convergence to these fixed points or the quality of the policies that these fixed points represent. In fact, in the example POMDP given in Perkins and Pendrith [2002], softmax was observed to converge to the worst of the two possible policies. In contrast, although ε-greedy (with fixed value of ε) failed to converge, cycling between the two policies, it spent the majority of its time following the optimal policy. It is worth noting that the existence of fixed points for the state-action values do not imply a stationary policy, the fixed points are stochastic and thus Q values could vary around these fixed points, resulting in an agent that alternates between policies.

3.2.6 Baird

[Baird and Moore, 1999] describes an algorithm, VAPs, that uses gradient descent to converge over both value and policy, the weighting given to each being a controllable factor. A simplified version of the VAPs algorithm can also be found in Peshkin et al. [1999].

Such gradient based approaches enjoy superior theoretical properties on non-Markovian problems, however "evidence to date suggests that these algorithms learn much more slowly than action-value based reinforcement learning algorithms such as SARSA(λ)" [Perkins, 2002b].

3.3 Modifiable Approaches

This section looks at examples of work where the *modifiable approach* has been used to extend simple agents. As observed in section 2.3, from the core-learning-algorithm's point of view it will be attempting to learn a *direct reactive* policy irrespective of whether a *basic* or *modifiable approach* has been applied. Thus the results from section 3.2 also apply to modifiable approaches. The key difference is that the modifiable approach allows the core-learning-algorithm more latitude in terms of the available actions and viewpoints which can be perceived. There are two main modifiable approaches that have been presented in the literature, one is providing agents with active perception, the other is providing agents with internal memory.

3.3.1 Active perception agents

The main example of an active perception agent being used in the absence of state-estimation is to be found in Whitehead [1992] where *Q*-learning was used with a simulated, active-perception, robotic arm. Whitehead [1992] report that *Q*-learning failed to learn the task, performing only slightly better than selecting actions at random. This

failure was thought to be due to the inability of Q-learning (or any reinforcement learning algorithm using 1-step backup) to learn stable policies in the presence of perceptual aliasing, the perceptual aliasing in their case being caused by the design of the active perception system. In chapter 4 we dispute this reasoning, showing that the theoretical example he presented is learnable if the agent acts greedily, that is, after some period ceases all exploratory behaviour.

3.3.2 Internal memory agents

The other main examples of modifiable agent are agents that contain some form of internal memory that is modifiable by the direct reinforcement learning algorithm, and the results of the modifications impact on the observations received by the learning algorithm.

Littman [1994]; Peshkin et al. [1999] and Lanzi [2000] present agents which have access to internal memory registers which are under the control of the same learning algorithm which selects the physical action that moves the agent around the environment. The state of the memory register is added to the external observation available from the world before being presented to the learning algorithm. Figure 2.9 (on page 2.9) shows an agent of the type described. The presence of memory bits which are under the control of the learning algorithm allow it to carry forward information and thus differentiate between otherwise aliased observations.

Littman [1994] uses an agent with a 1 bit memory and learns policies using a branch and bound technique. He successfully demonstrated that a single bit memory could be used to learn a reactive policy for McCallum's M maze (also known as McCallum's cheese maze). For a simple agent whose perceptions are limited to the four adjacent squares that lie in the principal compass directions, there is no satisficing stationary reactive policy for McCallum's M maze.

Peshkin et al. [1999] and Lanzi [2000] demonstrate that similar results can be obtained using reinforcement learning algorithms; Peshkin et al. [1999] using SARSA(λ), Lanzi [2000] using *Q*-learning and Watkins's Q(λ).

Lanzi [2000]; Lanzi and Wilson [2000] make two interesting observations. The first they refer to as *aliasing on the payoffs* , which they use to refer to the *increase* in the number of policies that converge to both satisficing and optimal solutions as the level of exploration used in the experiment is *reduced*. The experiments are run at fixed values for ε using ε-greedy exploration (that is the agent follows its current policy with a probability of $1-\varepsilon$ and selects an action at random with probability ε). This observation correlates with the result observed for a decaying schedules of ε as used in this thesis, see chapter 5.

Their second observation is that a redundant number of memory bits can improve learning capabilities in perceptually aliased situations. Lanzi and Wilson [2000] speculate that more internal memory bits mean more optimal solutions and thus more chance that an agent can learn one of them. This point is similar to a point we raised in section 1.1, that enhanced agents not only have access to better reactive policies, but have access to more satisficing policies, which we believe should increase the likelihood of learning. This point is examined in chapter 4.

3.3.2.1 HQ-learning

Wiering and Schmidhuber [1997] present an algorithm, HQ-learning, which assumes that most real world POMDPs can be decomposed into a series of Markovian sub-tasks. An ordered sequence of agents are used to learn reactive policies for each sub-part of the task, with only one agent being active at a time. Decomposition of the task is done by finding an observation for each of the agents which indicates when that agent's sub-task has been completed. These "sub-goal observations" are found using ε-greedy exploration over a set of observation-values that are maintained for each agent. This set of observation-values are known as the *HQ-values*. Each agent's reactive policy is learnt using off-line Q-learning or Q(λ) updates.

The design of HQ-learning is such that it can be seen as a form of internal memory agent. In particular, it can be seen as an internal memory agent that in place of bits has a counter for its memory and has an action that increments this counter. This is

equivalent to the memory in HQ-learning which consists of a pointer that indicates the agent which is currently active, this pointer being incremented when sub-goals are reached. The major difference, in comparison to the internal memory agents discussed previously, is that the updating of memory is decoupled from the selection of physical actions. In the case of the agents presented by Littman [1994]; Peshkin et al. [1999] and Lanzi [2000] each memory-modification action *has* to be coupled with a physical action, which typically results in a doubling of the action space, that is, each agent has one set of physical actions that modify its memory and a complimentary set that does not. By not coupling incrementation of its "memory" with physical actions, HQ-learning avoids this multiplicative increase in the action space.

A significant limitation of HQ-learning is that it can only learn policies for episodic tasks. HQ-learning requires the number of agents to be specified before the algorithm is run. As each agent is only selected once and in a fixed sequence, HQ-learning will eventually run out of agents when applied to a non-episodic task. A further drawback of HQ-learning is that when specifying the number of agents, some idea is required of the number of Markovian sub-tasks which will be required. Specifying too few agents could result in either poor policies or a complete inability of HQ-learning to learn any satisficing policies for the task. This is similar to providing insufficient memory bits for a given task to an internal memory agent. On the other hand, specifying too many agents for HQ-learning can result in overly complex and overly long satisficing policies.

3.4 Chapter Summary

This chapter reviews literature in the area of applying reinforcement learning to partially observable tasks, selectively presenting the main ideas that have influenced this field. The review was structured around the three approaches that we identified in section 2.3; the *fixed-SE*, *modifiable* and *basic* approaches. We present below summaries of the literature for each of these three categories, followed by a general overview of this issues arising from the literature.

3.4.1 Fixed-SE approaches

Fixed-SE approaches focus on achieving good *state-estimation* from the observations, that is, they attempt to learn mappings from the observations which the agent perceives to an internal Markovian representation against which a policy can then be learnt. State-estimation using active perception has largely been constrained to work carried out by Whitehead [1992] and Tan [1991b], which focused on achieving a consistent view for every significant state of the world through the selection of perceptual actions. State-estimation using the history of observations seen by the agent has been much more widely studied and a significant number of algorithms proposed. One of the most principled approaches, modelling environments as POMDPs, has floundered as learning POMDPs from observation data has proved to be unreliable [Shatkay and Kaelbling, 1997; Hartley, 2004]. Recently developed Predictive State Representation (PSR) approaches have the flexibility of POMDPs, and hopefully learning PSRs from observation data will prove more tractable than on-line learning of POMDPs. However at this time algorithms for learning PSR models are still in their infancy and very much limited to a sub class of tasks.

The best performing approaches in this area are those which assume that aliased states can be resolved using the k most recent observations, that is, k-Markov model style approaches. In applying such approaches it is preferable to use algorithms which allow k to vary across the state space, as theoretically they only provide extra details where required and allow generalisation elsewhere in the space. Sophisticated k-Markov model style approaches, with variable k, were presented by McCallum [1995b], for example, USM and U-Tree.

3.4.2 Basic approaches

Much work has been done in this area on identifying the existence of non-Markovian tasks where no guarantees of convergence can exist for direct reinforcement learning algorithms such as Q-learning, SARSA and SARSA(λ). Despite these results many

examples exist in the literature of such algorithms performing well in non-Markovian environments. This suggests that there might be a sub class of non-Markovian environments, yet to be identified, over which good results can be consistently achieved. We suspsect, however, that this will require relaxing the commonly used requirement of learning optimal policies in favour of learning satisficing policies.

Recent work by Pendrith and McGarity [1997, 1998]; Perkins [2001, 2002b] indicates that convergence is dependent on the style of policy and value updates employed. Based on this Perkins [2001, 2002b] present the Monte Carlo Exploring Starts for POMDPs (MCESP) algorithm, for which Perkins [2002b] claims "that special cases of our algorithms [for example, MCESP-CD] can achieve probability one convergence to locally optimal policies in the limit."

3.4.3 Modifiable approaches

Little work has been published where *active perception* has been combined with *direct* reinforcement learning algorithms, the main result being that of Whitehead [1992] where he reported the failure of Q-learning to learn a blocks world task involving a simulated robotic arm equipped with active perception.

The majority of positive results for *modifiable approaches* relate to agents with internal memory registers that are under the control of the direct reinforcement learning algorithm. Both Peshkin et al. [1999] and Lanzi [2000] report good results with agents learning tasks in a variety of partially observable grid world environments.

Lanzi [2000]; Lanzi and Wilson [2000] make two interesting observations. Experiments reported in Lanzi [2000] indicate the success with which modifiable approach agents converged to either optimal or satisficing solutions with differing levels of exploration. The results suggest that the lower the chances of an agent selecting an exploratory action the easier it is for its policy to converge to either an optimal or satisficing policy. The second observation is that a redundant number of memory bits can improve learning capabilities in perceptually aliased situations. Lanzi and Wilson

[2000] speculates that more internal memory bits means more optimal solutions and thus more chance that an agent can learn one of them. We return to both these points in the experiments presented in this thesis.

3.4.4 Summary table

A summary table listing all the methods discussed in this chapter, and giving an indication of each algorithm's strengths is presented in table 3.1.

3.4.5 Issues arising from literature review

There is little significant work using active perception with *direct* reinforcement learning algorithms. Where active perception has been used it is generally in conjunction with state-estimation approaches. Given the good results that have been achieved when applying agents that employ internal memory to non-Markovian tasks, it would seem that active perception approaches, which share similar design features (and hence the common modifiable approach classification) should work equally well. We present the results of such approaches in the experiments presented in chapters 4 and 8.

In terms of improving the learning of reactive policies using direct reinforcement learning algorithms, a series of issues are raised by this review:

- What is the impact of exploration on convergence to satisficing policies? This requires consideration of both the effect of the amount of exploration an agent does, as in the probability of executing an exploratory (non-policy) action at any one time. Also, the choice of action selection strategies used, that is, whether they are continuous over the action space, such as softmax, or discontinuous such as ε-greedy. Does the existence of stochastic stationary state-action value points on non-Markovian problems for continuous action selection translate into stationary policies? We examine these issue in chapter 5.

	can work in stocastic/noisy environments	can work with negative rewards	can work with continuous state spaces	makes no distinction between overt perceptual and physical actions	combines information from multiple perceptual actions to determine state	sub−divides observations into features	**no** requirement that each differently utile state can eventually be distinguished	policy learnt accounts for cost of perceptual actions	examines observation−action sequences of **arbitary** length to disambiguate states	differentiates between states based on their utility to the task	algorithm is **not** computationality expensive	maintains a instance list of all previous actions, rewards and observations	Additional Notes
Lion Algorithm										✓	✓		
G−algorithm	✓	✓		✓		✓	✓	✓		✓	✓		
CS−QL					✓			✓	✓	✓	✓		
Window−Q	✓	✓	✓	✓	(✓)		✓	✓		✓	✓		
recurrent−Q / LSTM	✓	✓	✓	✓	(✓)		✓	✓	✓	✓	✓		
recurrent−model	✓	✓	✓	✓	(✓)		✓	✓	✓		✓		
k−order Markov methods	✓	✓		✓	(✓)		✓	✓			✓		
Perceptual Distinctions approach (PDA)	✓	✓		✓	(✓)		✓	✓	✓			✓	} Simultanious learning of POMDP enviro−ment model and agent policy often results in instability and trapping in local minima
Utile Distinction Memory (UDM)	✓	✓		✓	(✓)		?*	✓		✓		✓	
Nearest Sequence Memory (NSM)		✓		✓	(✓)		✓	✓	✓		✓	✓	
Utile Suffix Memory (USM)	✓	✓		✓	(✓)		?*	✓		✓		✓	* = utile test appears to continually extend memories trying to distinguish between such states
U−Tree	✓	✓		✓	(✓)	✓	?*	✓		✓		✓	
PSR / OOMs	✓	✓		✓	(✓)		?	✓	✓		?	?	learning PSRs is an area of ongoing work
eligibility traces, *i.e.* SARSA(λ)	✓	✓		✓			✓	✓			✓		good but sensitive to exploration/parameters
MCESP−CD	✓	✓		✓			✓	✓			✓		performance close to SARSA(λ)
gradient decent approaches, *i.e.* VAPS	✓	✓		✓			✓	✓			✓		much slower than SARSA(λ) to converge
Active Perception Agents	✓	✓		✓			✓	✓			✓		
Internal Memory Agents	✓	✓		✓			✓	✓			✓		
HQ−learning	✓	✓		✓			✓	✓			✓		episodic tasks only

(✓) = combined indirectly through "memory" of previous perceptual and physical actions

Table 3.1: Summary of reinforcement learning methods reviewed in this chapter indicating each algorithm's strengths

- The Monte Carlo approaches proposed by Pendrith and McGarity [1997, 1998]; Perkins [2001, 2002b] are demonstrated to be stable when following optimal policies, but how well do these approaches converge on non-Markovian problems? How does their performance compare to SARSA(λ) whose performance is viewed by some as a high standard to beat [Perkins, 2002b; Loch and Singh, 1998]. Chapter 6 presents results relating to these questions.

- Is it possible to improve direct reinforcement learning algorithms' performance on non-Markovian tasks, through changes to their approach (i) to exploration, (ii) policy updates or (iii) value updates. We take a small step in this direction with the introduction of Consistent Exploration, see chapters 6 and 7.

- It would be interesting to try to extend the theoretical analysis undertaken for the convergence/non-convergence of direct reinforcement learning algorithms to consider convergence to *satisficing* reactive policies on non-Markovian environments. It would also be useful to try to identify if there exist sub-classes of non-Markovian tasks which are amenable to direct reinforcement learning. Although section 7.2 considers some theoretical limitations of a new reinforcement algorithm, theoretical consideration of these two specific points is left to future work.

Chapter 4

Learning with Active Perception

This chapter presents initial experiments which show the results of applying popular, reactive, reinforcement learning algorithms to problems which contain partially observable states.

The main aim, and one of the hypothesis, of this thesis is to demonstrate that in the field of reinforcement learning, equipping an agent with active perception is an effective approach to learning in partially observable worlds. Although ultimately we would like to implement active perception based reinforcement learning on robotic platforms, an aim which goes beyond the scope of this thesis, we initially want to better understand the problems caused by perceptual aliasing, that is, the confusion of two or more states of the world due to the unavoidable limitations of a robot's sensors, a problem that can be exacerbated when a robot has active control over its sensory input; many places can look the same when you are staring at the wall.

To study the fundamentals of the problem we consider simple simulated agents moving around deterministic grid worlds, such as Sutton's Grid World (figure 4.1). By limiting the state information that this simple agent can observe, it faces similar problems to those encountered by a real mobile robot, the latter's sensing abilities being naturally limited by the design and type of sensors that it is equipped with. Section 4.1.1.2 describes the Sutton's Grid World task in more detail and explains how it contains

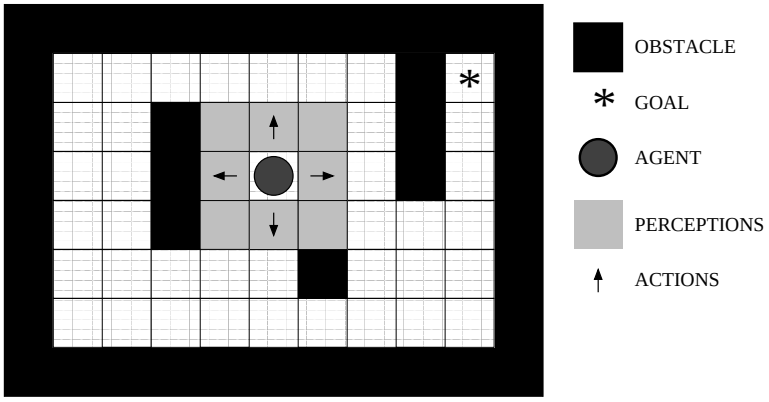


Figure 4.1: Sutton's Grid World and example agent. The agent shown "The Eight Adjacent Squares Agent" receives observations which are formed by testing if each of the eight squares surrounding its current location contain obstacles

the many different problems that can occur when applying reinforcement learning to partially observable worlds.

Tasks that contain aliased or hidden state can be modelled as partially observable Markov decision processes (POMDPs). Work reviewed in the previous chapter (section 3.2) outlined the lack of theoretical performance guarantees in this domain and presented examples of POMDP problems where commonly used reinforcement learning algorithms can provably not converge. Despite the existence of such a negative result [Pendrith and McGarity, 1998; Perkins, 2001] for the learning algorithm SARSA(λ) [Sutton and Barto, 1998, p181], empirical tests using this algorithm have demonstrated good results on many of the POMDP tasks on which it has been tested [Loch and Singh, 1998; Perkins, 2002b]. In fact Perkins [2002b] refers to results with SARSA(0.9) as a "high standard to compare to," and Loch and Singh [1998] "that SARSA(λ) may be hard to beat in problems where there exists a good policy that maps observation space to actions." These results motivate our initial focus on the use of SARSA(λ) and the closely related Watkins's Q(λ) algorithm in these initial studies.

Section 4.1 considers an agent with a perception system that is limited and fixed. It is *limited* in the sense that the information provided does not fully reflect the state of

world – in other words, there exists a many-to-one mapping between the state of the environment, in this case the agent's location, and the observations that the agent perceives – and *fixed* in the sense that the agent has no overt control over what observation it receives in any given state, that is, the observation obtained is determined solely by the state of the world, which in this case is determined by the agent's location.

Section 4.2 examines the ability of direct reinforcement learning algorithms to learn policies with an agent identical to that used in section 4.1, but equipped with the additional ability of being able to request information from some form of oracle. The oracles were conceived as a simplified form of active perception that could provide perfect information to the agent when requested. The addition of oracle's (and also later active perception) changes the mapping that occurs from the world's state to the observation received by the learning algorithm. The mapping is now many-to-many as more than one view is possible of each world state. It is therefore useful to show that the results of Loch and Singh [1998] which we replicated in section 4.1 can be extended to this many-to-many mapping. We identify in section 2.3 that active perception agents fall in the same category as internal memory agents as implemented by Littman [1994]; Peshkin et al. [1999] and Lanzi [2000]. One feature of the Littman [1994]; Peshkin et al. [1999]; Lanzi [2000] approaches is that disambiguating actions (change internal memory registers) are coupled with the agent moving location (that is, physical actions). We examine whether a similar coupling of disambiguation and movement is required for active perception agents.

Based on positive results from section 4.2, the penultimate set of experiments in this chapter considers an agent equipped with active perception actions. In this case not only is there a many-to-many mapping from the states of the worlds to observations but, unlike the oracles, the additional observations available will not necessarily provide a unique representation for every state of the world. The results of applying popular reinforcement learning algorithms to this type of agent are presented in section 4.3.

Finally, section 4.4 presents a problem for which it is known that no stationary, deterministic reactive policies exits. We show how extending the agent's ability with the addition of active perception can allow a stationary, deterministic reactive policy to be

learnt on this task.

In a POMDP, in contrast to a Markov decision process (MDP), a clear separation exists between the state of the world and the information that an agent can observe about that state. To reflect this distinction we adopt the following conventions in this chapter, and throughout this thesis. *State* is used to refer to the true underlying state of a problem and *observation* to refer to the information about the state that is received by the learning algorithm. It is against observations that reactive learning algorithms associate actions and values. This is in contrast to most reinforcement learning work which, because of its focus on MDPs, uses the word state to encompass both of the above meanings. Thus we refer to learning observation-values, not state-values, and action-values $Q(o,a)$ are indexed with respect to the observation o rather than the state s.

4.1 Agent with Limited Fixed Perception

The aim of these initial experiments was to replicated results obtained by Whitehead [1992] and Loch and Singh [1998] that:

- 1-step reinforcement learning algorithms are *not* able to learn policies which are both stable and optimal, when the task involves perceptually aliased states;
- Reinforcement algorithms that use eligibility traces can, however, learn optimal reactive[1] and stable policies for the same task.

[1]Optimal reactive policies are the best policies that can be found by a reactive agent which is not equipped with memory [Littman, 1994]. These optimal reactive solutions can be arbitrarily sub-optimal compared to memory based solutions [Singh et al., 1994].

4.1.1 Experimental setup

4.1.1.1 Agents

We conducted experiments using two types of agent, both of which move around and obtain observations from simple grid worlds.

(i) An *Absolute Position agent* who receives observations which uniquely represent its location in the grid world. Each observation consists of the agent's location in Cartesian coordinates. Note that the coordinate information contained within each observation is not used by the learning algorithm, the observations simple provide unique labels for each location. That the location labels are $\langle 1,1\rangle$, $\langle 1,2\rangle$ and $\langle 1,3\rangle$ is of no significance to the learning algorithm, they could equally be Alison, Charlie and Fred provided that a unique label is generated for each location.

(ii) An *Eight Adjacent Squares agent* whose observations are formed by testing if each of the eight squares adjacent to its current location contain obstacles, see figure 4.1.

The importance of these two agents is not in the detail of what they can observe but that the Absolute Position Agent obtains a unique observation for every location in the grid worlds that it occupies, while the latter, the Eight Adjacent Squares Agent, can obtain the same observation for multiple locations in a grid world. The latter agent's observations more closely match those that would be obtained by an embodied and situated agent which was located in the real world, that is, the sensations it receives from the world are limited to its local environment.

4.1.1.2 Grid worlds

We use two grid world navigation problems:

(i) Sutton's Grid World, see figure 4.1;

(ii) A simple 1-D example (figure 4.3) devised by Whitehead [1992, pp73–78] to illustrate the problems partial observability causes to 1-step Q learning.

Sutton's grid world

Sutton's grid world, figure 4.1, consists of a 9×6 grid containing various obstacles and a goal in the top right hand corner (indicated by an asterisk). An agent in this world can choose between four physical actions; move north, south, east and west. State transitions are deterministic and each action moves it one square in the appropriate direction. If an agent tries to move towards an obstacle or wall it is not allowed to move, that is, its location remains unaltered, although it receives the same penalty as if the action had succeeded. On executing an action the agent receives a reward of 0 if that action results in it immediately arriving at the goal location, otherwise it receives a penalty of -1. When the agent reaches the goal, it is relocated to a start location selected at random using a uniform distribution across all locations that do not contain obstacles.

Sutton's original problem was modified by Littman [1994] and made into a partially observable problem through the introduction of an agent whose observations are local, that is, the *Eight Adjacent Squares Agent* as shown in figure 4.1.

We choose to use this partially observable version of Sutton's Grid World as it is sufficiently complex to contain examples of the different problems that can occur when applying reinforcement learning to POMDPs. For the Eight Adjacent Squares Agent there are multiple locations that result in the agent obtaining the same observation, for example, the locations labelled 0, 2, 7 and 148 in figure 4.2. These aliased locations can present different challenges to reinforcement learning algorithms:

(i) The lower plot in figure 4.2 shows an example optimal policy. Examining this it can be seen that to achieve an optimal policy the agent needs to select a different

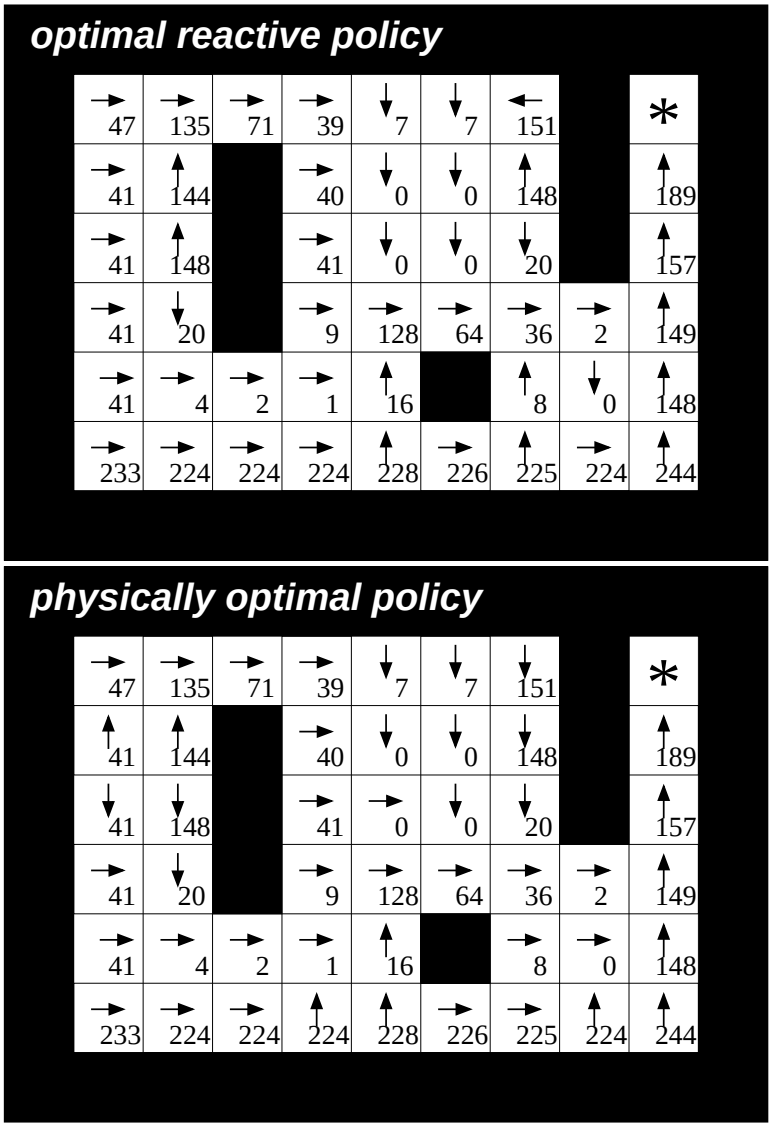


Figure 4.2: Sutton's Grid World. Values indicate observations obtained by an agent with fixed perception sampling the eight surrounding squares (Eight Adjacent Squares Agent). Filled black squares are obstacles or walls, and the goal is indicated by an asterisk (*). In the top plot arrows show an example optimal reactive policy (from Loch and Singh [1998]). Lower plot shows an example optimal policy

action in one of the three locations labelled 148. It needs to move north in location labelled 148 that lies directly below the goal, while in the location labelled 148 which lies just to left of the goal, on the other side of an obstacle, it needs to move south.

(ii) There are aliased states such as the locations labelled 2 where the same action suffices in both states. However, direct reinforcement learning algorithms will arrive at some observation-value for location 2 which will be a weighted average of the independent values of the two underlying states. As one occurrence of the aliased state is relatively close to the goal and the other somewhat distant this averaging of the observation-value will be significant. The average observation-valued learnt for observation 2 can then cause distortion in the observation-values of surrounding non-aliased locations, possibly leading to erroneous action selection in the policy that is learnt.

Finally, as will be demonstrated in chapter 5, there exists in the Sutton's Grid World task a number of *local minima* policies which are not *satisficing* policies, and to which reinforcement learning algorithms could conceivably converge.

Simple 1-D example

The simple 1-D example world consists of a 1×8 grid as shown in figure 4.3, with the goal at the far right hand side. An agent in this world is started in one of the non-goal states, that is, states 0–6, and its aim is to learn a policy that takes it to the goal. The agent can select between two physical actions; move east or move west. State transitions in this world are deterministic and the two actions move the agent one square in the appropriate direction. The agent is not allowed to move past either the far left hand end or far right hand end of the world; if it tries to do this its location remains unaltered. On reaching the goal the agent receives a reward of 5000. Moving to any other location yields zero reward.

The arrangement of the wall and gaps above and below the labelled squares in figure 4.3, play no part in the actions that the agent is allowed to execute. They do

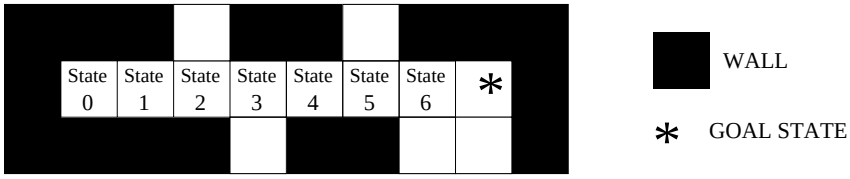


Figure 4.3: Simple 1-D example world to illustrate the problems caused by perceptual aliasing [Whitehead, 1992, pp73–78]

however encode the observation as seen by the Eight Adjacent Squares Agent. For this agent each location appears unique except those labelled State 2 and State 5 which to it appear to be one and the same.

This world is of interest as Whitehead [1992] created it to demonstrate how perceptual aliasing can interfere with Q-learning and claimed that "1-step Q-learning cannot learn the optimal policy for this task."

4.1.1.3 Learning algorithms & action selection

A selection of direct reinforcement learning algorithms were used: Q-learning, SARSA, SARSA(λ) with replacement traces and Watkins's Q(λ) with accumulating traces. For details of the learning algorithms see appendices B, C, D and E respectively, or alternatively Sutton and Barto [1998, p146, p149, p181, p184].

The learning algorithms select actions greedily using their current policy with a probability of $(1-\varepsilon)$, in the remaining ε cases the action executed was selected randomly between all the available actions. In cases of greedy action selection where two or more actions have the same value, ties are broken at random. In both the above cases, (i) selecting exploitative actions or (ii) splitting ties, the action executed was selected randomly with uniform probability across the actions under consideration.

The policies of the above algorithms are updated on-line applying $\pi \leftarrow \arg\max_a Q(o,a)$ after each action-learning time step, where π is the policy and Q is the action value for taking action a given observation o.

The following values were used for the learning algorithms:

- learning rate $\alpha = 0.1$,
- discount rate $\gamma = 0.9$,
- probability of random action ε started at 20% and decayed linearly reaching zero at the $200,000^{th}$ action-learning step. Thereafter it remained at zero.
- A range of values were used for the eligibility trace decay rate λ: 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 0.75 and 0.9.
- The observation-action values for all the learning algorithms were initiated at zero.

The parameter values tested in this thesis were generally selected after conducting a small number of initial runs and observing which values appeared to learn a reasonable number of satisficing policies for the task being considered. In the above case the such a method was used to select values for α, γ and ε whilst the experiment itself explores the range of values for λ.

To provide a baseline for comparison an additional set of runs were also conducted with the agent's actions being selected at random. In this latter case the control algorithm selected with uniform probability between all the available actions for each location.

4.1.1.4 Evaluation criteria

We adopted the same evaluation method as used by Loch and Singh [1998]. After every 1000 learning steps the policy is evaluated greedily to determine the total number of steps required to reach the goal from every possible starting location. The agent is limited to a maximum of 1000 steps to reach the goal from each possible starting location. Thus if a policy is evaluated in a world with N starting positions and fails to reach the goal from all of them it would have a maximum total steps of $N \times 1000$.

Each run consisted of a million action-learning steps (excluding evaluation steps), with evaluation of the current policy occurring every 1000 steps. Each combination of agent, world, learning algorithm and value of λ was repeated 100 times giving 100 samples per data point.

Classification of policies

To obtain an idea of the quality of the policies learnt we devised five policy categories and tracked the number of policies that fell into each category over time. In section 2.2.1 we introduced the concept of *satisficing* policies as being any policy that achieves some minimal aspirational level. In the case of the two grid worlds used in these experiments the minimum aspirational level is that of the agent reaching the goal state from all possible starting locations. This splits the policies learnt into two categories; *satisficing* and *non-satisficing*, but to provide more information we further subdivide the satisficing policies into;

(i) physically optimal (or *optimal* for short),

(ii) better than optimal reactive,

(iii) optimal reactive, and

(iv) other-satisficing.

These policy subcategories are based upon what can be achieved by differing agents. A *physically optimal policy* is the optimal policy that can be learnt by an agent unencumbered by perceptual aliasing, where the function to be optimised is the number of physical actions required in order to reach the goal state from every starting position. We introduce the term *physical actions* to distinguish between actions which cause the agent to change it physical location and *perceptual actions* which change the agent's perception but leave its location unchanged. In the case of these initial experiments only physical actions are available, those actions being the the four movements that it can select, that is, move north, east, south and west. Later experiments will include

perceptual actions. In defining the remaining policy subcategories read optimal as 'physically optimal'.

An agent which suffers from perceptual aliasing is unable to learn the physically optimal policy when such a policy requires the agent to take differing physical actions in two unrelated locations which to it appear perceptually identical. The best that such an agent can achieve is the *optimal reactive policy*. That is the optimal policy which is possible when forming fixed associations between each perceivably different perception and one of the possible actions. In the reinforcement learning literature such policies are also known as optimal memoryless policies. For more details of this concept see Littman [1994].

Agents which suffer from perceptual aliasing can be enhanced through the addition of perceptual actions, internal memory or state estimation. The policies that such enhanced agents can learn will depend on their abilities and the additional information they can glean from the environment. They best they can achieve is the physically optimal policy, and one would hope that they perform better than unenhanced agents, that is, they learn policies which are better than the optimal reactive policy. The subcategory of *better-than-optimal-reactive policies* encompasses the range of possibly policies that are better than the *optimal reactive policy* but worse than the *physically optimal policy*.

The final subcategory of satisficing policies is *other-satisficing policies* which are those policies that reach the goal state from all starting states but use more physical steps than the optimal reactive policy.

Sutton's grid world policy classes

For Sutton's Grid World the minimum total physical steps to reach the goal from all starting positions is 404 physical actions. An example of such a policy is shown in the lower plot in figure 4.2 on page 97. Such policies are *physically optimal policy* for this task.

Littman [1994] showed that *optimal reactive policies* for Sutton's grid world require 416 physical actions. An example optimal reactive policy is shown in the upper plot in figure 4.2.

These two "optimal" policies set the levels for the two remaining subcategories of satisficing policies, that is better-than-optimal-reactive (BTOR) policies and other-satisficing policies. The final category are policies which fail to reach the goal and are thus non-satisficing. Examples of better-than-optimal-reactive and other-satisficing policies can be seen in figure 4.10 on page 122; note that both these are in the context of an active perception agent. An example of a non-satisficing policy for Sutton's Grid World can be seen in figure 5.19 on page 197. All five categories are summarised in table 4.1.

Goal Reached From All Starting Locations	Total Physical Actions	Policy Category
yes	404	Physically Optimal
yes	$405-415$	Better Than Optimal Reactive (BTOR)
yes	416	Optimal Reactive
yes	>416	Other-Satisficing
no	-	Non-Satisficing

Table 4.1: Policy categories for Sutton's Grid World

Note that of necessity our measure of satisficing is stricter than the simple requirement that the agent reaches the goal from all possible starting locations. To ensure that policy evaluations terminate we limited the agent to 1,000 actions from each starting position. Thus, any policy that fails to reach the goal from any start location in under 1,000 steps is classed as non-satisficing irrespective of the total steps for that policy.

Simple 1-D example policy classes

Due to the very simple nature of the 1-D example world only three of the five policy classes identified above can exist. This can be seen by examining the two example

policies shown in figure 4.4. If the agent's policy has a fixed association between any of the perceptual states and the action move west then the policy becomes immediately non-satisficing. The physically optimal policy for this problem is that which associates the action move east with all states. This policy also corresponds to the optimal reactive policy, therefore the two policy classes optimal reactive policy and better-than-optimal-reactive are subsumed by the physically optimal policy. It is also possible to see from figure 4.4 that the physically optimal policy takes a total of 28 physical action steps to reach the goal from all start states.

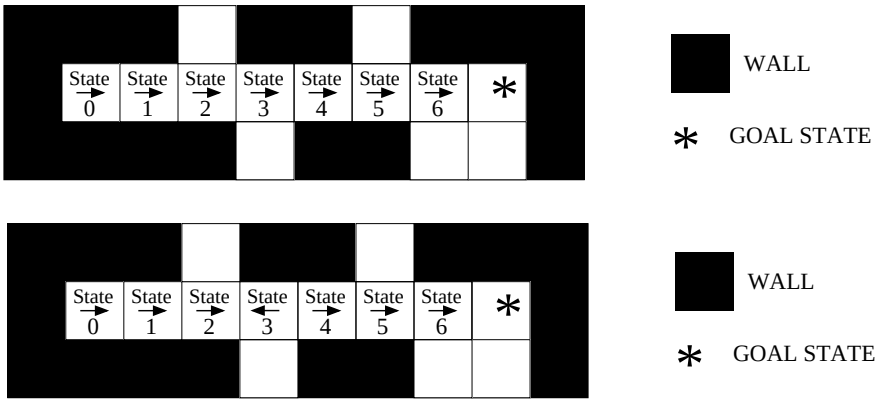


Figure 4.4: Simple 1-D Example, example policies. The top policy is the optimal policy for this problem, that is, move east in all location. The lower policy is non-satisficing as the agent is unable to reach the goal from states 0, 1, 2, or 3, instead it ends up shuttling ad infinitum between states 2 and 3

Although at first glance there appears to be only one possible satisficing policy, that is, move east for all observations, policies which take more than 28 physical action steps to reach the goal could exist. Where policy actions have the same action value ties are broken at random, thus it is possible to imagine a policy where in one or more locations the agent has no preference between moving east or west. An agent following this policy performs a limited random walk before ultimately reaching the goal. Such a policy, if it reached the goal in less than 1,000 steps from each starting location, would still be satisficing. Therefore we include the category of other-satisficing policies. The three categories are summarised in table 4.2.

Goal Reached From All Starting Locations	Total Physical Actions	Policy Category
yes	28	Optimal
yes	> 28	Other-Satisficing
no	-	Non-Satisficing

Table 4.2: Policy categories for Simple 1-D Example

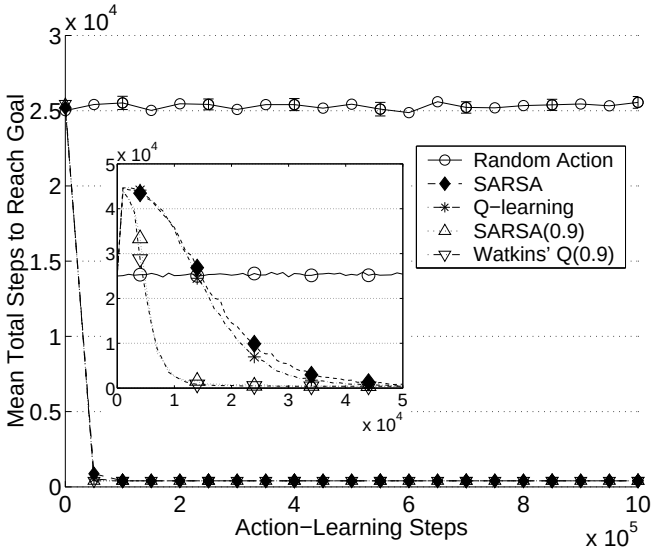
4.1.2 Results

4.1.2.1 Sutton's grid world

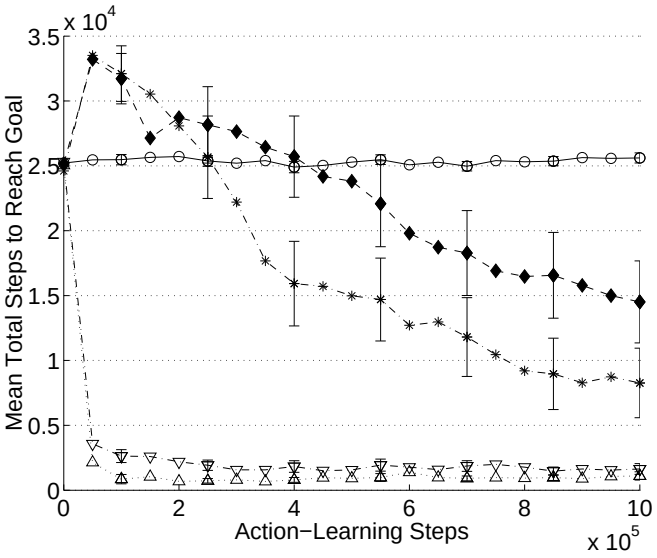
Figure 4.5 shows the mean total steps for all four learning algorithms. The top plot shows results with the Absolute Position Agent which experiences no perceptual aliasing. All four learning algorithms quickly converge on the optimal solution in around fifty thousand action-learning steps. This indicates that all four learning algorithms have no problem in learning this task if there is no perceptual aliasing.

The lower plot shows results for the Eight Adjacent Squares Agent which aliases multiple locations in the world. The mean total steps for SARSA(λ) and Watkins's Q(λ) with $\lambda = 0.9$ rapidly approach the optimal reactive solution, with SARSA(λ) reaching convergence in less than one hundred thousand action-learning steps, and Watkins's Q(λ) in around three hundred thousand action-learning steps. The other values of λ tried (which are not shown) converged to a similar number of mean total steps as $\lambda = 0.9$ but, as would be expected, lower values take longer to converge, with the lowest value, $\lambda = 0.001$, converging after eight hundred thousand action-learning steps.

The mean total steps of the policies learnt by the 1-step backup algorithms, Q-learning and SARSA, appear to be gradually converging towards a level where the majority of policies will be satisficing. This convergence is, however, extremely slow compared to SARSA(λ) and Watkins's Q(λ). In addition the 95% confidence intervals indicate there is a significant variation in the policies learnt by Q-learning and SARSA.



(a) Absolute Position Agent



(b) Eight Adjacent Squares Agent

Figure 4.5: Mean policies' path lengths versus action-learning steps for Sutton's Grid World. Bars indicate 95% confidence intervals (where not shown confidence interval is too small to be visible). To simplify plots data points are only plotted at 50,000 action-learning step intervals and these points are then connected with straight lines. The resulting plots allow the general trend to be clearly observed, but loses detail of the "noisy" variation that occurs between plotted data points. Insert in top figure shows enlargement of the first 50,000 steps which would otherwise not be visible (data plotted for every 1,000 steps).

We classified the policies which have been learnt according to the criteria summarised in table 4.1 and plotted the number of policies that fell into each category as vertically stacked bars, with bars being plotted at regular intervals of action-learning steps. The resulting four plots for each of the learning algorithms and parameter combinations are shown in figure 4.6. Each combination of parameters and learning algorithm was repeated 100 times. The height of the shaded areas on these plots indicate the number of policies falling into each policy category as measured after a particular number of action-learning steps. The use of vertical stacked bar charts allows easy visualisation of the change in the numbers of each policy class as learning progesses. Examining the top left hand plot, which shows the policies learnt using SARSA, initially all one hundred policies are non-satisficing (grey shading). At around one hundred thousand action-learning steps a small number of policies become satisficing, but their total physical steps exceeds 416 so they are classified as other-satisficing policies (black shading). The number of policies classified as other-satisficing gradually increases until after one million action learning steps 64% of the policies are other-satisficing and 36% are non-satisficing. The results for Q-learning (top right) are similar with final tallies of 81% other-satisficing policies and 19% non-satisficing. Neither learnt any policies that were better than other-satisficing at any stage. However, the other-satisficing policies found were reasonable with a mean total physical steps of 457 for SARSA and 487 for Q-learning.

By comparison, SARSA(λ) and Watkins's Q(λ) had, in less than six thousand action-learning steps, learnt a small number of policies that were classified as optimal reactive (white shaded areas in the lower two plots of figure 4.6). By the end of a million action-learning steps the distribution of policies for SARSA(λ) were: 80% optimal reactive; 0 other-satisficing, and 20% non-satisficing. Similarly, after a million action-learning steps, the distribution of policies for Watkins's Q(λ) was: 48% optimal reactive; 22% other-satisficing, and 30% non-satisficing

There existed the possibility for all of the learning algorithms that although a given proportion of the population of policies were continually categorised as satisficing, individual policies were not stable, switching back and forth between satisficing and non-satisficing solutions. With this possibility in mind we examined the stability of the

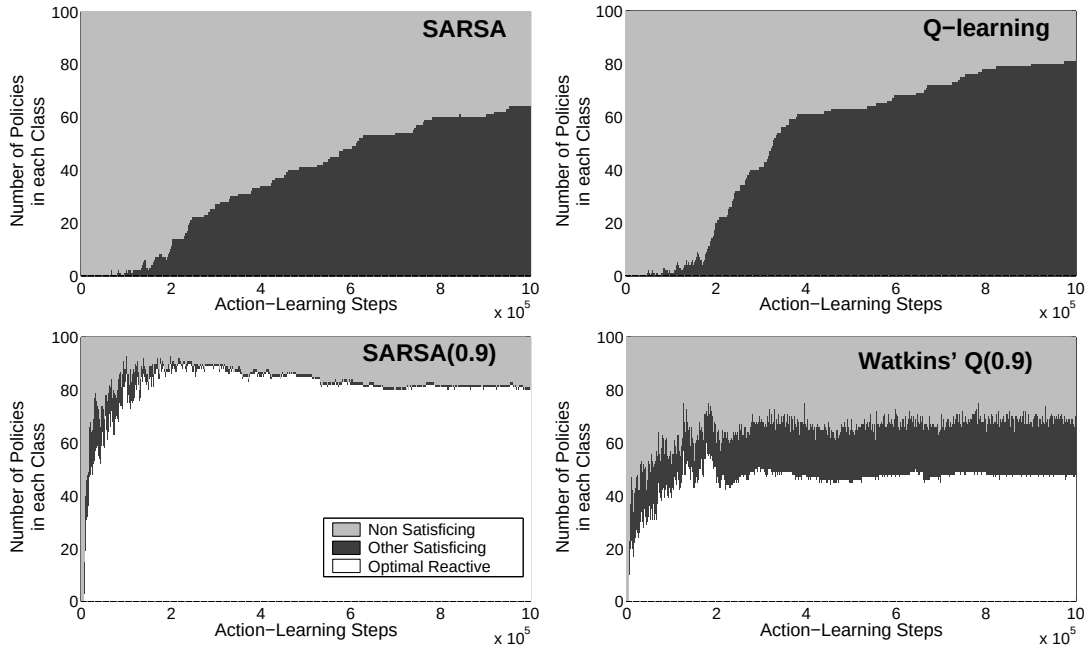


Figure 4.6: Categorisation of policies versus action-learning steps for the four learning algorithms when learning to solve Sutton's Grid World using the Eight Adjacent Squares Agent. Plots indicate how the numbers in each policy classification change as learning progresses. Plots are vertical stacked bar graphs with the height of the different shaded areas indicating the number of policies that fell into each classification at the point when the policies were evaluated. The policies were evaluated every thousand action-learning steps. The policy categorisations used are set out in section 4.1.1.4 and summarised in table 4.1

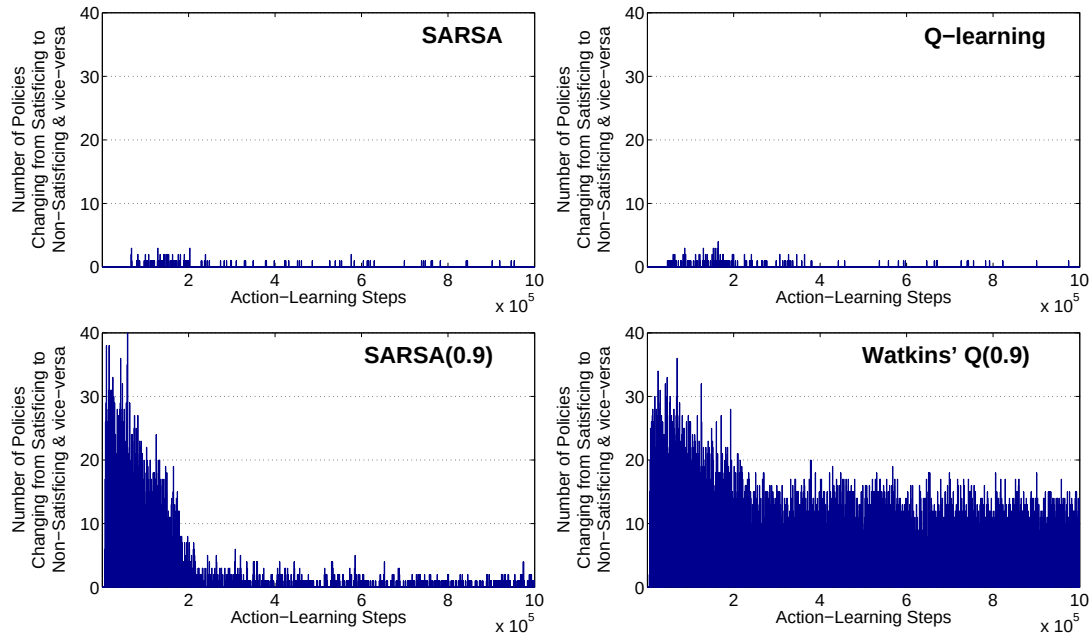


Figure 4.7: Stability of policy classification versus action-learning steps for the four learning algorithms. Plots are simple bar graphs with the height of the bars indicate the number of policies that changed from being satisficing to non-satisficing and vice-versa since the previous policy evaluation. All are learning to solve Sutton's grid world using the Eight Adjacent Squares Agent

policies learnt by the four learning algorithms

Plots for stability are shown in figure 4.7. These are derived by counting the number of individual policies that change classification between consecutive policy evaluations. A change in classification is counted when a policy changes from being non-satisficing to any satisficing classifications, or vice-versa. From figure 4.7 both SARSA and Q-learning are very stable with no more than three policies changing classification at any one time. Much larger changes in classification are seen initially for SARSA(λ) and Watkins's Q(λ), as we would expect. The number of changes then steadies at a fairly low level for SARSA(λ), but remains relatively high for Watkins's Q(λ).

A related observation is that Watkins's Q(λ) generates a much larger number of other-satisficing policies than SARSA(λ). SARSA(λ) learns either optimal reactive or non-satisficing policies, and virtually zero other-satisficing policies. An investigation revealed that the optimal reactive policies generated by Watkins's Q(λ) are reasonably stable, that is, the number of changes between optimal reactive policies and any other classification are comparable to the figures for SARSA(λ). However, a large number of policies, on average 11.6%, flip between other-satisficing and any other classification. This accounts for most of the changes reported on the Watkins's Q(λ) plot in figure 4.7. It thus appears that the policy update rule used by Watkins's Q(λ) learns a significant proportion of unstable, non optimal reactive policies.

4.1.2.2 Simple 1-D example

Results for the simple 1-D example world are shown in figure 4.8. The top two plots show mean total steps with 95% confidence intervals for the four learning algorithms and also for random action selection. The left hand plot shows results for the Absolute Position Agent which does not experience any perceptual aliasing. In the absence of any aliasing all four learning algorithms learn the optimal solution to this world.

The Eight Adjacent Squares Agent (top right hand plot) aliases State 2 and State 5 (figure 4.3) in this simple 1-D world. Using this agent both SARSA and Q-learning

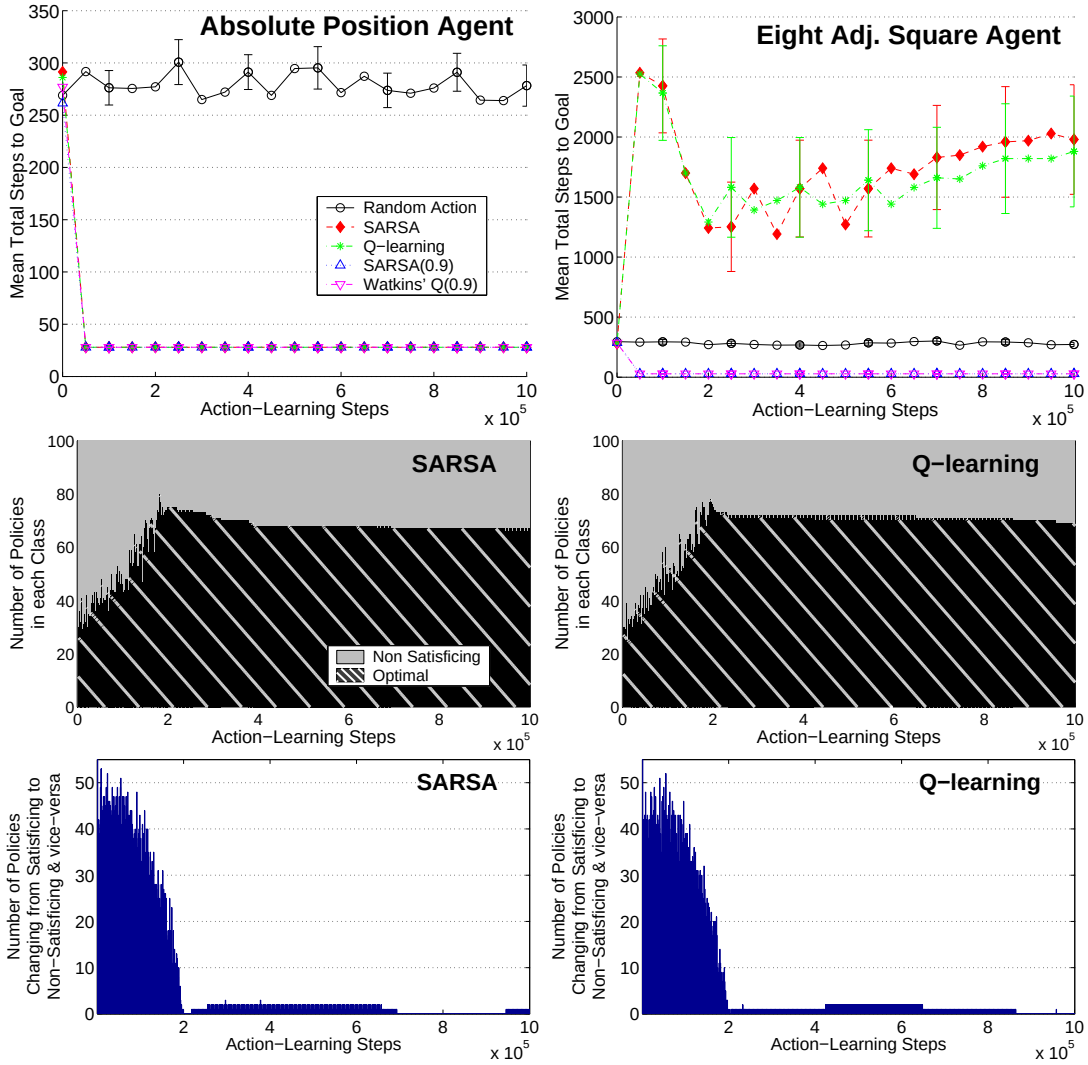


Figure 4.8: Plots for the Simple 1-D Example grid world. Top plots are mean total steps found when policies were evaluated versus action-learning steps. To simplify plots, data points are only shown for every 50,000 action-learning steps. Left hand graph shows results for Absolute Position Agent which suffers no perceptual aliasing. Right hand plot shows results for Eight Adjacent Squares Agent which aliases locations labelled State 2 and State 5. Middle plots show categorisation of policies versus action-learning steps for SARSA and Q-learning. Bottom plots show stability of policy classification versus action-learning steps for SARSA and Q-learning. Height of bars indicate the number of policies that changed from being optimal to non-satisficing and vice-versa since the previous policy evaluation. All plots with the exception of top left are for the Eight Adjacent Squares Agent

perform worse than an agent selecting actions at random, though the large confidence intervals suggest that it is worth investigating what is occurring with the individual policies. SARSA(λ) and Watkins's Q(λ), $\lambda = 0.9$, both learn the optimal solution in less than 50,000 action-learning steps.

The policies learnt were classified in accordance with the policy categories as summarised in table 4.2 and a vertical stacked bar chart of the numbers falling into each category was plotted against the number of action-learning steps that had past. The resulting plots for SARSA and Q-learning are the middle two plots of figure 4.8. Of the three policy categories defined, 'other-satisficing' policies never occurred and thus only two categories are shown on the plots.

We see from the middle two plots of figure 4.8 that both SARSA and Q-learning reach a plateau with just over 65% of the policies learnt being classified as optimal after just 300,000 action-learning steps. For these two learning algorithms we again plot the change in classification of the policies to test that the policies are stable. The two lower plots in figure 4.8 suggest that the optimal policies learnt are indeed stable.

Plots of the categorisation of policies learn by SARSA(λ) and Watkins's Q(λ) are not shown as 100% of the policies were classified as optimal after 1000 action-learning steps and there is very little variation from this initial level for the remainder of the one million action-learning steps.

4.1.3 Discussion

The results successfully replicate those of Loch and Singh [1998] demonstrating that SARSA(λ) can find optimal reactive solutions to tasks containing perceptual aliasing. In fact this result generalises to Watkins's Q(λ) suggesting that any method that uses eligibility traces can find optimal reactive solutions. These results therefore support their hypothesis, that reinforcement algorithms that use eligibility traces can learn optimal reactive policies. However, a question can be raised as to the stability of the solutions given that around 10% of the policies learnt by Watkins's Q(λ) appear to be flipping between satisficing and non-satisficing, see figure 4.7 on page 109.

The improvement in performance which occurs as λ increases ties in with results presented by Pendrith and McGarity [1998]. Their paper shows that the optimal reactive policy will be an equilibrium point for reinforcement learning algorithms that use undiscounted Monte Carlo returns when determining the value of observations. Temporal difference methods with eligibility traces, of which SARSA(λ) and Watkins's Q(λ) are examples, calculate returns which get steadily closer to those produced by Monte Carlo methods the closer λ is to 1 [Sutton and Barto, 1998, chp.7].

The surprise result is that SARSA and Q-learning could learn satisficing policies to Sutton's Grid World, and optimal policies for the simple 1-D example. The latter is even more remarkable as Whitehead [1992] presented the example in order to illustrate the extent to which perceptual aliasing can interfere with Q-learning and claimed that "1-step Q-learning cannot learn the optimal policy for this task" (p.73). In both instances the policies learnt appear to be stable.

Further examination of this issue indicates that exploration is important in determining whether reinforcement learning algorithms which use 1-step backup, such as SARSA and Q-learning, can learn policies that are both stable and satisficing in partially observable environments. In the experiments presented above the probability of selecting an exploratory action (ε) starts at 20% but reaches zero after two hundred thousand action-learning steps. For the remaining eight hundred thousand steps the agent always follows the current policy without trying any exploratory actions. The lack of exploration appears to avoid the destructive effects of global impairment[2] allowing policies to achieve stable solutions. The effect of exploration is nicely illustrated by figure 4.9 which shows the categorisation of policies learnt for the Simple 1-D Example world and Eight Adjacent Squares Agent, using Q-learning with ε fixed at 0.01. With a fixed value for ε the policies are not stable, and continuous oscillations are seen in the number of optimal policies. This is in contrast to the plateau seen in figure 4.8. A secondary point of note is that in figure 4.8 the number of optimal policies ramps up slowly as ε decreases from 0.2 to zero. This contrasts with figure 4.9 where, with a fixed, but

[2]Global impairment is a term coined by Whitehead [1992]. Given the bucket-brigade update employed by 1-step backup reinforcement learning algorithms, inaccurate estimates of observation values that occur with perceptually aliased underlying states can lead to errors in the observation values of non aliased states.

initially lower, value of ε, the number of optimal policies learnt increases more rapidly. The observed oscillations reinforce Whitehead [1992, p.78]'s argument that Q-learning (or any 1-step backup algorithm) is unable to converge on stable solutions in partially observable environments, provided there is some possibility of selecting an exploratory action. However, once exploration has ceased, it is possible for 1-step backup algorithms to converge on satisficing policies. In light of the this discussion the hypothesis that arose from Whitehead [1992], which we phrased as"1-step reinforcement learning algorithms are *not* able to learn policies which are both stable and optimal, when the task involves perceptually aliased states" at the start of this section, now needs to be modified to reflect the importance of selecting exploratory actions. We return to the issue of the effect of exploration on learning in partially observable environments in chapter 5.

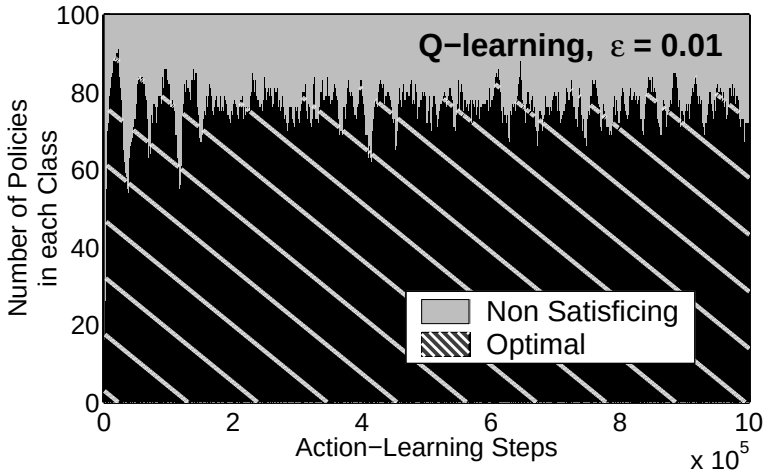


Figure 4.9: Categorisation of policies for Q-learning with fixed $\varepsilon = 0.01$ versus action-learning steps. Task consists of the Simple 1-D Example world combined with the Eight Adjacent Squares Agent

4.1.4 Summary

The main aim of the experiments presented above is to illuminate the problems that occur when applying reinforcement learning to partially observable environments. We

are interested in doing this in order to clear the ground before moving on to look at whether active perception can be used to address these issues. These results are, however, of interest in their own right as reinforcement learning is often used in robotics, and real limited sensor arrays certainly create the possibility of perceptual aliasing. An important observation, therefore, is that if there exists the possibility of state aliasing, then it either needs to be designed out of the task, or careful selection should be made of the learning algorithm. For example, it is probably worth avoiding reinforcement learning algorithms that do 1-step backup. In fact any reinforcement learning algorithm that uses truncated returns will be subject to some detrimental effects of global impairment [Whitehead, 1992, p.80]. However, as demonstrated by the above results, reinforcement learning algorithms that use eligibility traces can quickly learn reasonable solutions.

With regard to the overall aims of this thesis, having replicated work showing that learning reactive policies is possible where a one-to-many mapping exists between the state of the world and the learning algorithm's perceptions, we now proceed to consider learning when a many-to-many state to observation mapping exists, and where that mapping is to some extent under the control of the learning algorithm.

4.2 Agent with Access To An Oracle

We now move from showing that reinforcement learning is capable of learning reactive policies in the presence of perceptual aliasing, to demonstrating that is possible for reinforcement learning to learn reactive policies when observations are filtered via an active perception system. What follows, unlike that presented above, is not a replication of previous work but our own original work.

In order to demonstrate learning with active perception we need to consider what we mean by an active perception system, that is to say, we need to clarify the active perception systems design. One of the simplest designs that can be envisaged is that modelled on a camera. In this model an agent obtains an observation based upon where the camera is looking, executing an action that changes the direction the camera is pointing,

that is changing the view point, changes the observation received. In this case the observations received, though hopefully indicative of the action to be taken, have no explicit meaning and are just tokens against which actions are associated. It is plausible that such an approach may not be sufficient, so we also consider an active perception system that provides the agent with a greater level of information, specifically it provides instruction on the next action that the agent should execute. Comparison of the two should provide an indication of the design choice that needs to be made.

Consider first the case of the simplest active perception design envisaged above, that of a simple "camera" model. To demonstrate that reinforcement learning can learn satisficing reactive policies for such an active perception system, we need to extend the results from section 4.1. That section considered problems where multiple states mapped to the same observation. We now need to show that learning can occur when there exists the possibility of obtaining multiple observations from each state and where:

- each observation is not necessarily unique to each state, and
- the observation received by the agent is determined both by the state and by the actions it selects.

For a satisficing reactive policy to exist under such a many-to-many mapping there has to exist a sequence of actions which allows a unique representation to be observed for each state in which a necessarily different action has to be selected. In addition, the sequence of actions by which the agent arrives at each of the unique observations are themselves driven reactively based upon the observations received. In other words we require (i) the existence of at least one deterministic sequence of observations and actions which will achieve the given goal from all starts states, and (ii) that reinforcement learning can learn a policy which executes such a sequence.

For an simple active perception system that returns an arbitrary set of observations for each state it is not easy to show that point (i) above holds. At this stage, however, we are interested in demonstrating that point (ii) is possible. Thus to ensure a valid test we introduce a "perfect" simple active perception system that on the selection of a

perceptual action returns a unique observation for each state of the environment. This active perception system which can perfectly disambiguate the agent's current state we refer to as a *State Oracle*.

The second action perception design that we consider, the one which provides a greater amount of information, we refer to as an *Action Oracle*. The Action Oracle, on receipt of request from the agent, directs it based on a known optimal solution (the solution used is that indicated by the arrows in the lower plot of figure 4.2). This type of oracle was in part suggested by the coupling that we see between state disambiguation and movement actions for the other type of agent which, like active perception, represent a modifiable approach, that is internal memory agents [Littman, 1994; Peshkin et al., 1999; Lanzi, 2000]. In these agent's the actions that modify the internal memory bits also execute some external physical action. Examining the policies of internal memory agents we see that this coupling is necessary as the agents have no other mechanism for remembering their actions. In the case of active perception agents the situation changes with the agents relying on external cues, thus we believe that such a coupling should not be required, but it is useful to confirm this supposition.

Between them, the two oracles can be characterised as answering two possible questions that the agent could ask. That of "where am I?" and "where should I go?".

To summarise, the hypotheses that we aim to test in this section are:

(a) Reinforcement learning can, in spite of the presence of a many-to-many state to observation mapping as described above, learn deterministic policies which will achieve the given goal from all starts states (given that such a policy exists).

(b) That a simple active perception system which does no more than provide alternative tokens for each state, provides benefits in terms of the quality of satisficing policies learnt.

(c) That the improvement in policy quality achieved through providing more explicit information to the agent is relatively small. (Especially when the extra complexity of obtaining the information provided is considered.)

4.2.1 Experimental setup

4.2.1.1 Sutton's grid world

The experiments presented here use Sutton's Grid World (figure 4.2) as described previously in section 4.1.1.2.

4.2.1.2 Agents

We used three types of agent:

(i) *Eight Adjacent Squares Agent.* An agent whose observations represent whether or not the eight squares adjacent to its current location are occupied by an obstacle, as shown in figure 4.1 on page 92.

(ii) *State Oracle Agent.* An agent who normally perceives the same observations as the Eight Adjacent Squares Agent, but who can ask the State Oracle where it is. On asking the oracle the agent receives an observation that corresponds to its absolute location in the world given in Cartesian coordinates.

(iii) *Action Oracle Agent.* An agent who receives the same observations as the Eight Adjacent Squares Agent but has an additional action allowing it to ask the Action Oracle in which direction it should go. The agent then immediately executes the action specified by the oracle.

The latter two agents receive a reward of -1 for selecting the action to ask their respective oracles a question. This reward is in addition to the reward for executing any subsequent actions. The way this works in practice is that the Action Oracle Agent, on asking its oracle where to go, transitions to an adjacent location in the grid world based on the optimal action from its current location, and receives a reward of -2 (-1 for asking the oracle and -1 for the action executed). The State Oracle Agent, on asking

its oracle where it is, sees the observation it receives transition from its normal representation of its current location (for example, $\langle 148 \rangle$) to an absolute coordinate system representation of the same location (for example, $\langle 2,4 \rangle$), receiving a penalty of -1 for asking the oracle. It then has to learn the optimal action to execute based on this new observation. Once it selects a physical action, the observations it receives revert back to normal, that is, formed by observing the eight squares adjacent to its new location. This dual representation of the State Oracle Agent's location in the world more than doubles the observation space which it needs to explore.

The cost of asking a question of an oracle was deliberately chosen to match the cost of physical actions to ensure that the most valuable policies would only be those that reached the goal state. Given future discounting of rewards there exists a point where the discounted cost of an infinite series of low cost actions is less than the cost of a finite series of more costly actions which reach the goal state. In other words, if it costs less to talk to the oracle than move, then at a certain level of discount and certain distance from the goal it becomes more attractive to sit around and request information ad infinitum than actually walk to the goal state. Note that when using the State Oracle Agent there is no compulsion to move following a request to the oracle. Matching the cost of physical and perceptual actions avoids this problem. We discuss how real perceptual cost could be reflected in section 9.5.

4.2.1.3 Learning algorithms

A range of reinforcement learning algorithms were used: Q-learning, SARSA, SARSA(λ) with replacement traces, Watkins's Q(λ) with accumulating traces. For details of the learning algorithms see see appendices B, C, D and E All of the learning algorithms continuously updated their policies. Actions are selected greedily using the current policy with a probability of $(1-\varepsilon)$. In cases where actions have the same value, ties were broken at random. In the remaining ε cases the action executed was selected randomly between all the available actions. In all cases above, random selection is uniform across all possibilities. Observation-action values for all the learning algorithms were initiated at zero.

The following values were used for all learning algorithms: (i) learning rates (α) of 0.1 and 0.01, (ii) discount rate $\gamma = 0.9$, (iii) probability of random action (ε) started at 20% and decayed linearly reaching zero at the $200,000^{th}$ action-learning step, thereafter it remained at zero. For the learning algorithms SARSA(λ) and Watkins's Q(λ) a range of values were tried for the eligibility trace decay rate (λ); from 0.01 to 0.9. Results are shown for $\lambda = 0.9$ which, out of the range tested, was found to generally perform the best. Given that the best results for λ were found at the top of the value range tested we cannot say that better results would not be found for values greater than 0.9. However, based on experiments presented in chapters 7 and 8 we can reasonably suppose that SARSA(λ) would perform poorly at the limiting upper value of $\lambda = 1.0$, although this may not apply to Watkins's Q(λ). It may prove difficult to find a fully justifiable upper limit for the range of λ tested as we believe performance may continue to improve as $\lambda \rightarrow 1.0$ provided $\lambda < 1.0$, though the improvements we suspect would become more marginal the closer λ gets to 1.0.

4.2.1.4 Evaluation criteria

To examine hypothesis (a) we ran the agents for 400,000 action-learning steps and looked at a selection of the final policies which had been learnt. For the two type of oracle we observe whether satisficing policies have been learnt. As a qualitative evaluation of hypothesis (b) we then examine the use made of oracles in these policies. We consider whether or not the policies make sensible use of the additional action available to it, that of consulting the oracles.

To more rigorously evaluate hypothesis (b) and to evaluate hypothesis (c), we again run the agents for 400,000 action-learning steps, this time evaluating the current policy after every 1000 action-learning steps. We consider whether the policies learnt when the oracles are available are better than the policies learnt in the absence of of oracles. We also compare results between the two type of oracles.

The measure used in comparing policies is the total number of physical actions taken to reach the goal state from every possible start state. This is determined by executing

a policy greedily from every possible starting position. Separate counts were kept of physical actions and requests to the oracle. The agent is limited to a maximum of 1000 steps (total of both physical actions and requests to oracles) to reach the goal from each starting position. There are 46 non-goal starting locations in Sutton's Grid World, so a policy that fails to reach the goal from all of them would have a maximum total number of steps of $46,000$. We compare policies base on the number of physical actions (excluding requests to oracles) in order to allow a fair comparison between agents with oracles and those without.

To allow observation of the types of policies learnt we classify the policies using the five categories outlined in section 4.1.1.4 that is, optimal, better-than-optimal-reactive (BTOR), optimal-reactive, other-satisficing and non-satisficing. The classifications are summarised in table 4.1 on page 103. The number falling into each classification is plotted against the number of action learning steps that have passed. This gives a idea of the development of policies as learning progresses. Note that in classifying policies as non-satisficing, requests to oracles are counted towards the limit of 1000 actions. Without including these in the count it would be possible for an agent to select no physical actions and spend until the end of eternity just talking to the oracle.

Each combination of agent, world, learning algorithm and values of α and λ was repeated 100 times giving 100 samples per data point.

4.2.2 Results

4.2.2.1 First and second section hypothesis

Figure 4.10 shows examples of the policies learnt for both Action and State Oracle Agents, using SARSA, Q-learning and SARSA(λ). Grid squares containing filled blocks indicate where the policy is to consult the oracle. All the examples shown are satisficing policies and thus demonstrating hypothesis (a), that is, given that a deterministic policy exists, reinforcement learning can learn it in spite of a many-to-many mapping between observations and states.

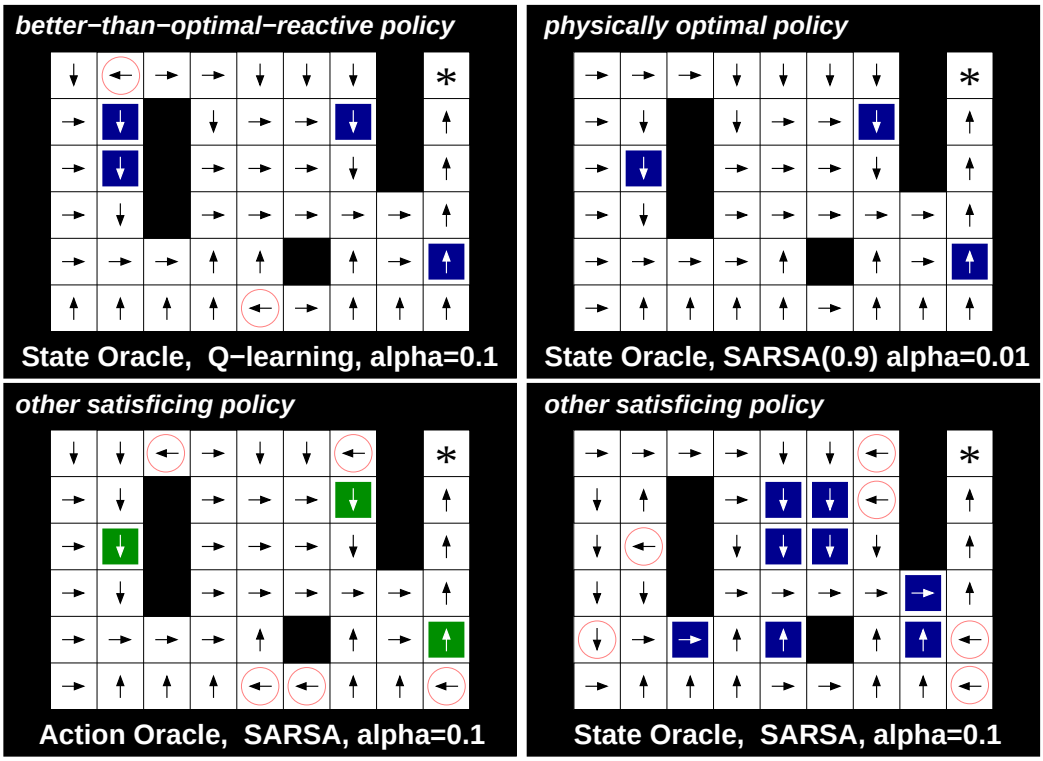


Figure 4.10: Example policies learnt after 400,000 action-learning steps. Arrows indicate the physical action selected by the policy. Squares containing filled blocks indicate where the policy is to consult the oracle in order to select the shown physical action. Non-optimal physical actions are indicated with a faint circle around the arrow. Of the four policies shown only the top right policy is *physically optimal*, taking a total of 404 physical action steps to reach the goal from all possible starting locations. The second best policy, in terms of physical action steps, is the top left policy which takes a total of 412 physical action steps. Using the classification scheme introduced in section 4.1.1.4 this policy is classified as *better-than-optimal-reactive* (BTOR). The bottom left policy takes 422 physical action steps and bottom right takes 424. Both these are classified as *other satisficing*

We also observe that the policies learnt do appear to make sensible use of the additional action available, whether that action be to consult the State or Action Oracles. Of the examples shown three have learnt that they can achieve a better solution if they consult their respective oracle when they are in the three squares labelled 148 in figure 4.2. As was discussed in section 4.1.1.2, the squares labelled 148 are perceived to be identical but the physical action required for an optimal solution is not the same in each location. In the fourth example in figure 4.10 the agent consults the State Oracle when in the squares labelled 0, 2 and 16 in figure 4.2. The grid squares labelled 0 require different actions to be executed in different locations. The squares labelled 2 are aliased but require the same action to be executed in both locations. Square 16 is not aliased. Of the examples shown, the top right (State Oracle, SARSA(λ) with $\lambda = 0.9$ and $\alpha = 0.01$) is an optimal solution in terms of physical actions.

The top two policies in figure 4.10 take less physical actions to reach the goal state than the optimal-reactive-policy, that is less than 416 physical steps. The top right policy taking 404 physical steps which is equal to the optimal policy for Sutton's Grid World. As these two policies are learnt using the State Oracle this provides evidence in support of hypothesis (b). In the following section (section 4.2.2.2) we quantify this result by considering how many agents actually utilised the oracles to learn better policies.

4.2.2.2 Second and third section hypothesis

Figure 4.11 allows comparison of the mean total steps (sum of physical actions and requests to the oracles) of the three agents; two with access to oracles and one without. Plots are shown for each of the learning algorithms with the left most plots showing results for $\alpha = 0.01$ and the right most plots showing results for $\alpha = 0.1$.

For $\alpha = 0.01$ the two 1-step backup algorithms (SARSA and Q-learning) fail to learn reasonable policies using any of the agents. The mean total steps in these six cases remains just below the maximum of 46,000 steps. Results for SARSA and Q-learning are better when $\alpha = 0.1$, with the mean total steps for all three agents falling as learning progresses. Even so, the Eight Adjacent Squares Agent struggles to learn reasonable

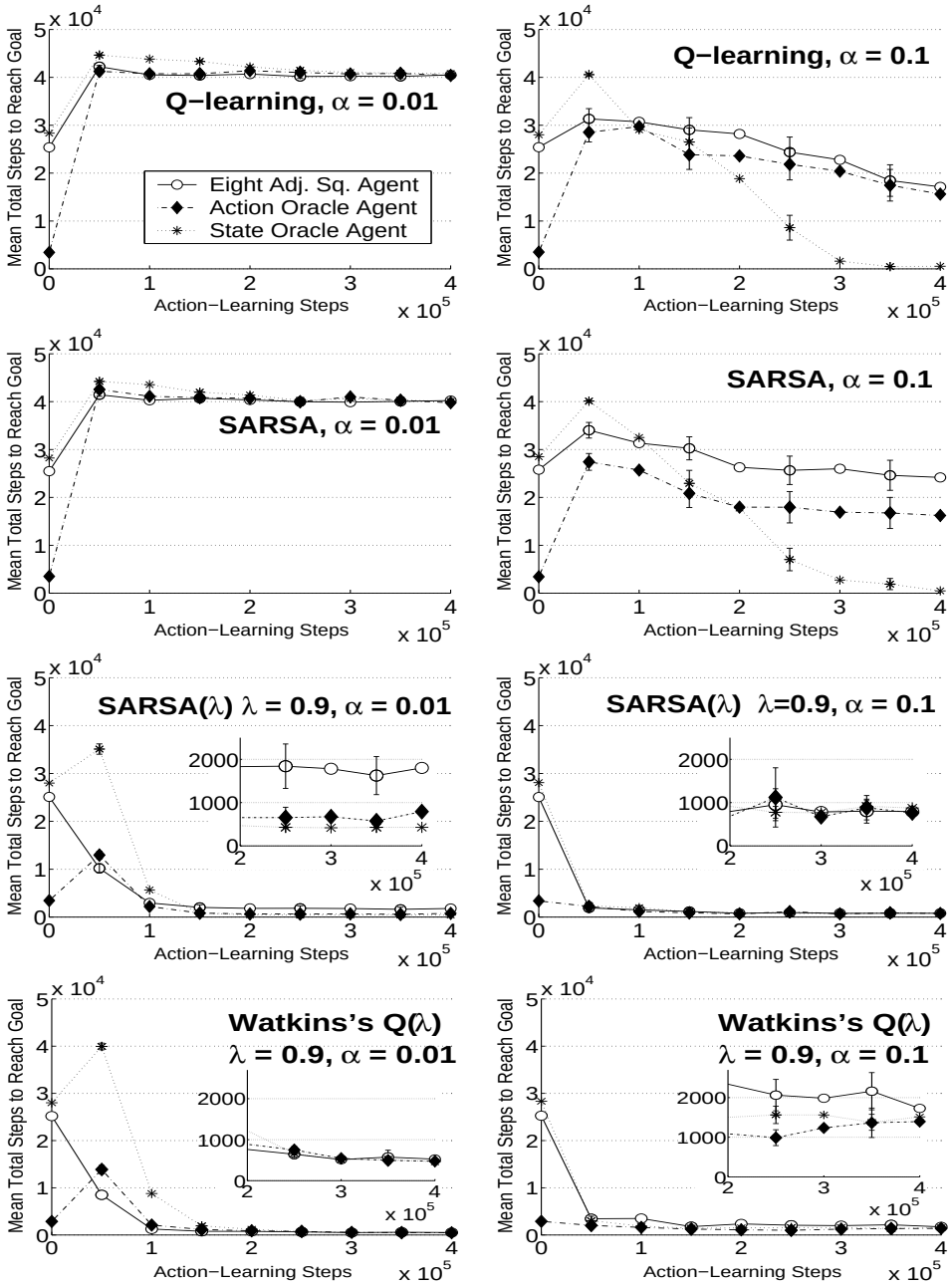


Figure 4.11: Plot of mean total steps (sum of physical actions and requests to oracles) found when policies were evaluated, versus action-learning steps. To simplify plots data points are only plotted at 50,000 action-learning step intervals and these points are then connected with straight lines. Bars indicate 95% confidence intervals. Inserts on the latter four plots show enlargements of their tails

solutions. This is expected as 1-step backup of observation-values will cause global impairment of the learnt policies. The curve for the Action Oracle Agent is better in the case of SARSA but almost identical to the Eight Adjacent Squares Agent for Q-learning. The most significant difference is shown by the State Oracle Agent. It is slower to learn initially, due to the increase in the observation space caused by the State Oracle (see section 4.2.1.2), but in the longer term it achieves much better results than either of the other two agents.

In the two plots for SARSA(λ), the total mean steps of all three agents reduce rapidly, convergence occurring more quickly with $\alpha = 0.1$ than $\alpha = 0.01$, as would be expected. Inserts, which show the tail of both of these plots, indicate that for $\alpha = 0.1$ there is no distinction between the mean total steps for the three types of agent. However, for $\alpha = 0.01$ the average policy learnt by the eight adjacent squares agent is worse than that for the two agents which can use oracles. Result for Watkins's Q(λ) are similar to SARSA(λ) with the exception that the mean total steps of the final policies for all three agents are better for $\alpha = 0.01$ than those for $\alpha = 0.1$.

To get a clearer picture of what is occurring for each agent and learning algorithm combination we plot the categorisation of policies against the number of steps the agent has spent learning, see figures 4.12 and 4.13. The plots are vertical stacked bar graphs with the different shadings indicating the number of policies that fell into that classification when the evaluation was run. The classification of policies used is that presented in section 4.1.1.4 and summarised in table 4.1. The policies are evaluated every thousand action-learning steps, thus changes in the number of each policy class can been seen as learning progresses. The two figures show the categorisation of policies from 100 trials over the course of 400,000 action-learning steps.

Figure 4.12 shows the policies learnt by the State Oracle Agent using the two 1-step-backup reinforcement learning algorithms, Q-learning and SARSA, with learning rate $\alpha = 0.1$. Only results for the State Oracle Agent and $\alpha = 0.1$ are shown as very few satisficing policies were learnt for the other two types of agent and for the lower learning rate of $\alpha = 0.01$. By the end of the trials the majority of policies learnt by Q-learning (left hand plot) are classified as other-satisficing (72 policies), with 4 that are optimal

reactive, 20 that are better-than-optimal-reactive and 4 non-satisficing. No physically optimal policies have been learnt. These results are better than expected as the State Oracle only addresses the issue of local action selection, not that of global impairment. The success of this agent with 1-step backup learning algorithms is, however, heavily dependent on the value of α as no satisficing policies are learnt for $\alpha = 0.01$. The categorisation of policies learnt by the State Oracle Agent using SARSA (right hand plot) is very similar with final results of 79 other-satisficing, 6 optimal reactive, 14 better than optimal reactive, and 1 non-satisficing.

Figure 4.13 shows the categorisation of policies for SARSA(λ) for each of the three agents and both learning rates tried. The left hand plots are for $\alpha = 0.01$ and the right hand plots for $\alpha = 0.1$. For the Eight Adjacent Squares Agent, which has no access to an oracle, with $\alpha = 0.01$ the majority of solutions are optimal reactive, 68 at the end of 100 trials, with 11 other-satisficing and 21 non-satisficing policies. The results are similar for this agent with $\alpha = 0.1$; 83 optimal reactive, 4 other-satisficing and 13 non-satisficing policies.

With $\alpha = 0.1$ the Action Oracle Agent initially generates a small number of physically optimal policies, however these quickly disappear indicating that they are not stable and by the end the majority of policies (65) are optimal reactive, 19 are better than optimal reactive, 1 other-satisficing, and 15 non-satisficing. The State Oracle Agent learns a large number (73) of better than optimal reactive policies, 7 optimal reactive, 4 optimal, and 1 other-satisficing policy, the remaining 15 being non-satisficing. With $\alpha = 0.01$ both oracle agents learn significantly more optimal policies. The Action Oracle Agent learns 41 physically optimal policies, 10 better than optimal reactive, 39 optimal reactive, 3 other-satisficing, and 7 non-satisficing. The State Oracle learns 31 physically optimal policies, 33 better than reactive optimal, 30 optimal reactive, 5 other-satisficing, and 1 non-satisficing.

Overall, the plots for SARSA(λ) indicate that although the mean total steps for the three types of agent are close (figure 4.11), the policies that each learns varies significantly. Unlike the Eight Adjacent Squares Agent, both of the oracle agents learn optimal and better-than-optimal-reactive policies, especially when $\alpha = 0.01$.

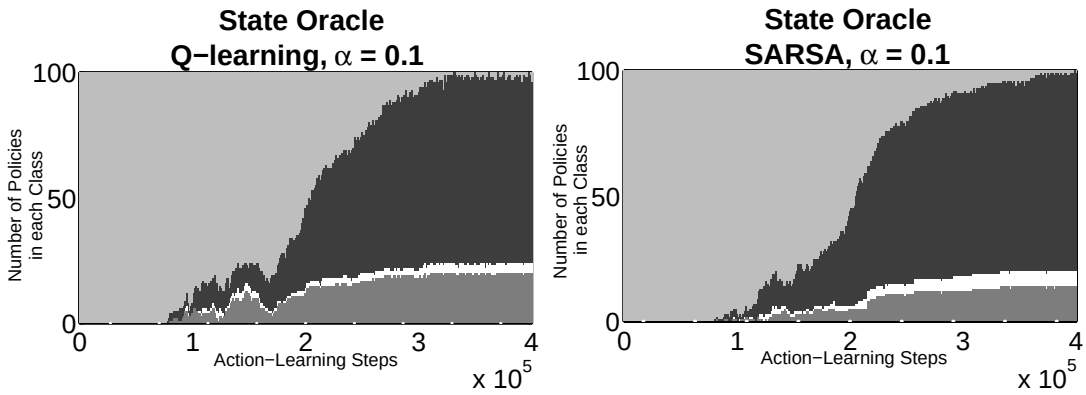


Figure 4.12: Categorisation of policies for Q-learning and SARSA versus action-learning steps. Plots are vertical stacked bar graphs with the height of the different shaded areas indicating the number of policies that fall into each classification. The policy categories used are those presented in section 4.1.1.4 and summarised in table 4.1. Plots indicate how the numbers in each policy classification change as learning progresses

Results for Watkins's Q(λ) (plots not shown) are similar to SARSA(λ) though in general Watkins's Q(λ) appears to learn satisficing policies at a slower rate and it finds significantly fewer optimal or better-than-optimal-reactive policies.

4.2.3 Discussion

The first hypothesis of this section, hypothesis (a), is clearly demonstrated by any of the example policies shown in figure 4.10. Hypothesis (a) being that reinforcement learning can, in spite of the presence of a many-to-many mapping between states and observations, learn deterministic policies which will achieved a given goal provided such policies exist. The policies shown in figure 4.10 are all satisficing policies which achieve the given goal, and the two types of oracle agents used in these examples, coupled with the design of the grid world, causes a many-to-many state to observation mapping to exist.

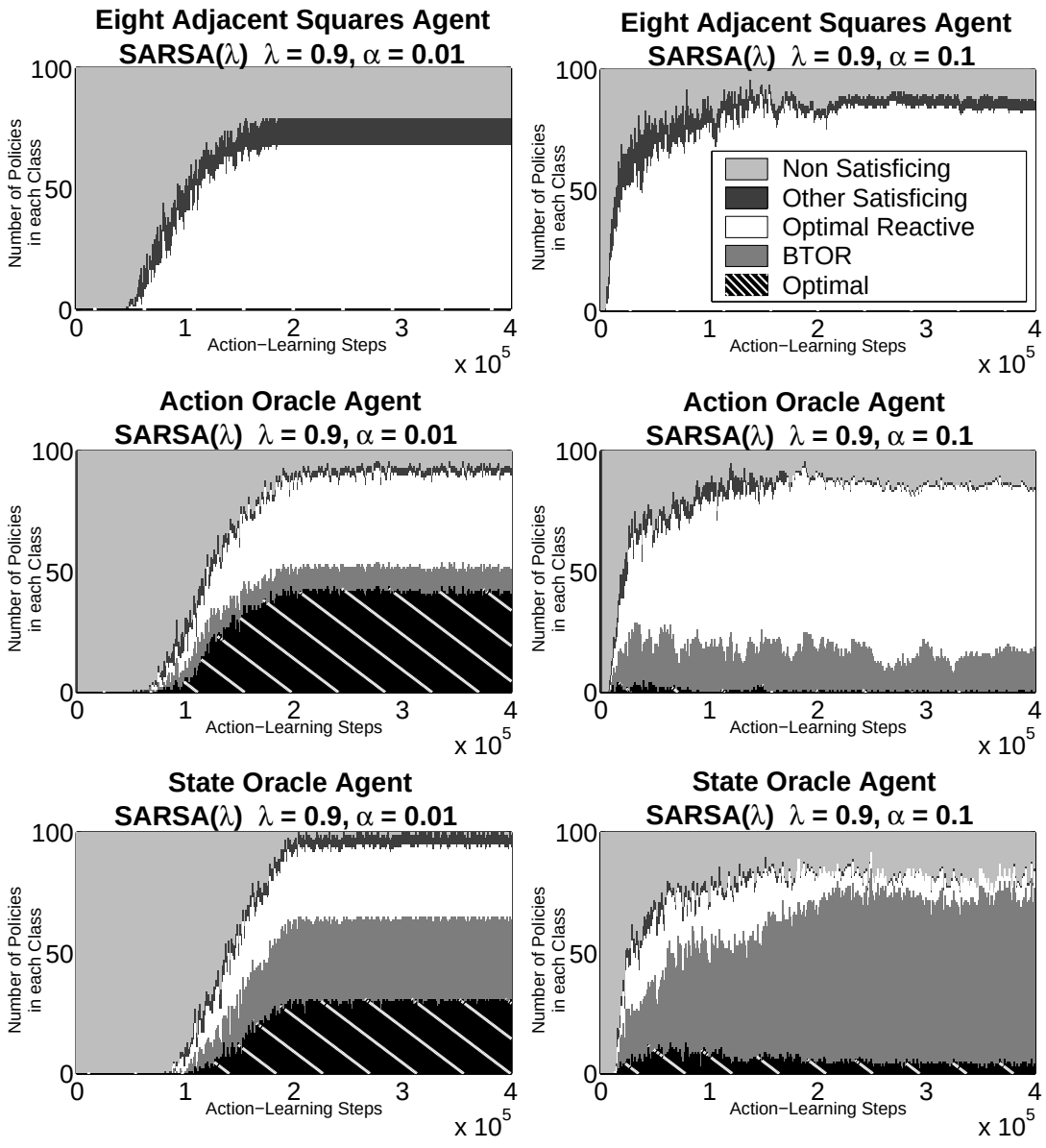


Figure 4.13: Categorisation of policies for SARSA(λ) versus action-learning steps. Plots are vertical stacked bar graphs with the height of the different shaded areas indicating the number of policies that fall into each classification. The policy categories used are those presented in section 4.1.1.4 and summarised in table 4.1. Plots indicate how the numbers in each policy classification change as learning progresses

Section hypothesis (b) is also supported by the example policies shown in figure 4.10. In general terms we see that the locations where the oracles are consulted typically correspond to places where we would expect difficulties in selecting a physical action due to state aliasing. This suggests that reinforcement learning algorithms are capable of making appropriate use of additional actions which solve state aliasing by changing the agent's view of the current state.

More specific support of hypothesis (b) is provided by the top two example policies. These two policies take less physical action steps to reach the goal from all possible starting states than is possible using an equivalent agent that does not have access to the State Oracle. These two policies take 412 and 404 physical action-steps to reach their goal compared to 416 for the optimal-reactive policy. Thus we have shown that a simple active perception system which does no more than provide alternative tokens for each state, that is the State Oracle, provides benefits in terms of the quality of satisficing policies learnt.

Support for the third section hypothesis, hypothesis (c), is provided by considering the relative performance of the Action Oracle Agent and State Oracle Agent. For the two 1-step back algorithms, SARSA and Q-learning, the results for the State Oracle Agent are clearly better than those for the Action Oracle Agent when the learning rate α equals 0.1, see figure 4.11. However the results for $\alpha = 0.01$ indicate that the ability of the oracle/active perception approach to learn satisficing policies when used in conjunction with 1-step back algorithms is very dependent on the parameter values used. Given this observation it is probably more informative to compare the results of the two oracle agents when they are coupled with learning algorithms which use eligibility traces, such as SARSA(λ) and Watkins's Q(λ).

Using SARSA(λ) and Watkins's Q(λ) the performance of both the Action Oracle Agent and State Oracle Agent are similar, both in terms of the mean total steps and the classification of policies. Overall the Action Oracle Agent's performance is marginally better than the State Oracle Agent but this difference is relatively small when compared to the difference in results obtained by the Eight Adjacent Squares Agent. This relatively small difference is very encouraging since compared to the Action Oracle,

which has access to a known optimal policy, the State Oracle has no extra information about the task. The results indicate that in order to aid an agent dealing with a partially observable task, an active perception system only has to provide non-ambiguous representations (within the context of the task) for the current state, that is, it does not have to provide any additional knowledge or reasoning about the problem. We go on to show in section 4.3, and also in chapter 8, that the alternative observations provided by the active perception system can in themselves contain aliasing, yet still be useful to the agent.

Oracles were introduced as a specific form of an active perception system which allow us to guarantee that there existed reactive policies which reached the required goal. Based on these results we can say that agents should be able to make use of more general designs of active perception systems to solve partially observable tasks provided that the combination of active perception system and task is such that reactive policies exists. Further we can say that such active perception systems can allow for better reactive policies to be learnt when compared than those that can be learnt by agents without access to additional active perception.

We do not currently envisage using oracles when implementing this technique on real robots. They are introduced as a useful simplification to aid understanding of the underlying problems. However, it is possible to imagine scenarios where resource constraints might make the use of oracles worthwhile, for example, mass produced military robots with limited computational power and sensors, which have access to a central computer to aid their exploration. To avoid overloading the central computer or to minimise the number of transmissions they make, these robots might only call on the central computer (their oracle) when unable to determine their location independently.

An alternative scenario is a planetary rover, with computational power and sensors limited by available power and payload weight. It would want to limit the amount of communication it needed with an oracle (human or computer) back on Earth due to the time delays caused by distance.

In conclusion, the oracles were introduced as an idealised active perception system. Their success suggests that (i) direct reinforcement learning algorithms can learn sat-

isficing policies despite the many-to-many mapping that exists, and (ii) that the use of active perception should provide a feasible approach to solving partially observable problems.

4.3 Agent with Active Perception

Section 4.2 demonstrated that it is possible for reinforcement learning to learn satisficing policies in spite of the existing of a many-to-many mapping between states and observations. Further, it also demonstrated that such policies are on average no worse and often better in terms of the number of physical actions taken to reach a goal. Although this is encouraging it does not offer any guarantee that an agent equipped with a more realistic active perceptual system will be able to perform as well. An agent equipped with an active perception system faces two significant complications over that faced by an agent which has access to an oracle: (i) The space of possible actions that the State Oracle Agent had to explore was only increased by a single action, the action of requesting information from the oracle. An active perception system is likely to have a range of actions over which the agent will have to select. This will represent a significant increase in the action space and will affect the speed with which the agent learns; (ii) There is no guarantee as to which of the active perception actions, if any, will return a unique observation for any given state. Even worse, some of the active perceptual actions might return the same observation for many of the states, resulting in a possible increase in the number of aliased states[3]

The number of perceptual actions that are necessary in order to be able to find better solutions to a given POMDP is in general not easy to determine. In the case of Sutton's Grid World, as used in the experiments below, it would be possible to identify a minimum set of perceptual actions. However, as we wish to simulate the situation in

[3]We assume here that the observation returned by selecting a perceptual actions is not combined with the current observation. Such an approach is justifiable on the grounds that it avoids a multiplicative increase in the observation space which the agent has to learn. See section 9.2.5 for a more detailed discussion of this point.

which most designers will find themselves we arbitrarily equip the agent with a set of eight active perceptual actions.

Assuming an agent which is provided with a set of perceptual actions which allow it to modify the observations it receives, those observations being related to the true state of the agent's task, then what we seek to show is that, despite the existence of redundancy[4] in the number of perceptual actions introduced, and the possibility of adding further aliased observations to the agent's observation space, that provided the new observations compliment those that the agent can already perceive, then it is possible for reinforcement learning to learn better policies.

4.3.1 Experimental setup

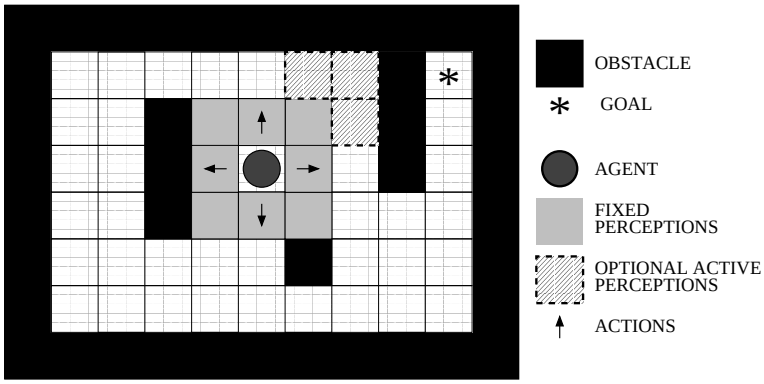


Figure 4.14: Sutton's Grid World and example active perception agent. The agent shown has, as an example, just selected the perceptual action "Look North East" and thus obtains new observation indicating the presence of obstacles in the hatched squares in that direction

We use Sutton's grid world (figure 4.14) as described previously in section 4.1.1.2. Using this world we compare two agents. One agent's observations are based on whether

[4] Meaning that all the perceptual actions introduced are not necessarily required in order to learn the physically optimal policy.

or not the eight squares adjacent to its current location contain obstacles. This agent experiences state aliasing in multiple locations of Sutton's grid world (see section 4.1.1.2 for details). The second agent normally receives the same basic observations as the first, but has eight *perceptual actions* that it can select: look north, north east, east, south east, south, south west, west or north west. On selecting one of these perceptual actions the observation it receives changes to an 11-bit string with the addition of information from three additional squares in the direction of its choosing (for example, the three hatched squares in figure 4.14). It receives a penalty of -1 for selecting a perceptual action. A policy could select a sequence of perceptual actions one after another. In such an event the information is not additive, but limited to the basic eight squares plus the three selected by the last perceptual action. The agent's observations revert back to just the eight adjacent squares after it selects a physical action. The performance of the first agent, who has no active perception, forms a baseline against which we compare the performance of the second.

The reinforcement learning algorithms SARSA, Q-learning and SARSA(λ) with replacement traces (Sutton and Barto [1998, p146,p149,p181]) were used with both agents. Parameters used: discount rate $\gamma = 0.9$; action selection ε-greedy with the probability of selecting an exploratory action (ε) initially 0.2 and decaying linearly to zero by the $500{,}000^{th}$ action-learning step; learning rate α 0.02 and 0.05. For SARSA(λ) the learning rate $\alpha = 0.02$ was coupled with an eligibility trace decay rate (λ) of 0.99, and for $\alpha = 0.05$ a value of $\lambda = 0.9$ was used. In an ad-hoc search of the parameter space, these values gave rise to the minimum mean total of perceptual actions and the maximum number of physically optimal solutions for the active perception agent when combined with learning algorithm SARSA(λ). One hundred repetitions were run for each set of parameters, each run lasting one million action-learning steps and the policy being sampled and categorised every ten thousand action-learning steps. Action values were initiated at zero for every observation.

4.3.2 Results

Results are presented in figures 4.15 and 4.16. The policies learnt are classified every ten thousand action-learning steps in accordance with the criteria set out in section 4.1.1.4 and summarised in table 4.1 on page 103. The number of policies in each category are then plotted as vertical stacked bar graphs with the height of the different shaded areas indicating the number of policies that fall into each classification. By plotting a stacked bar for each evaluation point, the plots indicate how the numbers of each policy classification change as learning progresses.

Results for Q-learning and SARSA are very poor. Plots for $\alpha = 0.02$ are not shown as no satisficing policies were learnt using this learning rate for either of these learning algorithms. Figure 4.15 shows results for $\alpha = 0.05$ for both Q-learning and SARSA. For the agent equipped with active perception neither algorithm appears to perform particularly well. Q-learning has learnt three satisficing policies whilst SARSA has learnt only one satisficing policy after a million action learning steps. For the agent without active perceptual actions both of these learning algorithms fare better, learning 59 policies that are classified as other-satisficing for Q-learning and 34 other-satisficing policies for SARSA.

The results for SARSA(λ) are much better. As can be seen in figure 4.16 the Active Perception Agent can indeed learn policies that are better than the optimal-reactive policy. The mean total physical actions of these policies are significantly less than the best achievable by a reactive agent without active perception; satisficing Active Perception Agent policies take an average of 408.5 total physical action steps compared with a best possible of 416 total physical action steps, and an average of 416.7, for the Eight Agent Squares Agent. Checking statistical significance using Student's t-test for unequal variance indicates that the results for the Active Perception and Eight Adjacent Squares Agent are very unlikely to be drawn from the same distribution (probability $p = 4 \times 10^{-40}$). Table 4.3 summaries the results for SARSA(λ).

Examining the results for physically optimal policies presented in table 4.3 we observe that the policies that are optimal in terms of physical actions do not generally use the

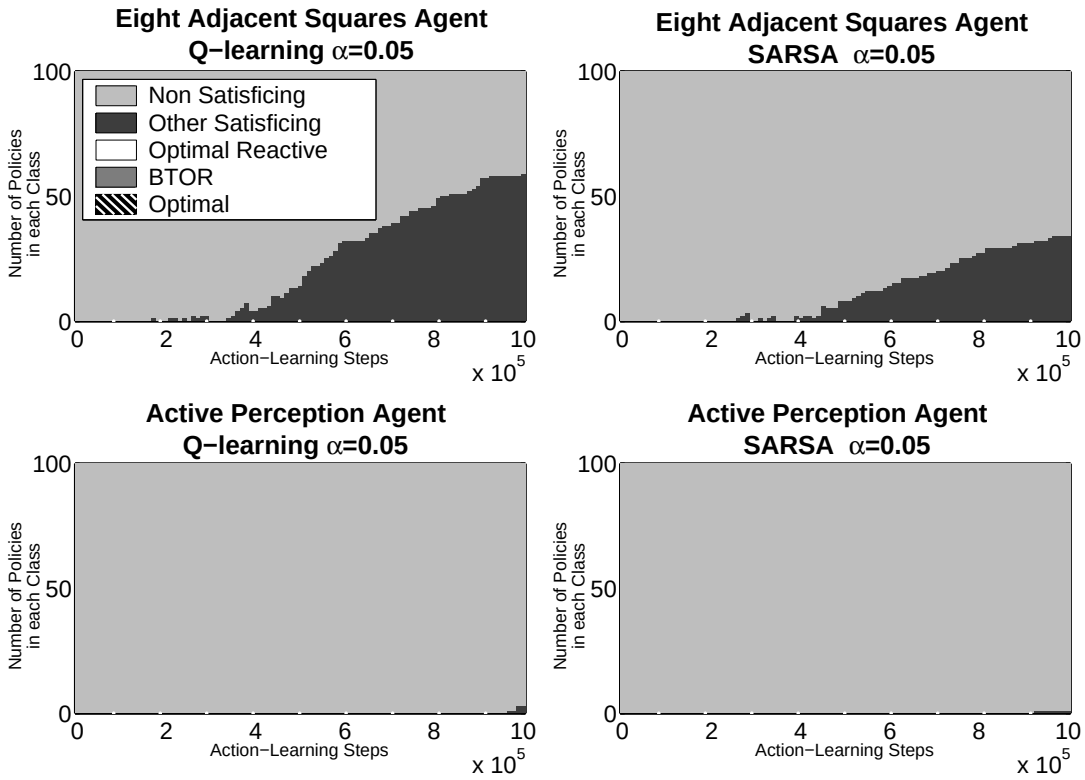


Figure 4.15: Categorisation of policies for agents with fixed and active perception versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. Top two plots are for fixed perception agent and lower two plots are for active perception agent. Left hand plots are Q-learning with $\alpha = 0.05$. Right hand plots are SARSA with $\alpha = 0.05$

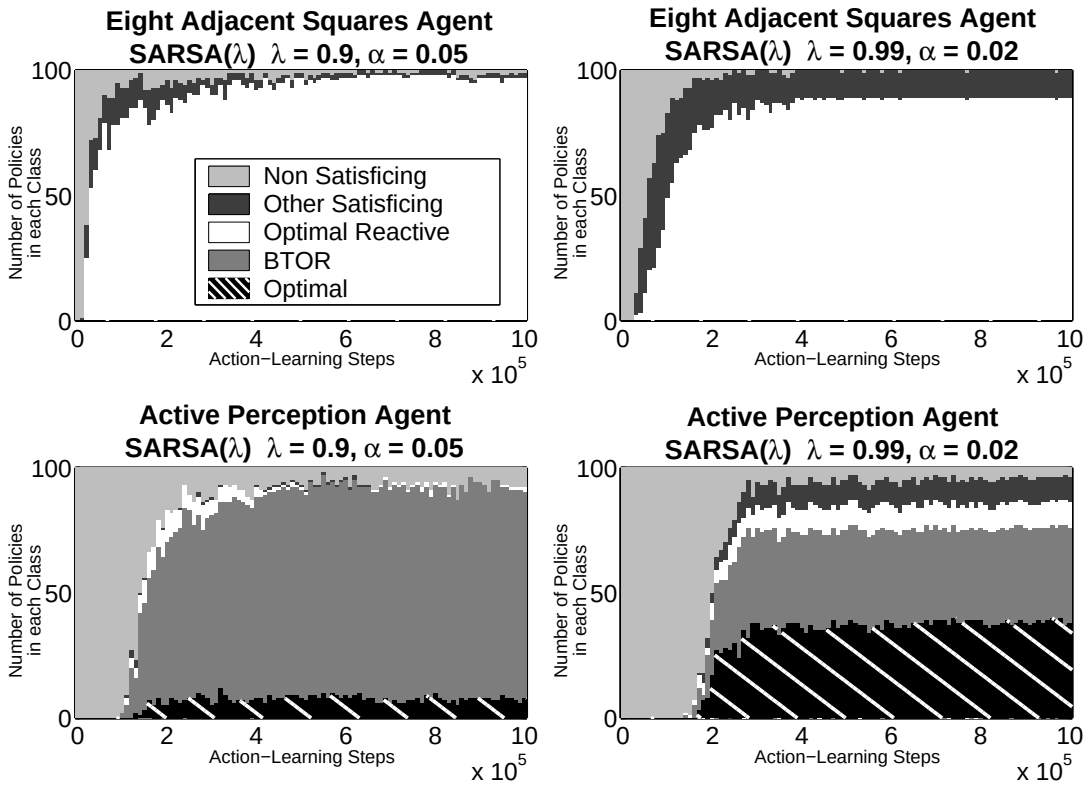


Figure 4.16: Categorisation of policies for agents with fixed and active perception versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for SARSA(λ). Top two plots are for fixed perception agent and lower two plots are for active perception agent. Left hand plots are for $\alpha = 0.05$ and $\lambda = 0.9$. Right hand plots are for $\alpha = 0.02$ and $\lambda = 0.99$

Agent	**Eight Adj. Squares**		**Active Perception**	
Learning Algorithm	SARSA(λ)	SARSA(λ)	SARSA(λ)	SARSA(λ)
λ	0.9	0.99	0.9	0.99
α	0.05	0.02	0.05	0.02
Number of Satisficing Policies	99	100	92	96
Mean total physical actions of all satisficing policies (± 95% confidence interval)	416.2 ± 0.2	417.3 ± 0.6	407.3 ± 0.4	409.8 ± 1.5
Number of Physically Optimal Policies	—	—	8	38
Mean total perceptual actions of physically optimal policies (± 95% confidence interval)	—	—	28.0 ± 14.3	30.0 ± 3.7

Table 4.3: Summary of the four SARSA(λ) policies' characteristics after one million action learning steps

minimum number of perceptual actions. Examining the task by eye we believe that a physically optimal policy, which is one which takes 404 physical actions, should only require a total of 12 perceptual actions[5].

4.3.3 Discussion

The result that we sought is confirmed by the results for SARSA(λ). We demonstrate that an agent which has a set of perceptual actions which allow it to modify the observations it receives can learn policies that are better than the optimal reactive policy. This is despite redundancy in the number of perceptual actions introduced, and the fact that these actions add further aliased observations to the agent's observation space.

[5]NB, in section 5.2.5.4 we observe a policy which we believe to take the minimum *total* actions, that is minimises the total of physical and perceptual actions. This policies uses 406 physical actions and 8 perceptual actions.

In comparison to experiments presented earlier in this chapter, the period over which exploration occurs has been lengthened from 200,000 action-learning steps to 500,000. In selecting the parameters used, we found that increasing the period over which exploratory actions can occur, that is, decreasing the rate of decay of ε, significantly reduces the number of non satisficing policies; compare plots for Eight Adjacent Squares Agent in figure 4.13 and figure 4.16. We also found that the increase in exploration resulted in fewer redundant perceptual actions in the policies learnt by the active perception agent.

Unexpectedly increasing exploration had only a small effect on the number of physically optimal solutions found, this number appearing to be more dependent on the values of α and λ, that is, in figure 4.16 more physically optimal policies are learnt by the active perception agent when λ is increased towards 1, and α is reduced. We would need to run additional experiments to determine which variable, α or λ, is the most significant in causing this effect.

This section provides support for both our first and second thesis hypothesis:

- In support of our first thesis hypothesis, we demonstrate that an agent enhanced with active perception can learn satisficing reactive policies that are better that those achieve by a non-enhanced agent.
- In support of our second thesis hypothesis, we show that an agent equipped with active perception can learn satisficing reactive policies without the learning algorithm distinguishing between physical and perceptual actions.

4.4 McCallum's M-maze

McCallum's M-maze (also know as McCallum's cheese maze) [McCallum, 1995b] is shown in figure 4.17. The aim of the agent that inhabits this world is to learn a policy such that starting from any location, it can find its way to the goal. The agent's observations are limed to four adjacent squares, north, south, east and west, which are

also the directions it can move in. As indicated by the numbers in figure 4.17, the agent's observations alias three sets of locations; these are the grid squares labelled 0, 1 and 2. It is known that no stationary, deterministic reactive policies exist for this grid world using this agent [McCallum, 1995b; Littman, 1994].

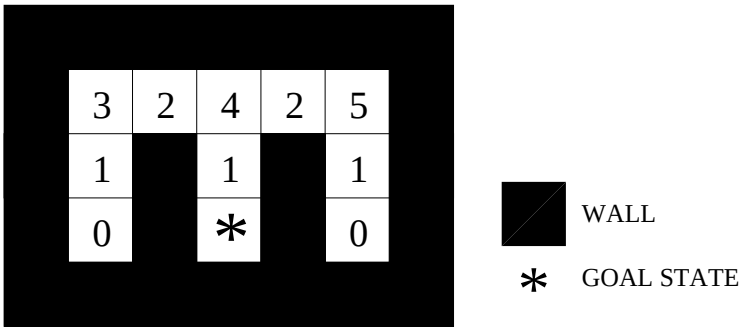


Figure 4.17: McCallum's M-Maze. Number shown indicate observation seen by agent

Littman [1994] demonstrated that a modifiable approach style agent enhanced with a 1 bit internal memory could learn a stationary, deterministic reactive policy for this problem. We show below that similarly extending the agent, but this time with active perception, also makes a stationary, deterministic reactive policy possible.

In order to learn the optimal policy in this case, the agent has to find distinguishing features for five of the eight aliased states. These are the five states where it has to move in different directions depending on which leg of the maze it is in. We could provide some active perception system that will find features at these points, but for the purpose of simply demonstrating how this makes the problem learnable, we simply extend the agent's abilities by allowing it access to a state oracle, as used previously in section 4.2.

A run of our temporal difference, direct reinforcement learning algorithm, CEQ(λ), which we will introduce in chapter 7, learnt the satisficing policy shown in figure 4.18. The parameters used were learning rate $\alpha = 0.1$, discount factor $\gamma = 0.99$, eligibility trace discount factor $\lambda = 0.99$ using replacement eligibility traces, ε-greedy action selection with ε initiated at 0.2 and would have reached zero after 500,000 action-learning steps. The policy was learnt at some point before 200,000 action-learning

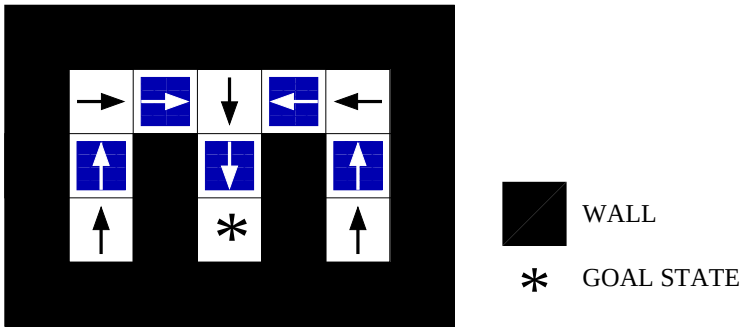


Figure 4.18: Active perception policy for McCallum's M Maze. Filled squares containing arrows indicate where the state oracle is consulted and action then taken based on the observation received from the oracle

steps had passed. Each physical action has an associated cost of -0.1 provided the agent does not attempt to move towards a wall. If it attempts to moves towards a wall a penalty of -1 is received and it remains in its current location. Each request to the oracle has an associated cost of -0.1 in line with the basic cost of a physical action. Section 4.2.1.2 sets out the reasoning for selecting equal physical and perceptual action costs.

Thus, we demonstrate the ability of active perception, though extending the action and observation space available to the learning algorithm, to bring into existence satisficing policies on problems where none previously existed.

4.5 Limitations on Learning of Physically Optimal Policies

Examining the results presented in section 4.2 and section 4.3, for agents with access to oracles and agents with active perception, it appears that the frequency with which physically optimal policies are learnt is limited and possibly dependent on the value of α. Two possible causes of this limit could be: (i) the cost of perceptual actions is too high; (ii) there is some step change in complexity of the policies between better-than-

optimal-reactive and physically optimal policies for Sutton's Grid World that limits the probability of reinforcement learning converging on such solutions.

Unfortunately from these experiments we are unable to rule out either effect. Based on our examination of the task, we believe that a physically optimal policy that uses the minimum number of perceptual actions, requires 12 perceptual actions (or requests to the oracle) and 404 physical actions. This gives a total of 416 actions. Given that the agent receives the same level of penalty for each perceptual action (or request to an oracle) as it does for physical actions, then such a policy has the same total cost as an optimal reactive policy, the latter requiring a total of 416 physical actions. We did not differentiate between the cost of perceptual and physical actions in order to ensure effective learning when using discounted future rewards, see observations below.

Results presented in the following chapter which look at whether the policies learnt are local minima suggest that hardly any of the policies have converged to local minima. This makes it unlikely that the cost of perceptual actions is limiting the number of physically optimal policies learnt.

In terms of the cost of perceptual actions we did considered the possibility of reducing this cost in section 4.2.1.2. As we note in that section, if future rewards (and costs) are discounted then for an agent learning a minimum path problem there is a limited range of active perception costs which will work. Given any action that attracts a lesser penalty, the cost of this action when summed to infinity using a given discount rate can work out less costly than selecting the actions required to reach the goal. Thus an agent finds it preferable to continually select a perceptual action, or converse with the oracle, rather than attempting to move towards the goal. In Sutton's Grid World, when we use a discount rate (γ) of 0.9, solutions that reach the goal from all starting locations will only be the least costly for perceptual action penalties that are more negative than -0.8. This range however is task specific, and identifying it requires some knowledge of the problem, specifically maximum path length. Given that we wish to apply these methods to more complex tasks where the maximum path length might be difficult to determine, we we would prefer not to have to include this knowledge in the reward structure chosen. The simple solution adopted in this thesis is to equate physical and

perceptual costs. We return to the issue of differential costing of perceptual and physical actions in section 9.5.

Note that none of these observations undermine the hypothesis which we set out to test in this section. The focus was whether or not policies which were better than the optimal-reactive policy could be learnt by active perception agents, and did not consider the reliability with which such policies are learnt. The ability of learning algorithms to converge reliably on partially observable tasks and the extent to which they converge on better than optimal-reactive policies is the focus of subsequent chapters.

4.6 Chapter Summary

The three experiments presented in this chapter have demonstrated the following results.

- Provided there is some chance of selecting an explorative action, then 1-step backup reinforcement learning algorithms cannot learn policies which are both stable and optimal for tasks which involve perceptually aliased states. In the case where exploration *ceases* and thus the agent always acts greedily then the experiments in section 4.1 show that 1-step backup learning algorithms *can* converge on satisficing policies.
- Reinforcement algorithms that use eligibility traces can learn optimal reactive policies. Our results extend the work in Loch and Singh [1998] as we show that optimal reactive solutions can also be found using Watkins's Q(λ) using accumulating traces. Loch and Singh [1998] demonstrated the result only for SARSA(λ) using replacement eligibility traces. This reinforces their conjecture that eligibility trace based reinforcement learning methods are useful in learning POMDP tasks for which there exists good reactive policies.
- When deterministic policies which achieve a given goal exist, it is possible for reinforcement learning to learn such policies even in the presence of a many-to-many state to observation mapping. This is important as such many-to-many

mappings will occur when an agent is equipped with active perception in a partially observable environment.

- A simple active perception system which does no more than provide alternative tokens for each state can provide benefits in terms of the quality of satisficing policies learnt. In other words, no additional knowledge of the task or extra analysis need be provided by an active perception system for it to be effective.
- An agent which has a set of perceptual actions that allow it to modify the observations it receives can learn policies which are better than the best policy that can be learnt by an agent not equipped with active perception. This is achieved in spite of redundancy in the number of perceptual actions and the possibility of these actions adding further aliased observations to the agent's observation space.
- The addition of active perception can allow an agent to learn satisficing, deterministic, reactive policies for partially observable tasks, where no satisficing, stationary, deterministic policies could have been expressed before.

Thus in this chapter we have shown supporting evidence for both of our thesis hypotheses as set out in section 1.2. However, in the experiments presented here, none of agent and algorithm combinations succeeded in *consistently* learning the best possible policy, and often a small number of policies would fail to even converge to satisficing solutions. As we are interested in comparing active perception approaches with other approaches in the literature, and one of our main points of comparison is reliability, we turn our attention to possible changes that might improve convergence of policies. At the heart of our active perception approach we are attempting to learn reactive policies for partially observable problems, using direct reinforcement learning algorithms. In the chapters that follow, we therefore consider approaches that might improve the performance of direct reinforcement learning algorithms on such problems.

Chapter 5

Effects of Action Selection

In this chapter we undertake a preliminary examination of how exploration can affect the existence of stable policies for POMDPs, and the convergence of learning algorithms to these points.

Our aim is to propose methods that may provide robust performance when direct reinforcement learning is applied to POMDPs. Existing work has demonstrated that sometimes no stable policies exist when using ε-greedy exploration (where $\varepsilon > 0$) [Whitehead and Ballard, 1991; Perkins and Pendrith, 2002] but stationary points do exist when using continuous action selection methods to guide exploration. We consider a third approach where after a period of exploration the agent acts greedily. We demonstrate that this approach can result in stable policies for reinforcement learning algorithms that update observation-action values based solely on the experience provided by the current policy. We then compare the results obtained using such greedy agents to those achieved using a continuous action selection method such as the Gibbs/Boltzmann distribution to guide action selection.

Chapter 4 demonstrates that, when an agent is equipped with additional control over its perception of the environment, it is possible for reactive reinforcement learning algorithms to utilise these actions to learn better policies, in spite of the increased possibility of state aliasing. However, the same experiments indicate that the addition of

these perceptual actions decreases the likelihood of convergence of individual policies. This is not solely an issue of convergence to *optimal* policies, but more worryingly the failure of some runs to converge to *satisficing* policies.

To support our first thesis hypothesis we are interested showing empirically "that the performance of active perception agents is comparable with other approaches in the field of reinforcement learning" (section 1.2). As we are interested in apply these techniques to agents embedded in the real world, the focus of our comparison is on the reliability with which agents can learn satisficing policies. Given that the heart of the active perception approach involves direct reinforcement learning of satisficing, deterministic, reactive policies in partially observable worlds, we are interested in techniques that can improve the reliability with which direct reinforcement learning algorithms converge on POMDP problems. This is the area that we examine in this and the following chapter (chapter 6).

An underlying problem with the reactive approach that we are pursuing is that we are applying reinforcement learning (RL) to partially observable Markovian decision processes (POMDPs). "Unfortunately all of the elegant theory of RL is limited to Markovian decision processes (MDPs)" and once we apply reinforcement learning to POMDPs all the nice guarantees are lost [Singh et al., 1994].

Once the Markov property, formally expressed as

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\} = \Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} \tag{5.1}$$

is broken, it is no longer possible to model the complete process using the one-step relationship $\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t=s, a_t=a\}$ for all s and a. This creates a series of issues for convergence of learning algorithms in POMDPs:

(i) There is no longer an assurance that there exists a set of optimal policies π^* that share the *same* optimal state-value function V^*, where $V^*(s) = max_\pi V^\pi(s)$ for all s. In other words, it may not be possible to find a policy which simultaneously maximises the value of *every* state.

(ii) In MDPs incremental improvements in policies always move you in the right direction [Pendrith and McGarity, 1998], that is, for a MDP, changing a single action in such a way that it improves the policy, always leads to an increase in the policy value and the value of individual state-action pairs. This a guarantee of the absence of local minima in MDPs, something that is definitely not true in POMDPs [Pendrith and McGarity, 1998].

(iii) 1-step backing up of values (as used by Q-learning and SARSA [Sutton and Barto, 1998]) can no longer be relied on to propagate back the correct value to earlier states, as a POMDP cannot be modelled using a one-step probabilistic relationship, such as $\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a\}$. This effect was labelled as global impairment by Whitehead [1992].

The few theoretical guarantees that have been demonstrated for reinforcement learning in POMDPs are somewhat weaker:

(i) Perkins and Pendrith [2002] show that for continuous action selection functions, such as a Gibbs/Boltzmann distribution (equation 5.23 on page 164), stationary learning points exist for the observation-action values when using Q-learning and SARSA. Perkins and Pendrith [2002] further speculate that this result may hold for SARSA(λ). In section 5.2 we examine the quality of policies that result from using continuous action selection in practice.

(ii) Pendrith and McGarity [1997, 1998] indicate that calculating state values using undiscounted first visit Monte Carlo returns ensures that the relationship $V^*(s) = max_\pi V^\pi(s)$ still holds for POMDPs and thus policies that satisfy this relationship will be stationary points when using reactive reinforcement learning. The following chapter, chapter 6, looks at the quality of policies learnt using undiscounted Monte Carlo approaches.

(iii) Perkins's Monte Carlo Exploring Starts for POMDPs (MCESP) algorithm [Perkins, 2002b] extends the results presented by Pendrith and McGarity [1998] in (ii) above, by allowing discounted returns and claiming a guarantee of convergence on POMDPs. The effectiveness of this algorithm is examined in chapter 6.

The guarantees that are provided for the MCESP learning algorithm, point (iii) above, are achieved through control of exploration, ensuring changes in policies are Lipszich continuous. Similarly (i) deals with control of exploration through the method of action selection used. Thus an important theme for learning in POMDPs is control of the actions the agent selects, specifically exploratory actions taken by the agent.

We regard this chapter as preliminary work in this area, as our comparisons are conducted on two grid world POMDP problems, though we demonstrate the effectiveness of the greedy approach on further problems in chapters 7 and 8. Also, we are yet unable to offer general theoretical proofs of the stability, or otherwise, of POMDP policies for a greedy agent.

5.1 Dependency Between Observation-Action Values and Exploration

For a Markov decision process (MDP) there is a unique solution to the Bellman Optimality Equation for state-action values $Q^*(s,a)$:

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') \mid s_t = s, a_t = a\} \quad (5.2)$$

$$= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma max_{a'} Q^*(s',a')] \quad (5.3)$$

where s is the state, a is the action selected from that state, $E\{\ldots\}$ is expectation, r_{t+1} is the reward received at the next time step, γ is the discount factor, $P^a_{ss'}$ is the probability of transition from state s to state s' when selecting action a, and $R^a_{ss'}$ is the average reward received on transition from state s to state s' when selecting action a [Sutton and Barto, 1998].

The unique solution Q^* defines the optimal policy (or policies[1]) for the MDP which is given by $\pi^* = \arg\max_a Q^*(s,a)$. No element of the Bellman Optimality Equation for $Q^*(s,a)$, equation 5.3, is dependent on the current policy π which the agent is

[1] multiple state-action pairs may have the same value.

following. Thus irrespective of whether π is for example $\arg\max_a Q(s,a)$ or a random walk, the optimal solution Q^* (and hence π^*) is unaffected.

For POMDPs the equivalent equation for observation-action values $Q^*(o,a)$, as opposed to state-action values, does not have a unique solution as violation of the Markov principle means that the reward term r^{π}_{t+1} or $R^{a,\pi}_{oo'}$ and probability of transition between observations $P^{a,\pi}_{oo'}$ can depend on previous actions and states visited. This means that they are dependent on the current policy π. This we have indicated using the superscript π.

$$Q^*(o,a) = E\{r^{\pi}_{t+1} + \gamma \max_{a'} Q^*(o_{t+1},a') \mid o_t = o, a_t = a\} \quad (5.4)$$

$$= \sum_{s'} P^{a,\pi}_{oo'}[R^{a,\pi}_{oo'} + \gamma max_{a'} Q^*(o',a')] \quad (5.5)$$

For MDPs, *off-policy* algorithms such as Q-learning and Watkins's Q(λ) estimate, through repeated sampling of the state action space, state-action values $Q(s,a)$ that satisfy equation 5.3. *On-policy* algorithms such as SARSA and SARSA(λ) differ in that they learn $Q(s,a)$ values that are relate to the policy they are following. However, SARSA does provably converge to $Q(s,a)$ values that satisfy the equation 5.3 if exploration is reduced in an appropriate fashion [Singh et al., 2000], and it is generally assumed that a similar result holds for SARSA(λ). When we apply the same learning algorithms to a POMDP it can therefore be seen as an attempting to find *observation-action* values $Q(o,a)$ that satisfy the observation-action variation of the "Bellman Optimality Equation", that is equation 5.5. Assuming that the algorithms can find solutions to equation 5.5, it is of interest to understand how these solutions will vary with variation in the agent's current policy.

Just to clarify the point, when we speak of policy in this regard we include consideration of how each and every action is selected, that is, how the agent selects exploratory actions and not just the underlying greedy policy.

To undertake this analysis we use two small POMDP problems that separately demonstrate the dependence of $R^{a,\pi}_{oo'}$ and $P^{a,\pi}_{oo'}$ for POMDPs on the policy being followed. For these two examples we consider in detail the variation in solutions to equation 5.5 in

order to gain some insight into the problem. We especially consider the case when exploration has ceased and the agent follows a fixed policy.

5.1.1 Perkins and Pendrith's simple POMDP example

Perkins and Pendrith [2002] provide a simple example for which the observation-action values converge when using the continuous action selection but oscillate when using the discontinuous action selection function of ε-greedy, $\varepsilon > 0$. We consider policy stability for ε-greedy, greedy and continuous action selection methods in section 5.1.2 but our current focus is on solutions to equation 5.5. Our analysis, which is presented below, extends the work of Perkins and Pendrith [2002] in that we consider the full range of values that the exploration parameter ε can take. Through this we gain some insight into the variation of policy stability for different values of ε.

The environment used by Perkins and Pendrith is shown in figure 5.1. The four circles represent four states of the environment. The letters inside the circles indicate the observation as seen by the agent. Thus to the agent in this environment the two right hand states labelled C appear the same. The arrows represent deterministic state transitions which occur when the agent selects the corresponding action. Each arrow is labelled with two digits representing the action and the reward obtained for taking that action. From three of the states only one action is available, action 0. The state labelled with observation A is the only state to offer a choice of actions; action 0 and action 1.

Using the Q-learning update rule, what observation-action values should the agent learn? Observation-action pair $(A,0)$ is always followed by zero reward and observation B, thus the agent should learn:

$$Q(A,0) = \gamma Q(B,0) \tag{5.6}$$

Similarly observation-action pair $(B,0)$ is always followed by zero reward and observation C:

$$Q(B,0) = \gamma Q(C,0) \tag{5.7}$$

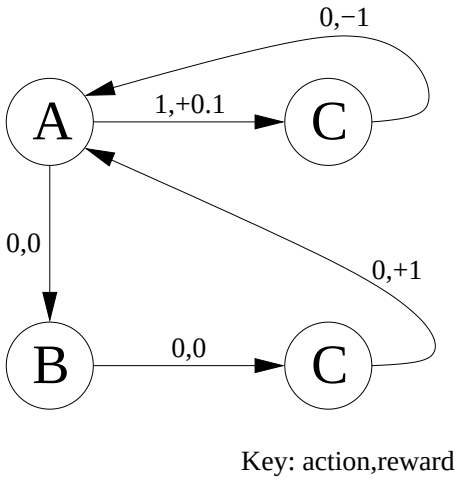


Figure 5.1: Example POMDP from Perkins and Pendrith [2002]

Pairing $(A,1)$ is followed by a reward of 0.1 and then state C:

$$Q(A,1) \quad = \quad 0.1+\gamma Q(C,0) \tag{5.8}$$

The case is most interesting for observation C. For a given policy π the reward for moving from observation C to A will be $+1$ for $\pi(A,0)$ fraction of the time, and -1 for $\pi(A,1)$ fraction of the time, that is, the reward obtained will depend on which of the underlying states the system is in, which depends on the prior choice made at observation A. In either case the observation following C is A, thus when using Q-learning the agent should learn the observation-action value:

$$Q(C,0) = (\pi(A,0)-\pi(A,1))+\gamma max_{a\in\{0,1\}}Q(A,a). \tag{5.9}$$

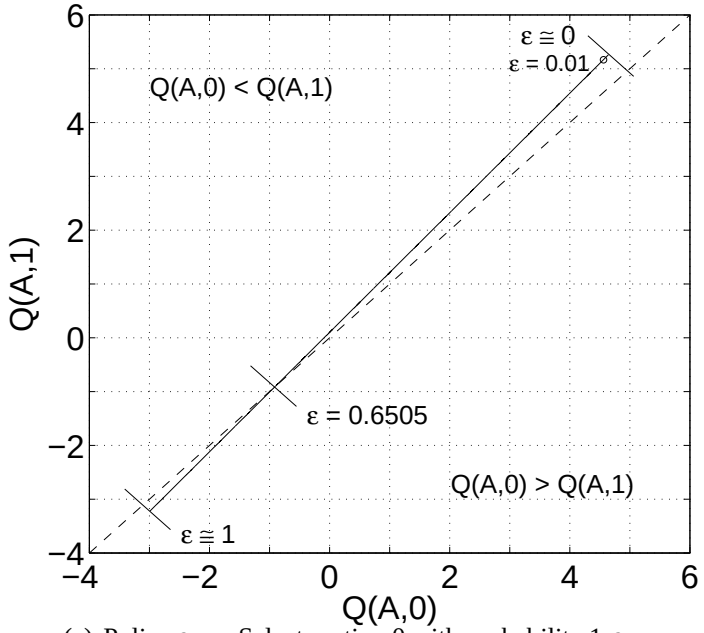
Equation 5.9 clearly demonstrates a dependence of $R_{oo'}^{a,\pi}$ on the agent's policy. The reward gained when executing action 0 at C is dependent on the proportion of the time that the policy being followed selects action 0 when at A and the proportion that action 1 is selected at A.

We can solve this set of equations to determine the observation-action values $Q(A,0)$, $Q(A,1)$, $Q(B,0)$ and $Q(C,0)$ for various policies, with the two values of most interest being the relative values of $Q(A,0)$, $Q(A,1)$. Using $\arg\max_a Q(o,a)$ the relative value

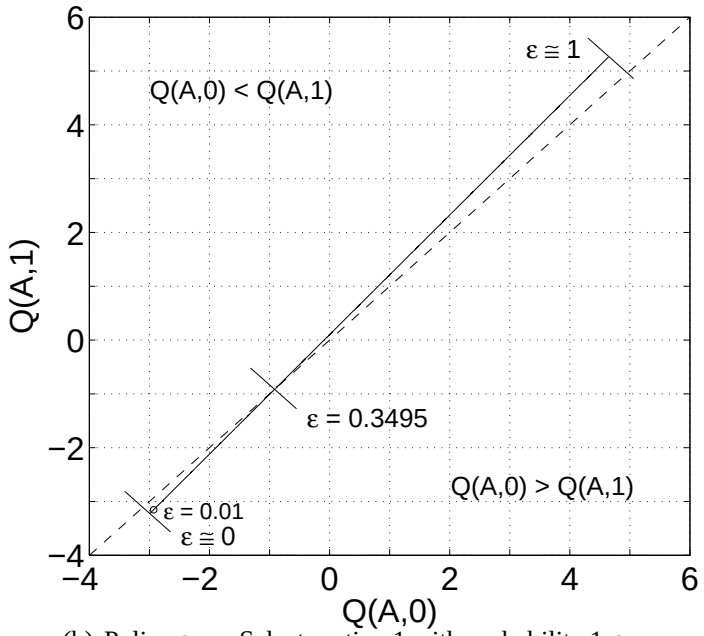
of these two will determine the policy being followed (where *o* is the observation seen by the agent, that is, A, B or C; and *a* is the selected action, that is, 0 or 1.)

Let us introduce two policies: (i) ε-π_0 (read as *epsilon pi nought*, like ε-greedy, that is with a hyphen not a minus sign) that with probability $1-\varepsilon$ selects action 0 for observation *A*, otherwise selects action 1 with probability ε; and (ii) ε-π_1 that selects action 1 for observation *A* with probability $1-\varepsilon$ and action 0 with probability ε, where $1 < \varepsilon < 0$. If we assume a value of 0.9 for γ we can plot the values of $Q(A,0)$, $Q(A,1)$ that satisfy the above set of equations as ε is varied. Figure 5.2(a) shows such a plot for policy ε-π_0 and figure 5.2(b) is a plot for policy ε-π_1.

The two figures 5.2(a) and 5.2(b) show the variation of values which satisfy equations 5.6—5.9 as ε is varied. These values form a a continuous line bounded at either end as ε approaches 0 or 1. Unsurprisingly the two plots are virtually identical with the exception that ε in one has the value of $1-\varepsilon$ from the other. A dotted line lies along the path $Q(A,0) = Q(A,1)$ dividing both plots into two regions; (i) top left region where $Q(A,0) < Q(A,1)$ and (ii) bottom right where $Q(A,0) > Q(A,1)$. The significance of these two regions is that when using $\arg\max_a Q(o,a)$ to select the policy, these two regions represent different policies. The line representing values of $Q(A,0)$ and $Q(A,1)$ that satisfies equations 5.6—5.9 for different values of ε crosses from one region to the other, the value of ε where it crosses being marked on both plots. If we consider figure 5.2(b) the solid line defines the point, for a given value of ε, towards which the values estimated by Q-learning, etc. for $Q(A,0)$ and $Q(A,1)$ will converge when following the policy ε-π_1. If we take two differing values of ε which lie either side of 0.3495, the values towards which the learning algorithms will converge fall in different regions of the plot. Thus we can see that if such a learning algorithm was following an ε-greedy policy, that is, selects the next action to execute according to $\arg\max_a Q(o,a)$ with probability of $1-\varepsilon$ and another action with probability ε, then if it started with following a policy that resembles ε-π_1, that is, $Q(A,0) < Q(A,1)$, then depending on the value of ε it can converge towards two completely different policies for this problem. One of these policies will not be stable for $\arg\max_a Q(o,a)$, as its on the "wrong" side of the dotted line, but nevertheless we have demonstrated that the policy towards which algorithms like Q-learning converge will, for a POMDP, depend



(a) Policy ε-π_0. Selects action 0 with probability 1-ε.



(b) Policy ε-π_1. Selects action 1 with probability 1-ε.

Figure 5.2: Variation in "Bellman Optimality Equation" solutions for example POMDP for variation in ε between 0 and 1. The dashed line divides the space into two regions. Above the dashed line lies the region where $Q(A,0) < Q(A,1)$, below the dashed line is where $Q(A,0) > Q(A,1)$. We use the symbol ≅ to indicate that at the ends of the plotted line ε is infinitely close to but not actually equal to 0 or 1. Plots for $\gamma = 0.9$

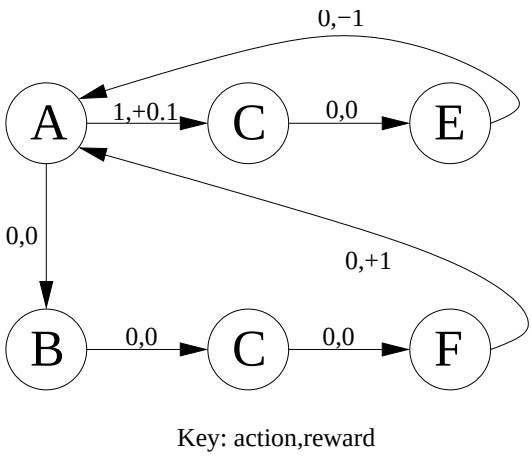


Figure 5.3: Variation in transition probabilities with exploration on example POMDP

on the current policy. We return to this problem and consider the stability of policies under reinforcement learning algorithms in section 5.1.2.

We now present a POMDP which we have developed by modifying Perkins and Pendrith [2002]'s original, in order to show that the same effect can occur though variation of the transition probabilities $P_{oo'}^{a,\pi}$, whose values are also dependent on the current policy. The modified form of the problem is shown in figure 5.3. The set of equations that satisfy the "Bellman Optimality Equation" for this problem are given by:

$$Q(A,0) = \gamma Q(B,0) \tag{5.10}$$

$$Q(A,1) = 0.1 + \gamma Q(C,0) \tag{5.11}$$

$$Q(B,0) = \gamma Q(C,0) \tag{5.12}$$

$$Q(C,0) = \pi(A,0)\gamma Q(F,0) + \pi(A,1)\gamma Q(E,0) \tag{5.13}$$

$$Q(E,0) = -1 + \gamma \max_a Q(A,a) \tag{5.14}$$

$$Q(F,0) = +1 + \gamma \max_a Q(A,a) \tag{5.15}$$

The value of $Q(C,0)$ is now determined not by the likely reward, but by the likelihood of transition to observation E or F which is dependent on the decision made at A.

Again the only decision point is when observation A is seen, so the observation-action values of interest are $Q(A,1)$ and $Q(A,0)$. As before we plot the variation of the values

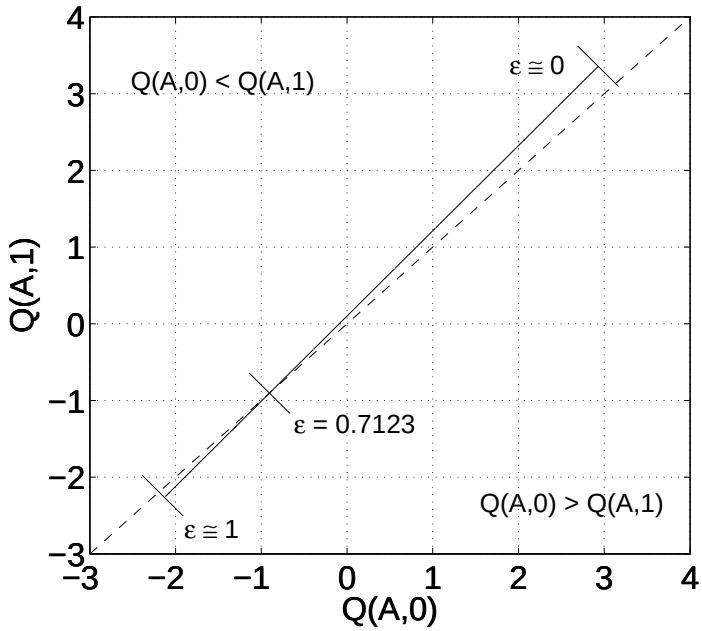


Figure 5.4: Variation in "Bellman Optimality Equation" solutions for modified example POMDP for variation in ε between and 1 for policy ε-π_0. The dashed line divides the space into two regions. Above the dashed line lies the region where $Q(A,0) < Q(A,1)$ below the dashed line is where $Q(A,0) > Q(A,1)$. We use the symbol $\cong$ to indicate that at the ends of the plotted line ε is infinitely close to but not actually equal to 0 or 1. Plot for $\gamma = 0.9$

for $Q(A,1)$ and $Q(A,0)$ that satisfy equations 5.10—5.15 as ε is varied for an agent following policy ε-π_0, where $0 < \varepsilon < 1$. The plot shown in figure 5.4 shows the same characteristics as figure 5.2(a). Thus we can conclude that the policy towards which algorithms like Q-learning will converge will again depend on the current policy.

5.1.2 Policy stability

The POMDP example in figure 5.1 was developed by Perkins and Pendrith [2002] to demonstrate that when ε-greedy action selection is used there is no guarantee that it can converge to a stable policy. We briefly reconstruct their argument by considering points marked $\varepsilon = 0.01$ on figures 5.2(a) and 5.2(b). Consider Q-learning using ε-

greedy to select actions, that is, it selects an action using $\arg\max_{a\in\{0,1\}} Q(s,a)$ with probability $1-\varepsilon$ and some other action with probability ε. If $Q(A,0) > Q(A,1)$ then this is equivalent to ε-π_0 and the Q-values will converge towards the point $\varepsilon = 0.01$ in figure 5.2(a). Unfortunately this point lies in the region where $Q(A,0) < Q(A,1)$, so at some point as the values converge, the action selected by $\arg\max_{a\in\{0,1\}} Q(s,a)$ changes from 0 to 1, and as a result the ε-greedy policy changes from resembling ε-π_0 to instead resembling ε-π_1. This changes the point of convergence to that labelled $\varepsilon = 0.01$ in figure 5.2(b). However, this point lies in the region $Q(A,0) > Q(A,1)$, so again at some point during convergence the result returned by $\arg\max_{a\in\{0,1\}} Q(s,a)$ will revert to 0, resulting in another change in the ε-greedy policy. Thus Q-learning and similar algorithms when used with ε-greedy with $\varepsilon = 0.01$ will flip back and forth between the two possible policies and not converge on a single policy.

In contrast to this result for ε-greedy, Perkins and Pendrith [2002] demonstrate that convergence to a single policy will occur for for Q-learning and SARSA for any ergodic POMDP when *a continuous action selection strategy* is employed. We examine the quality of such policies in section 5.2.

We note through our analysis in section 5.1.1 that potentially stable solutions do exist for this problem when using ε-greedy action selection. For $\varepsilon > 0.3495$ a stable policy resembling ε-π_1 can be seen to exist. This is the point where the line in figure 5.2(b) crosses into the region where $Q(A,0) < Q(A,1)$, although an interesting observation this is probably of little practical consequence. Of more practical use is the observation that the way in which Q-learning and similar methods sample observation-action values can be exploited to provide stationary policy points for an agent that is acting greedily, that is, acting as if $\varepsilon = 0$.

5.1.2.1 Acting greedily

If we were to substitute $\varepsilon = 0$ into equations 5.6—5.9, along with some reasonable value of γ, we would arrive at observation-action values not dissimilar to those for $\varepsilon = 0.01$. The values indicate, as above, that the observation-action values converged

towards when following ε-π_0 lie in the region of the observation-action space where $Q(A,0) < Q(A,1)$, that is, in the region where if we were using ε-greedy the policy would resemble ε-π_1. This suggests that neither policy would be stable for a greedy agent. However this analysis is flawed. The target values for $Q(A,0)$ and $Q(A,1)$ plotted in figures 5.2(a) and 5.2(b) are the values that an idealised reinforcement learning algorithm would converge on given some small probability of sampling *all* observation-action values and an infinite amount of time. Q-learning and SARSA algorithms update their observation-action values based solely on experience, thus, in order to guarantee convergence of observation-action values, an agent has to sample every observation-action combination [Sutton and Barto, 1998, p.122]. For $\varepsilon > 0$ we can assume there is the possibility of every action being sampled in every observation. Thus given a sufficiently long period using ε-greedy action selection, all of the observation-action combinations will be visited sufficiently to allow convergence towards the plotted values. This is not the case when one policy is being followed exclusively, without any exploration. Any observation-action combination that does not form part of the policy will not be visited and thus the associated observation-action value will not be updated.

If we assume that $Q(A,0) > Q(A,1)$ and the agent acts greedily then the set of observation-action values that satisfy the following equations represent stationary policies.

$$Q(A,0) = \gamma Q(B,0) \tag{5.16}$$
$$Q(A,1) < Q(A,0) \tag{5.17}$$
$$Q(B,0) = \gamma Q(C,0) \tag{5.18}$$
$$Q(C,0) = 1+\gamma Q(A,0) \tag{5.19}$$

These are essentially equations 5.6—5.9 but with equation 5.8 removed and replaced by the condition $Q(A,1) < Q(A,0)$. As the agent is acting greedily it never samples $Q(A,1)$ so the value is never updated hence, equation 5.8 does not hold.

Similarly there exists a set of observation-action values assuming a greedy agent for the stationary policy that results from the condition $Q(A,0) < Q(A,1)$.

$$Q(A,0) < Q(A,1) \tag{5.20}$$

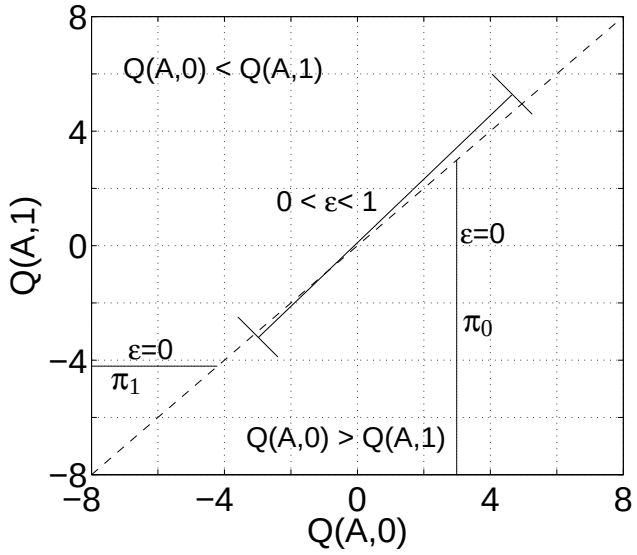


Figure 5.5: Variation in "Bellman Optimality Equation" solutions for example POMDP for variation in ε between 0 and 1 for policy ε-π_0. In addition lines are plotted for greedy policies where (i) $Q(A,0) > Q(A,1)$ and (ii) $Q(A,1) > Q(A,0)$. The dashed line divides the space into two regions. Above the dashed line is where $Q(A,0) < Q(A,1)$ below the dashed line is where $Q(A,0) > Q(A,1)$. Plot is for $\gamma = 0.9$

$$Q(A,1) = 0.1 + \gamma Q(C,0) \tag{5.21}$$

$$Q(C,0) = -1 + \gamma Q(A,1) \tag{5.22}$$

Note in this case the value of Q(B,0) is undetermined as it plays no role in the policy behaviour nor is it sampled. Figure 5.5 shows a plot of the values for $Q(A,0)$ and $Q(A,1)$ that satisfy both of the above sets of equations for $\gamma = 0.9$.

We label the two greedy policies that result from the above observation-action values as π_0 and π_1, where π_0 greedily selects action 0 for observation A and policy π_1 always selects action 1 for observation A.

5.1.2.2 Convergence on greedy policies

Having established the existence of stationary policies for greedy agents, will convergence to these policies occur? As an example, let us continue to consider the POMDP

in figure 5.1, which has been learning with $\varepsilon = 0.01$ and $\gamma = 0.9$. After following policy ε-π_0 for a long period the observation-action values converge towards:

$$\begin{aligned} Q(A,0) &= 4.56, & Q(A,1) &= 5.17, \\ Q(B,0) &= 5.07, & Q(C,0) &= 5.63, \end{aligned}$$

Now consider that ε is set equal to zero at the point when the observation-action values have virtually converged to those given above. At the same instant the agent's policy is changed to that described by $\arg\max_a Q(o,a)$.

Given the current observation-action values, the new policy, $\arg\max_a Q(o,a)$, is equivalent to π_1. The observation-action pairs will converge towards values which are solutions to equations 5.20 – 5.22. Note $Q(A,0)$ will not be visited, nor will $Q(B,0)$. Thus the values of both remain unchanged. (One update of $Q(B,0)$ could occur but only if the agent were in the state corresponding to observation B at the point the policy became greedy.)

While $Q(A,0) < Q(A,1)$, these equations can be solved for $Q(A,1)$ to give:

$$\begin{aligned} Q(A,1) &= 0.1 + \gamma(-1 + \gamma Q(A,1)) \\ &= \frac{0.1 - \gamma}{1 - \gamma^2} \end{aligned}$$

Given that discount factor γ lies in the range $0 < \gamma < 1$, we can solve the above equations to give a range of values towards which $Q(A,1)$ will converge. The resulting range extends between $+0.1$ and $-\infty$. If we use $\gamma = 0.9$ then $Q(A,1) = -4.21$. Hence while $Q(A,1) > Q(A,0)$, $Q(A,1)$ will be reduced each time it is sampled. Given the initially unchanging value of $Q(A,0) = 4.56$ then $Q(A,1)$ will eventually fall below it. At the point that $Q(A,1) < Q(A,0)$ the agent's policy $\arg\max_a Q(o,a)$ becomes equivalent to π_0 above and the observation-action pairs will start to converge towards the values which are solutions to equations 5.16 – 5.19. Note $Q(A,1)$ will not be visited and thus its value is frozen at a value slightly below that of $Q(A,0)$, that is, slightly below 4.56.

While $Q(A,1) < Q(A,0)$ these equations can be solved for $Q(A,0)$ giving:

$$\begin{aligned} Q(A,0) &= \gamma^2(1+\gamma Q(A,0)) \\ &= \frac{\gamma^2}{1-\gamma^3} \end{aligned}$$

For the range $0 < \gamma < 1$ the above equation gives a range of values for $Q(A,0)$ between 0 and $+\infty$. Taking γ as 0.9 gives $Q(A,0) = 2.99$. $Q(A,0)$ will be adjusted towards this target value while $Q(A,0) > Q(A,1)$. At some point its value will become lower than $Q(A,1)$ and the policy will once more begin to behave like π_1, with the consequence that $Q(A,1)$ will again decrease in value until it is once more less than $Q(A,0)$. This flipping between policies will continue as the two observation-action values track each other down until the point when one of $Q(A,0)$ or $Q(A,1)$ cannot fall in value any further. At this point the policy stabilises. In the case of $\gamma = 0.9$, $Q(A,0)$ will stop falling in value once it has converged on 2.99, whilst $Q(A,1)$, which is tracking $Q(A,0)$ downwards, will fall past the floor of 2.99 as it is aiming to converge on -4.21. At this point, with $Q(A,0) = 2.99$ and $Q(A,1)$ marginally less than 2.99, the policy $\arg\max_a Q(o,a)$ stabilises and acts like π_0 above. The observation-action pair $Q(A,1)$ is no longer visited and is value remains frozen.

Which policy will dominate is partially dependent on the discount rate γ that is used. For $\gamma > 0.0917$, $Q(A,0)$ dominates over $Q(A,1)$ and from the above starting observation-action values the agent's policy will stabilise to resemble π_0. For $\gamma < 0.0917$ $Q(A,1)$ dominates over $Q(A,0)$ and the agent's policy will stabilise resembling π_1. This division makes sense if we consider that this low value of γ makes the agent very short sighted. Recall that the only place the agent can make a decision is when it sees observation A. When $\gamma < 0.0917$, an immediate reward of +0.1 is available by taking action 1 out of A, which is then followed by a heavily discounted penalty of less than -1×0.0917. This more valuable than the very heavily discounted reward, which will be less than $+1 \times 0.0917^2$, that is available for taking action 0.

The same analysis holds when the starting values are given by the observation-action values learnt following policy ε-π_1 with $\varepsilon = 0.01$.

$$\begin{aligned} Q(A,0) &= -2.93, & Q(A,1) &= -3.15, \\ Q(B,0) &= -3.25, & Q(C,0) &= -3.62, \end{aligned}$$

In this case, at the point that ε is set equal to zero, $Q(A,0)$ already dominates $Q(A,1)$, thus the policy $\arg\max_a Q(o,a)$ is equivalent to π_0. If we use a value of $\gamma = 0.9$ then following this policy increases the value of $Q(A,0)$ towards 2.99 and therefore no further policy transitions occur. The policy stabilises immediately to resemble π_0 and the observation-action pair $(A,1)$ is not visited.

The above analysis shows that if ε is set equal to zero after some period of exploration has been allowed, then, provided the policy is defined as $\arg\max_a Q(o,a)$, Q-learning, when applied to this policy, will stabilise on a single policy. The policy will resemble π_0 for $\gamma > 0.0917$ and π_1 for $\gamma < 0.0917$.

Figure 5.5 indicates that for $\gamma = 0.9$ there are two stable sets of observation-action values for a greedy agent corresponding to the two possible greedy policies; π_0 and π_1. In practice, when we have empirically observed greedy policies converge on this problem, they have always converged to the π_0 policy (for $\gamma = 0.9$). We believed that this is because ε is initially not zero, so the range of values that $Q(A,0)$ has converged towards when the agents starts acting greedily is always greater than the value of -4.21. For the policy to converge to π_1 with $\gamma = 0.9$, the value of $Q(A,0)$ needs to be less than -4.21 when exploration ceases in order to remain less than $Q(A,1)$.

We have empirically demonstrated convergence to stable greedy policies for another simple POMDP problem posed by Whitehead in chapter 4. In addition, in carrying out experimental work for this thesis, our general experience is that decaying ε to zero and then having the learning algorithm act greedily for a sustained period whilst maintaining a positive learning rate, α, has generally aided convergence to stable policies. We would be greatly interested in (i) a formal proof that stable greedy policies always exist for POMDPs or a sub-class of POMDPs, (ii) a proof of convergence to such greedy policies, including conditions required for convergence such as control of learning rate, (iii) the quality of such greedy policies, that is, for minimum cost to goal problems,

does it always converge on satisficing policies? We ourselves are as yet unable to offer such proofs.

5.1.3 Analogy with simulated annealing

Comparisons can be made between the effect seen here of a distinct change in the stable policy space and *phase transitions* which occur at *critical temperatures* in simulated anneal [Basu and Frazer, 1990; McFadden et al., 1993; Santos, 1995; Selman, 1995]. The similarities suggesting that statistical mechanics may offer a way of arriving at the proofs sought above. That similarities exist is not completely unexpected as an agent in following a given policy, but with some chance of selecting an exploratory action, is in effect performing *simulated annealing* over the task. To make the analog explicit consider the following. At anyone time the greedy policy which the agent is following is formed by selecting actions that minimises the cost, thus moving the agent towards a low "energy state" (if we equate cost and energy). Exploration, whether it be ε-greedy or Gibbs/Boltzmann action selection (as set out in equation 5.23), introduces some possibility of selecting a non-policy action which could possibly place the agent into a high energy state. Such actions allow the possibility of lifting the agent out of local minimum that the policy has caused it to become trapped in and thus allow it to find the global minimum (that is, the goal state). This is obviously a description of simulated annealing as proposed by Metropolis et al. [1953]. The above analogy between optimisation problems and simulated cooling of materials is not new, it being realised in 1983 by Kirkpatrick et al. [1983].

The description above is focused at the level of the agent's movements. What is of more interest is whether the *policy* learnt on a POMDP task can be said to under go simulated annealing. For temporal-difference reinforcement-learning methods the policy is formed based on the observation-action values, which are in turn updated based on the samples recorded by the agent. Thus the policy and exploration bias the samples used to update the policy. A similarity to simulated annealing can be seen in that (i) given a high level of exploration (hight value of ε or τ) the agent will sample a large

number of possible policies which is equivalent to a system switching between many varied states which is typically only possible given plenty of available energy, (ii) for a low level of exploration the agent will generally be limited to trying small variations on a largely fixed policy, equivalent to a system where only small variations in the structure are possible due to the limited energy available, and (iii) for no exploration the current policy can only change in a way that reduces the cost of the policy, equivalent to only allowing changes in the structure of a system which results in an overall decrease in energy. The simulated annealing analogy would therefore seem to hold, though possibly only for MDPs. The factors that complicate the policy update when applied to POMDPs is that stochastic policies can exist which have higher rewards (lower energy) than the reactive deterministic policies which the learning algorithm is attempting to learn. Section 6.1.1 and appendix I demonstrate the existence of such stochastic policies. When a degree of exploration is allowed any policy being followed by the agent is in part stochastic, allowing it to find these higher valued (lower energy) policies. The amount of stochasticity will even affect the value of these policies, for example it will effect the average time it takes to break a set of cyclic actions. So part of the "phase change" that occurs when exploration ceases on POMDPs is that stochastic policies which could be more valuable become unreachable and thus cease to be sampled. Assuming learning continues, the observation-action values that are visited will be updated to reflect only deterministic policy values. A second effect that occurs is that separate from stochastic policies becoming unreachable, parts of the observation-action space are also unreachable, that is they will no longer be visited while following the current policy. To continue the simulated annealing analogy this "freezes" certain observation-action values as they are no longer sampled. This is the other major "phase change" that occurs when exploration ceases, it is random exploration that allows all states in the observation-action space some probability of being updated. This freezing seems unlike any process which normally occurs in simulated annealing as some of the frozen values may not have reached their lowest energy levels, yet are somehow prohibited from changing. This would seem to indicate that the greedy phase of learning on POMDP models cannot straight forwardly be represented as simulated annealing, and thus careful consideration would be required as to the application of statistical

mechanics to formalise our approach.

The freezing of observation-action values in areas of the observation-action space that are no longer visited does not indicate a freezing of the policy. Policy updates continue with the advantage that it will now only consider deterministic policies. The disadvantage is that updates will only change the current policy in ways that are immediately advantageous, that is, lead to an immediate reduction in "energy", it will not be able to climb out of local minimum.

5.2 Comparison of Greedy Action Selection with Softmax

$$\Pr\{a\} = \frac{e^{Q(o,a)/\tau}}{\sum_{b=1}^{n} e^{Q(o,b)/\tau}} \tag{5.23}$$

The equation shown above, equation 5.23, is the Gibbs/Boltzmann distribution for action selection [Sutton and Barto, 1998, chp.2,p30] also known as softmax. $\Pr\{a\}$ is the probability of selecting action a given that the observation is o, and the observation-action values are given by $Q(o,x)$. The observation-action values are the current estimate of the value of taking action x when the observation is o. τ is referred to as the "temperature" and controls ratio of exploratory actions to exploitative (or policy) actions. High temperatures cause all actions to become nearly equiprobable in being selected, low temperatures cause actions associated with the highest state-action values to become the most probable to be selected. In the limit as $\tau \to 0$ action selection becomes greedy.

Perkins and Pendrith [2002] show that for continuous action selection functions such as Gibbs/Boltzmann distribution, stationary points exist in the policy space for Q-learning and SARSA. They also speculate that this result should extend to SARSA(λ). This is in contrast to discontinuous action selection strategies, such as ε-greedy, where, as illustrated in section 5.1.2, examples have been devised for which no stationary policies exists.

Perkins and Pendrith [2002], in showing the existence of stationary points for continuous action selection, does not offer any guarantees as to the quality of the policies at the action-value stationary points, nor does the paper give any guarantees that Q-learning or SARSA will actually converge to the stationary points.

In this section we compare results obtained on a POMDP using Gibbs/Boltzmann distribution for action selection with those obtained through use of ε-greedy exploration where ε is reduced eventually to zero and learning continues whilst the agent acts greedily. This latter strategy is suggested by the previous section (section 5.1.2.1).

Our aim in undertaking this comparison is to test if there are consistent significant differences between the policies learnt using these two different exploration approaches. The hope is that if any clear differences are identified, these may suggest which of the two approaches should be preferred, or which might suit different situations.

Using the modified form of Sutton's Grid World introduced in the previous chapter we ran a set of tests using Gibbs/Boltzmann distribution to examine how well it performs in comparison to results obtained for ε-greedy/greedy approach. Experiments are performed using both 1-step backup algorithms, that is, SARSA and Q-learning, and also SARSA(λ) which uses eligibility traces to back up rewards.

We perform this comparison by looking at; (i) the number of policies that converge to satisficing solutions, (ii) the quality of the satisficing solutions which the algorithms converge to and (iii) if the deterministic policies converged to are locally minimum policies (see section 5.2.1.4 for definition of locally minimum policies).

5.2.1 Experimental setup

5.2.1.1 Learning algorithms and parameters

We run two distinct sets of tests. The first set of tests are using the 1-step backup algorithms SARSA and Q-learning for which the theoretical results indicating the existence of stationary points hold [Perkins and Pendrith, 2002]. The second sets of tests

are using SARSA(λ) for which Perkins and Pendrith [2002] conjecture that a similar result should hold.

For all three learning algorithms listed below (Q-learning, SARSA and SARSA(λ)), a discount rate of $\gamma = 0.9$ was used, action values were initiated at zero for every observation and each independent run lasted for a total of one million action-learning steps.

As we have indicated previously in this chapter, the pattern of exploration is important for convergence on POMDP tasks. One idea that we are seeking to test is that the eventual curtailment of exploration helps with the learning of stable policies. We suspect that this effect may be equally important when using continuous action selection methods. The amount of exploration using a Gibbs/Boltzmann distribution is controlled by two factors: the relative value of the action values for each observation; and the temperature parameter τ. With a fixed value for τ, required for stationary observation-action values by the theoretical result of Perkins and Pendrith [2002], the amount of exploration will be dependent on the relative value of the action-values. The value of action-values in turn depend on the placement and value of rewards and penalties in the given task. To ensure that the amount of exploration eventually becomes small, the fixed value of τ will need to be set at such a level that the difference between the action values in each state is sufficient for one action to dominate over the others, that is, that the value of τ will need to be tuned to suit the rewards in the given task. To allow for this effect we tried the following range of τ values: 0.01; 0.03; 0.05; 0.1 and 0.3.

For the main set of experiments we use Sutton's Grid World as described previously in section 4.1.1.2. To further test the hypothesis that that τ needs to be tuned to suit the reward structure of the problem we introduce a second version of Sutton's Grid World: Sutton's Grid World B. In Sutton's Grid World B the zero valued reward on reaching the goal is replaced with a reward of $+10$. No other rewards are changed, with the agent receiving a penalty of -1 for each physical action that it tries to execute. This simple change alters the reward structure of the problem and will affect the action values learnt. If our above supposition is correct, the "tuned" value of τ for which performance is maximised will be changed by this simple modification. Experiments

with Sutton's Grid World B are only run using the learning algorithm SARSA(λ).

Q-learning and SARSA

For Q-learning and SARSA a range of learning rates (α) were tried; 0.02, 0.05, 0.1, 0.2, 0.3, 0.5, and 0.7. This range was based on the observation in section 4.2.2.2 that higher α values tended to encourage convergence to satisficing solutions for 1-step backup algorithms. Action selection methods based both on the Gibbs/Boltzmann distribution and ε-greedy were tried, with τ values as set out above and ε initiated at 0.2 and decaying linearly to zero after the 500,000 action-learning steps. For each combination of learning algorithms, values of α and values of τ or ε, twenty independent trials were made. All runs used Sutton's Grid World.

SARSA(λ)

We tested SARSA(λ) with replacement traces. The combinations of learning rate and eligibility trace values tried were $\alpha = 0.02$ $\lambda = 0.99$ and $\alpha = 0.05$ $\lambda = 0.9$. These values were found to give good results in section 4.3 and their use allows comparison between the results obtained here for the Gibbs/Boltzmann action selection method and those obtained previously using ε-greedy. For the Gibbs/Boltzmann method of action selection we use the range of values of τ set out above. Tests were carried out on both Sutton's Grid World and Sutton's Grid World B. One hundred independent runs were carried out for each combination of α, λ, τ and grid world.

5.2.1.2 Grid worlds

The main task uses Sutton's Grid World as described previously in section 4.1.1.2 and shown in figure 5.6. As mentioned above, we introduce a second variation on this problem, Sutton's Grid World B, where the zero valued reward on reaching the goal is replaced with a reward of $+10$. No other rewards are changed, that is, the agent still receives a penalty of -1 for each physical action that it attempts to execute which does not bring it immediately to the goal state.

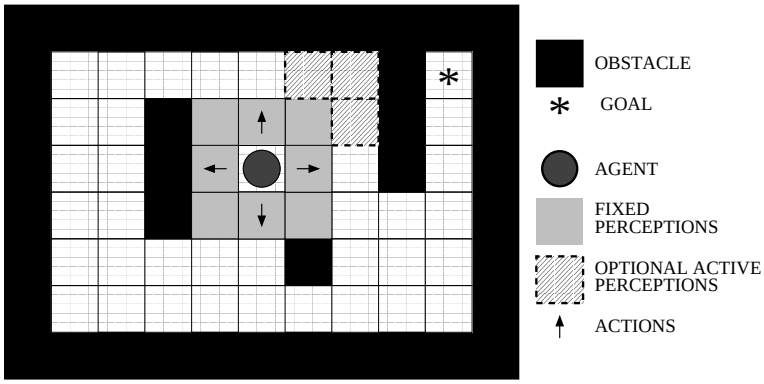


Figure 5.6: Sutton's Grid World and example agent. The greyed squares indicated the eight squares that make up the observation that the agent normally obtains from the world. The three hatched squares are additional locations that would be added to its observation on selecting the perceptual action look north east

5.2.1.3 Agents

As in section 4.3, we have two agents with differing perceptual abilities:

(i) **The Fixed Eight Adjacent Squares Agent.** Its observations consist of an eight-bit string which indicates whether the eight squares adjacent to its current location contain obstacles. This agent experiences state aliasing in multiple locations of Sutton's grid world (see section 4.1.1.2 for details).

(ii) **An Active Perception Agent.** This agent receives the same basic observation as the first, but has eight additional *perceptual actions* that it can select: look north, look north east, look east, look south east, look south, look south west, look west or look north west. On selecting one of these perceptual actions, its observation changes to an eleven-bit string by including information from three additional squares in the direction of its choosing, for example, having execute the action look north east its observation is enhanced by the three additional hatched squares shown in figure 5.6. It receives a penalty of -1 for selecting a perceptual action. Its observations revert to the eight adjacent squares after

selecting a physical action. The agent is at liberty to select a sequence of perceptual actions. It does not have to select a physical action after a perceptual action, or vice versa. In the case of selecting a sequence of perceptual actions, it never observes more than eleven locations at any one time: the basic eight adjacent squares and three additional squares based on the perceptual action last chosen. Nor does the agent build any memory of previous observations.

5.2.1.4 Evaluation criteria

Count of perceptual and physical actions

Each run was evaluated every ten thousand action steps and the total number of physical and perceptual actions required by the agent to reach the goal state from all possible start states[2] when following its policy deterministically was recorded. During evaluation, any trial that failed to reach the goal after a total of one thousand perceptual and physical actions was terminated. As there are forty six starting locations (excluding the goal) a policy that fails to reach the goal from all start states will have a maximum total of 46,000 actions. We use the count of physical actions, perceptual actions and their combined totals to compare the relative effectiveness of the policies learnt.

Policy classification

We can classify the policies at each evaluation point. They are classified based on whether or not they were satisficing, that is, reach the goal from all start states, and those that are satisficing are further classified based on the total number of physical actions required to reach the goal, see table 5.1. A more detailed description of these policy classifications can be found in section 4.1.1.4.

[2] For Sutton's Grid World and Sutton's Grid World B the agent is started in every location in the world that does not contain an obstacle. The agent with active perception always starts with its perception limited to the basic eight surrounding squares.

Goal Reached From All Starting Locations	Total Physical Actions	Policy Category
yes	404	Physically Optimal
yes	$405-415$	Better Than Optimal Reactive
yes	416	Optimal Reactive
yes	>416	Other Satisficing
no	-	Non-Satisficing

Table 5.1: Policy categories for Sutton's Grid World

Locally minimum policies

As demonstrated by Perkins [2001, 2002b], locally minimum policies can exist in POMDP problems. To test if convergence is occurring it is therefore not sufficient to look at the number of optimal policies. It is also necessary to consider whether the learning algorithm has converged to locally minimum policies. As the total number of possible policies for the version of Sutton's Grid World that we consider is 4^{30} for the agent with fixed perception (Fixed Eight Adjacent Squares Agent) and 12^{183} for the agent with active perception, it is not possible to enumerate every possible policy which corresponds to a local minimum, and then observe how many policies converge to these known local minima. Instead we choose to test each policy that has been learnt.

We test every policy at the end of one million action-learning steps to determine if it has converged to a local minimum. By locally minimum policy we mean a policy that cannot be improved making a *single* change to its observation-action associations. To test if policy π is a local minimum we generate every possible policy that differs from π by a single action choice. To do this we generate new policies by going through each observation that the agent has experienced and promoting in turn each of the actions which currently is not selected under π. We then test whether the total number of actions needed to reach the goal from all states for any one of these policies is less than the total number of actions needed to reach the goal for policy π. If no better policy that differs by a single action choice is found then policy π is taken as a locally

minimum policy. The optimal policy will of course fit this definition but have the additional property of being globally optimal, that is, it cannot be bettered by *any* other policy that it is possible to generate.

For the above learning algorithms Q-learning, SARSA and SARSA(λ), the policy is arrived at using $\arg\max_a Q(o,a)$. In the event that more than one action (a) shares the same maximum value for a given observation (o), then the selection between these tied actions occurs at random. An effect of this is that some policies may be stochastic. Where a policy π is stochastic, we test the likelihood of it being a local minimum by sampling the total number of actions needed to reach the goal for twenty evaluation runs of the same policy. We then look for a policy that differs by a single change to its observation-action associations, whose mean total number of actions over a course of twenty samples is less than the mean total number of actions for π. In addition, we required that these two means differ with 0.99 confidence using either Student's t-test (when the variance of the two samples are equal) or Student's t-test for samples with unequal variances. If we find a single policy that meets these criteria we conclude that the policy π is unlikely to be a local minimum.

Observation-action stationary points

Unfortunately we have been unable to devise a test that would identify if a policy has converged to a "stationary point" [Perkins and Pendrith, 2002]. Stationary points are not locally minimum policies: the definition of a stationary point is based on *expected* observation-action values and thus is a broader concept than that of locally minimum policies. A stationary point could, for example, represent a number of policies which a learning algorithm cycles between in a stochastic manner, but in such a way as to ensure that the long term average of the observation-action values remains stationary.

5.2.2 Results for Q-learning and SARSA

Results obtained for the 1-step learning algorithms are shown in figures 5.7 through to 5.10. We consider first the number of satisficing policies learnt by Q-learning after one

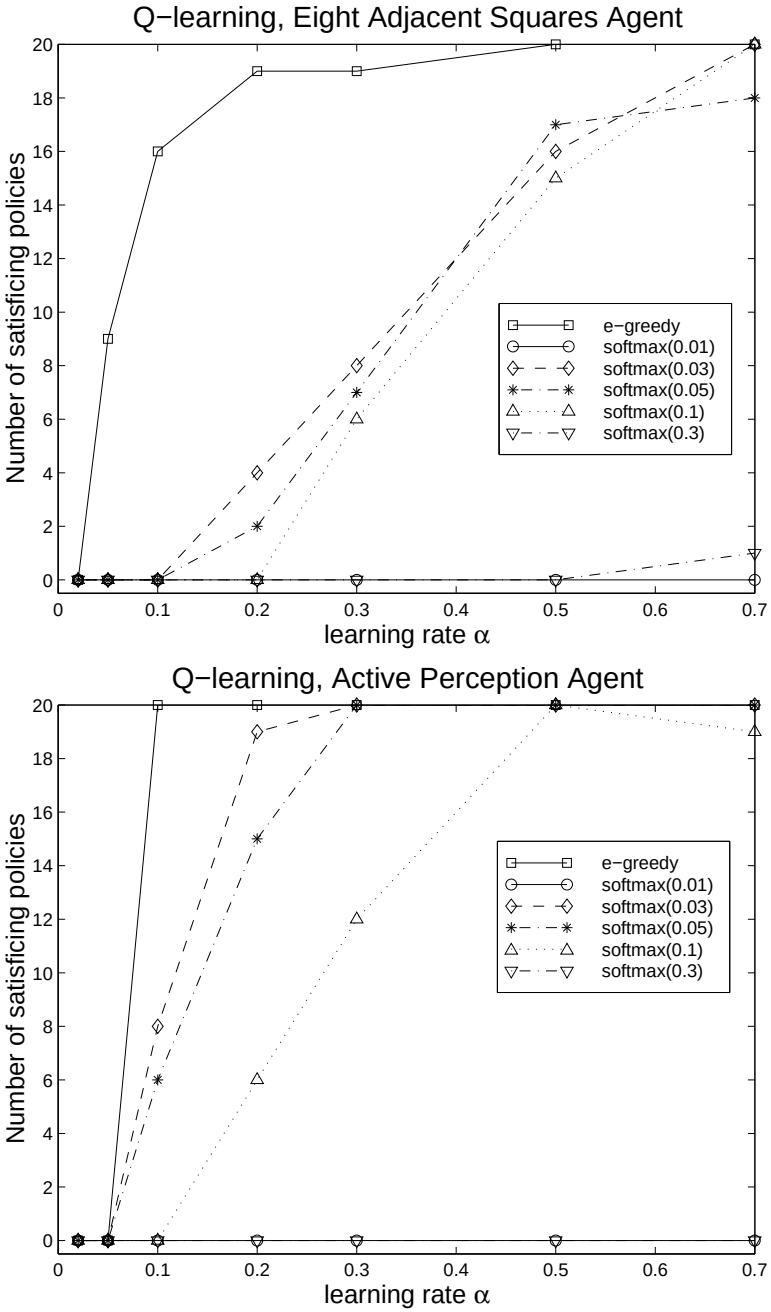
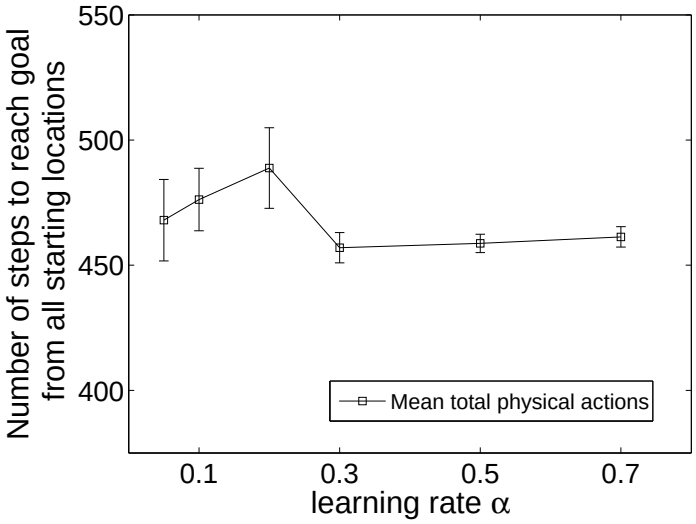


Figure 5.7: Number of satisficing policies after one million action-learning steps plotted against variation in the value of α. Number given is out of a total of twenty independent runs

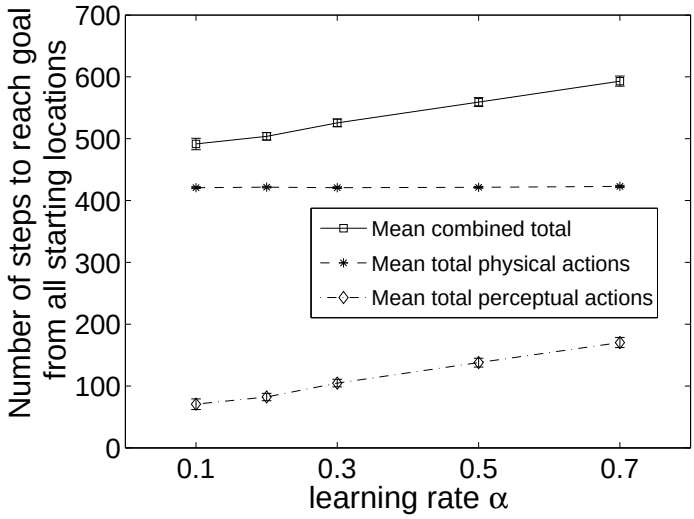
million action learning steps, see figure 5.7. When using Gibbs/Boltzmann distribution for action selection (softmax) no satisficing policies are learnt for either of the two lowest values of α, 0.02 and 0.05. A small number are learnt for $\alpha = 0.05$ when using the ε-greedy/greedy action selection but only for the Fixed Eight Adjacent Squares Agent (top plot). Around α values of 0.1 and 0.2, a number of policies converge to satisficing solutions, especially when the ε-greedy/greedy approach is used. The number of satisficing policies then continues to increase with increasing values of α until at the highest value tested, 0.7, nearly all policies converge to satisficing solutions for ε-greedy/greedy and Gibbs/Boltzmann distribution with τ between 0.03 to 0.1 inclusive. The lowest value of τ, 0.01 never learns any satisfactory policies over the range of α values tried. The highest value of τ also performs poorly as it hardly learns any satisfactory policies. Plots for SARSA, which are not shown, are very similar to those described above for Q-learning.

These results complement those from section 4.1.2.2 in showing that reinforcement learning algorithms using 1-step backup can learn satisficing policies for POMDPs. These results are all the more significant as Sutton's Grid World is a more complex task than Whitehead's simple 1-D example. The complexity of the task is further increased when the active perception agent is used, but these cases also show positive results.

Although increasing the learning rate α appears to increase the probability of learning satisficing policies, one would expect some trade off in the quality of the policies. To examine this we look at the quality of the satisficing solutions that have been learnt after one million action-learning steps. We measure the quality of a policy in terms of the total steps taken to reach the goal location from every start location. This we plot against against variation of of the learning rate α. Figures 5.8(a) and 5.8(b) are plots, one per agent type, of the average total steps for all the satisficing policies learnt after one million action-learning steps against variation of of the learning rate α. Results are averaged across both SARSA and Q-learning and all the action selection methods tried. For the Fixed Eight Adjacent Squares Agent, figure 5.8(a), we plot the average total physical actions. For the active perception agent, figure 5.8(b), we plot the average total physical actions, average total perceptual actions and the average combined total



(a) Mean total physical action steps for Fixed Eight Adjacent Squares Agent



(b) Mean total physical, perceptual and combined action steps for Active Perception Agent

Figure 5.8: Mean total action steps for satisficing policies versus learning rate α. Action steps averaged over all combinations of Q-learning, SARSA and action selection strategies/parameters. Error bars indicate 95% confidence intervals

of both physical and perceptual actions.

Examining these plots, possible signs of policy degradation only occur with the active perception agent, figure 5.8(b), where the average combined total actions increase with alpha for both ε-greedy/greedy and Gibbs/Boltzmann distribution action selection. This increase can be seen to be driven by an increase in the average number of perceptual actions that each policy uses to reach the goal as α increases. In contrast the average number of physical actions remains level.

The plot for the Fixed Eight Adjacent Squares Agent, figure 5.8(a), shows neither a consistent increase or decrease in the number of physical steps as α is increased. The peak which is observed around $\alpha = 0.2$ is associated with a small number of satisficing policies, hence the large confidence interval bars, so it is quite possibly a random effect. Overall it appears that the average number of physical actions for this agent, which has no access to perceptual actions, remains roughly level.

Appendix M shows the same information but broken down by learning algorithm and choice of action selection method. The plots presented there indicate that there is no significant difference in policy degradation between SARSA and Q-learning or between the action selection methods tried.

Figures 5.9 and 5.10 show plots of mean total number of steps taken to reach the goal location when policies were evaluated, against the number of action-learning steps, that is, the amount of time the agent has been learning. Plots are shown for the most successful of the α values tried, $\alpha = 0.7$. Plots are only shown for Q-learning as the results obtained for SARSA are virtually identical. Averages values are obtained by averaging across the policies learnt for the twenty independent runs that were undertaken for each combination of agent and learning parameters. When the policy is evaluated any run that fails to reach the goal after 1,000 action steps is terminated. Given that there are a total of 46 start locations (excluding the goal location), a policy that fails to reach the goal from all start locations will take a maximum of 46,000 steps.

In figures 5.9 and 5.10 the top plots show curves for both Gibbs/Boltzmann distribution and ε-greedy/greedy action selection, the former for values of τ of 0.03, 0.05 and 0.1.

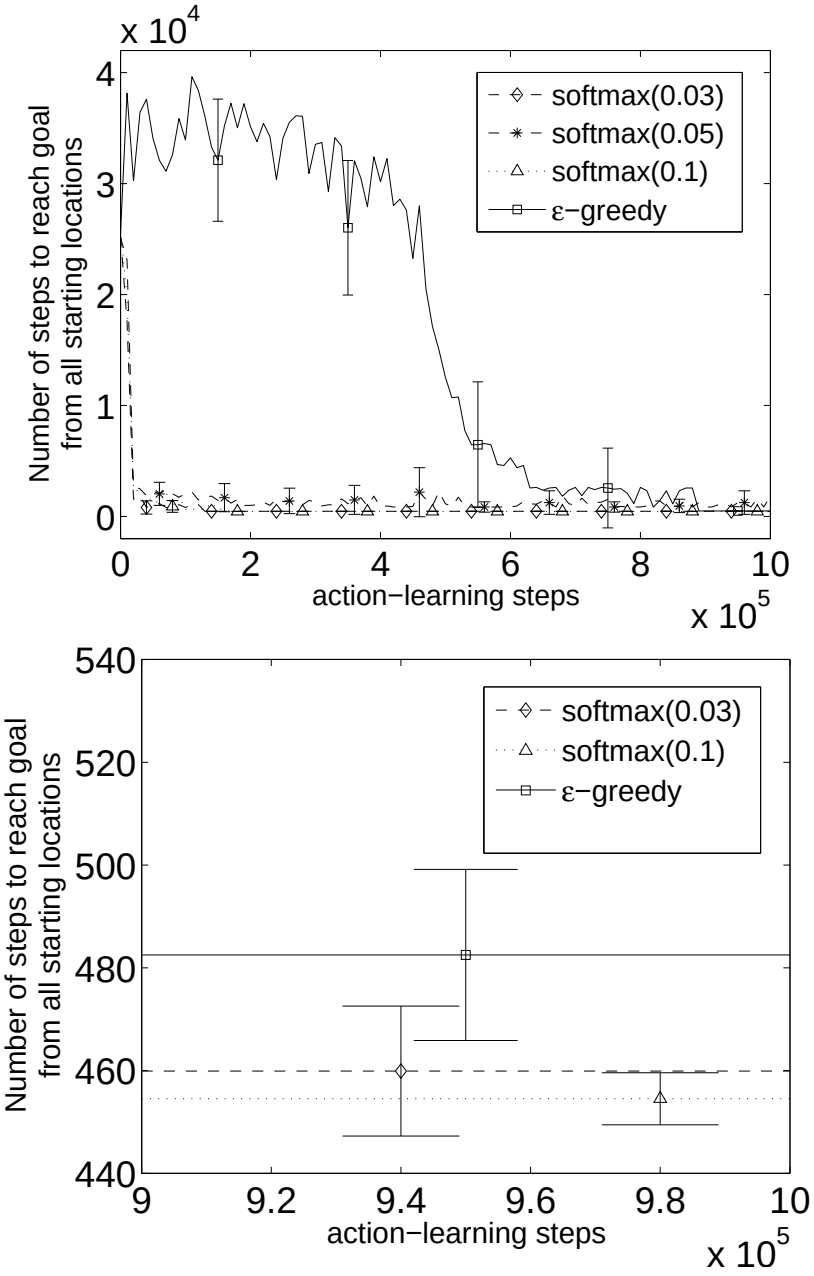


Figure 5.9: Mean total number of steps to reach the goal when evaluated, averaged over all twenty policies learnt for Q-learning with Eight Adjacent Squares agent and $\alpha = 0.7$. This is plotted against the action-learning steps that have passed, that is, the amount of time the agent has been learning. Error bars indicate 95% confidence intervals. The lower plot shows a closeup of the tail of the plot above. The curve for softmax(0.05) is deliberately missing from the lower plot so that the separation between the other curves can be seen; softmax(0.5) lies someway off the top of the lower plot

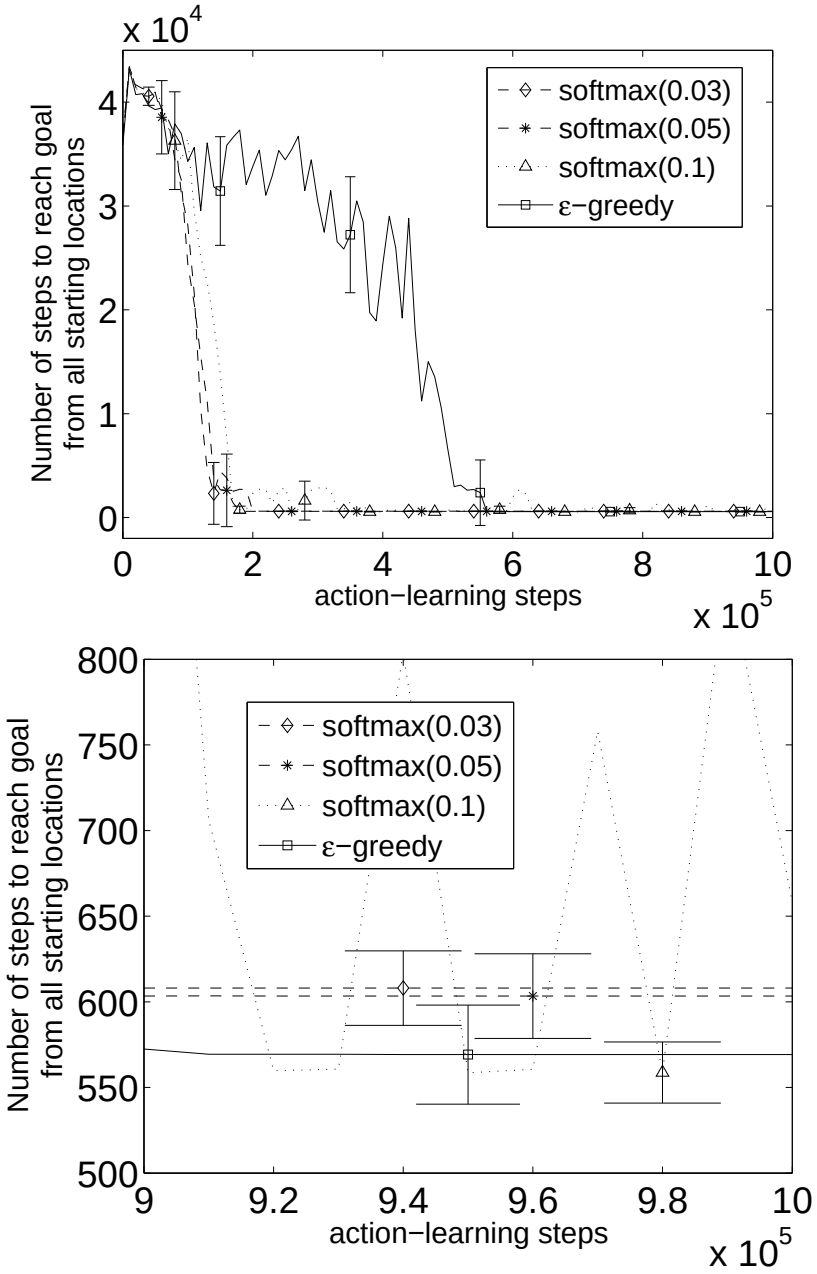


Figure 5.10: Mean total number of steps to reach the goal when evaluated, averaged over all policies learnt for Q-learning with Active Perception agent and $\alpha = 0.7$. This is plotted against the action-learning steps that have passed, that is, the amount of time the agent has been learning. Error bars indicate 95% confidence intervals. The lower plot shows a closeup of the tail of the plot above

Policies learnt with τ values 0.01 and 0.3 were generally not satisficing so have not been plotted. The lower plots focus on the tail end of each of the above plots and thus show more clearly the average number of steps the policy converges to for each set of parameters. The upper plots show clearly the relative rate with which each combination of parameters converges. It can be seen from these plots that for those values of τ for which Gibbs/Boltzmann action selection converges, it converges far more quickly than ε-greedy/greedy action selection. It appears that the convergence of ε-greedy/greedy policies to satisficing solutions depends entirely on the falling value of ε which reaches zero at $500,000$ action-learning steps. This observation is in line with those for convergence of 1-step reinforcement learning using ε-greedy/greedy action selection on Whitehead's simple 1-D example as presented in section 4.1.2.2. The schedule of values used for ε could be changed to speed convergence and presumably achieve as rapidly converging results as those achieved using Gibbs/Boltzmann action selection with fixed values of τ.

The lower plots in figures 5.9 and 5.10 show a closeup of the tails of each of the above curves. It is very interesting to observe the stability of the tails of these plots with both the average number of steps and 95% confidence interval remaining largely static once all policies have converged to satisficing solutions. Examination of individual policies confirms that this stability of the average value and confidence interval is reflected by the stability of individual policies. Once a policy reaches a satisficing solution then the total number of steps taken by the policy (every time it is evaluated) remains stable. It appears that, for this particular grid world problem, satisficing policies, once reached, remain stable under 1-step reinforcement learning whichever action selection method is used.

5.2.2.1 Locally minimum policies

Policies that are satisficing are not necessarily locally minimum policies or, for that matter, globally minimum policies. The reverse is also true in that a non-satisficing policy could be a locally minimum policy. Our aim in this section is to look for comparative advantages between using Gibbs/Boltzmann distribution and ε-greedy/greedy.

Thus we consider it useful to test if any of the above policies have converged to local (or possibly) global minimum solutions.

We tested all 3,360 policies generated above. This number represents twenty independent runs of every combination of α values, variation in action selection methods (including values of τ), agents and learning algorithms. Out of these policies, only three had clearly converged to local minimum solutions. For a further two policies, which were stochastic, we were unable to clearly establish if they were local minimum policies or not. In both cases the agent failed to locate the goal location during learning and hence its policy resembled a random walk through the environment. The amount of variation in these random walks when evaluated was such that it was difficult to find a single change in the observation-action table that resulted in a new policy that over 20 samples was statistically better, even if a more generous confidence interval of say 0.95 was allowed for the t-test. Given their resemblance to random walks we suspect that neither was a local minimum. Overall 57 out of the 3,360 policies were found to be stochastic.

The three locally minimum policies were also satisficing policies. All three were learnt using the Eight Adjacent Squares agent, and, since we know that the optimal reactive solution for this agent is a total of 416 physical action steps [Littman, 1994], we are able to state that none of the three were globally minimum policies. One locally minimum policy took a total of 444 physical action steps to reach the goal location from all start locations. The other two both took a total of 428 action steps to reach the goal location. All three were learnt using SARSA. The first was learnt using an α value of 0.5 with ε-greedy/greedy action selection. The second was also learnt with an α value of 0.5, using Gibbs/Boltzmann distribution for action selection with value of τ of 0.1. The third was learnt with an α value of 0.7, using Gibbs/Boltzmann distribution for action selection with value of τ of 0.05. No locally minimum policies were found for the Active Perception agent.

5.2.3 Conclusions for Q-learning and SARSA

The principal aim of the above results was to compare performance of Gibbs/Boltzmann distribution for action selection with ε-greedy/greedy action selection. These results relate to 1-step algorithms Q-learning and SARSA only. Results for SARSA(λ) are presented in section 5.2.5 below.

Looking at the percentage of satisficing policies learnt, the performance of the ε-greedy/greedy approach is better than Gibbs/Boltzmann distribution for all of the values of α which were tested. This suggests that for 1-step backup reinforcement learning algorithms the ε-greedy/greedy approach is more robust against variation in the learning rate α. This is an important feature as tuning the learning rate to suit a task can prove time consuming.

The results bear out our ideas with regard to tuning τ to match the task when using Gibbs/Boltzmann distribution for action selection. The best performance, in terms of number of satisficing policies, is achieved with values of τ around 0.03 and 0.05. Slightly worse performance is obtained for $\tau = 0.1$, and the both the lowest and highest values tried, 0.01 and 0.3 perform very poorly. This can be seen as a disadvantage of using Gibbs/Boltzmann distribution, as the required value of τ will probably not be known in advance. A decreasing schedule of τ could be used but this would not be covered by the theoretical results of Perkins and Pendrith [2002] nor would it be easy to determine the rate at which the schedule should be varied. It is possible that work on simulated annealing can be of use in suggesting a schedule of τ, perhaps through monitoring the "free energy" or "heat capacity" of the system [Santos, 1995]. Section 5.1.3 discusses the possible parallels between simulated-annealing phase-transitions and the changes that occur as the level of exploration varies on POMDP tasks.

The need to tune the value of τ suggests that the Gibbs/Boltzmann distribution may ultimately rely on the same mechanism as the ε-greedy/greedy algorithms for the formation of stable policies on POMDPs. The common mechanism is that there needs to be a near complete absence of exploration in order to allow stable policies to form for

POMDPs.[3]

In terms of speed of convergence, when the Gibbs/Boltzmann distribution approach does converge, it is much quicker than the ε-greedy/greedy approach. It should however be noted that the schedule of values used for ε could be changed to speed convergence, and could possibly then achieve as rapidly converging results as those achieved using Gibbs/Boltzmann action selection with fixed values of τ. An issue would then be tuning the schedule of ε as opposed to tuning the fixed value of τ. Whether setting this schedule becomes critical or not will depend on how much leeway there is in the length of time allowed for learning, that is, for how many steps we can allow for exploration, and whether this is sufficient time to explore the observation-action space.

The three local minima found are insufficient to reach any conclusion as to the comparative effectiveness of ε-greedy and Gibbs/Boltzmann distribution in learning locally optimal policies. The one conclusion that can be drawn from these results is that the 1-step backup reinforcement learning algorithms, although they have been shown to learn satisficing policies, in general do not appear to learn globally, or even locally, minimum policies.

Overall it is difficult to recommend one action selection approach over the other based on these results. The main advantage of the Gibbs/Boltzmann distribution approach is the speed of convergence when convergence occurs. The ε-greedy/greedy approach's main advantage is the avoidance of the need to precisely tune a parameter such as τ to suit the task, though there probably needs to be a degree of generosity in the schedule for ε. The ε-greedy/greedy approach also demonstrates a better level of robustness to variation in learning rate.

An additional result is the further demonstration of the ability of 1-step learning algorithms to converge on stable solutions. This is counter to Whitehead [1992, p.78]'s argument that Q-learning (or any 1-step backup algorithm) is unable to converge on stable solutions in partially observable environments, though in fairness it appears that the probability of exploration has to be virtually zero for this to be true – witness the

[3] N.B. In making this statement we are only considering reactive policies learnt using simple temporal difference reinforcement learning techniques.

oscillation of Gibbs/Boltzmann distribution for $\tau = 0.1$ when using the Active Perception Agent, figure 5.10 .

5.2.4 Future work for Q-learning and SARSA

A point of interesting speculation that arises from these results relates to the convergence of ε-greedy/greedy to satisficing policies as ε approaches zero. The apparent improvement in policies as ε decreases might reflect the fact that as ε approaches zero the evaluation and learning conditions become more similar. In evaluating policies we always make the assumption that the policy is to be followed greedily. It may be that policies that are deemed non-satisficing when evaluated in this way, are actually well adapted to the conditions under which they are being learnt. In other words, they have adapted to the level of exploration that they experience at that point in time, and are not well suited to being evaluated greedily. This mismatch may contribute to the apparently poor results when the action learning steps are less than 500,000 (see figures 5.9 and 5.10). Further experimental work is required to test this possibility.

5.2.5 Results for SARSA(λ)

For SARSA(λ), plots showing the categorisation of policies versus action-learning steps are shown in figures 5.11 – 5.18. These plots are vertical stacked bar graphs with the number of policies falling into each different category represented by the height of the differently shaded bars. Policies are evaluated, categorised and plotted every ten thousand action learning steps. This allows the change in the number of policies falling into each category as learning progresses to be visualised. Section 4.1.1.4 provides an explanation of the policies categories used, which are also summarised in table 4.1.

Plots are show for each combination of parameters, agents and version of Sutton's Grid World tried. The plots for Sutton's Grid World, figures 5.11 – 5.14, also include plots for ε-greedy action selection from section 4.3. The latter are reproduced here to allow

easy comparison between the categorisation of policies learnt using Gibbs/Boltzmann distribution and those learnt using ε-greedy/greedy action selection.

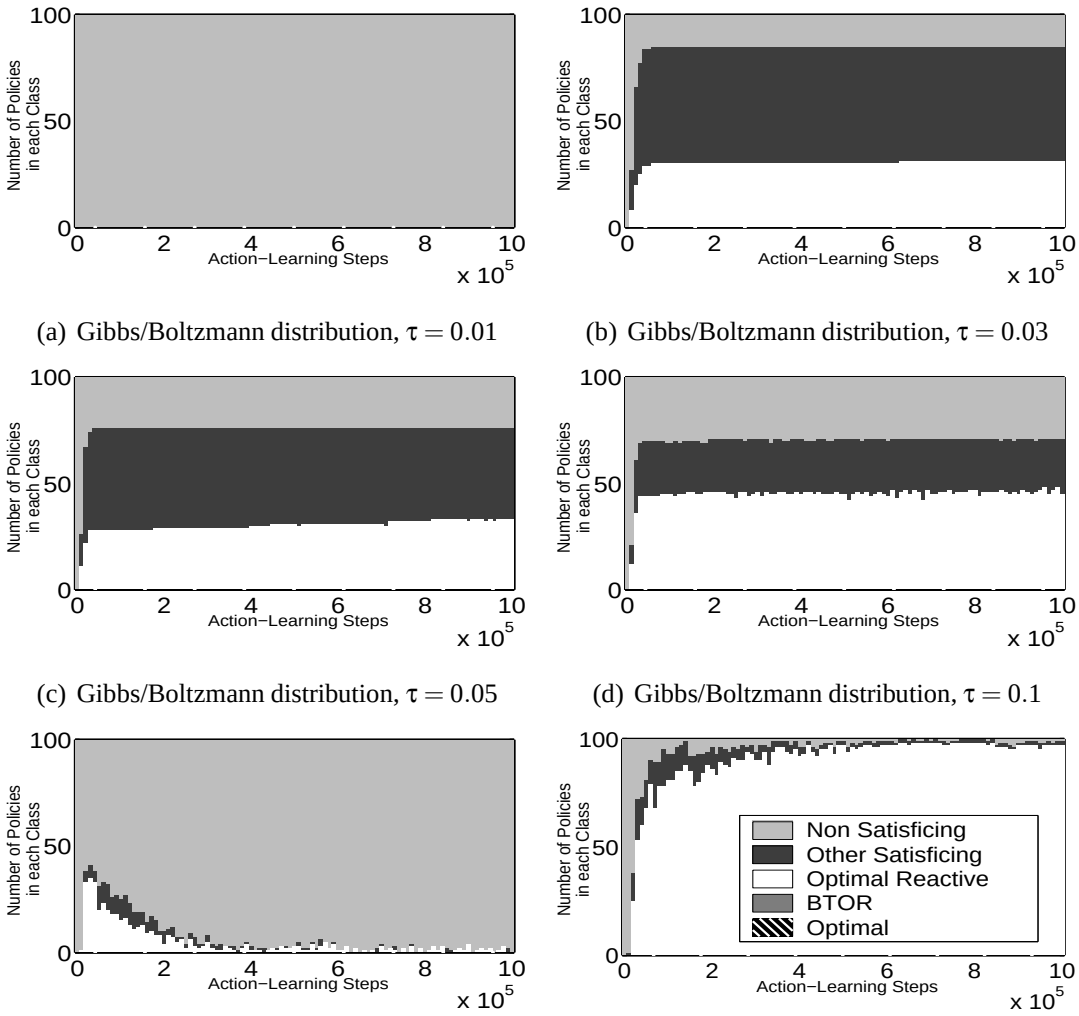
5.2.5.1 Sutton's grid world using eight adjacent squares agent

Figures 5.11 and 5.12 show results for Sutton's Grid World using Eight Adjacent Squares Agent. Subplots (a) – (e) are where Gibbs/Boltzmann distribution is used for action selection using τ values 0.01 through to 0.3. Subplot (f) shows results for ε-greedy/greedy.

The most obvious result from these two sets of plots is that the ε-greedy/greedy approach converges to far more satisficing policies than the Gibbs/Boltzmann distribution for any of the values of τ tried. The ε-greedy/greedy approach achieves close to 100% satisficing policies where the best obtained by Gibbs/Boltzmann distribution is around 80%. In addition, ε-greedy/greedy achieves a very high number of optimal reactive policies, around 90%, compared to slightly less than 50% for Gibbs/Boltzmann distribution.

Where policies converge using Gibbs/Boltzmann distribution it is more rapid than that for ε-greedy/greedy, parallelling the results for SARSA and Q-learning (see section 5.2.2 above). Convergence of ε-greedy/greedy is much more rapid using SARSA(λ) compared to SARSA and Q-learning above. As above ε is decayed to zero by the $500{,}000^{th}$ action learning step but with SARSA(λ) policy convergence occurs before this point is reached.

It is possible to observe from these two plots that the Gibbs/Boltzmann distribution performs best for τ values around 0.03 (though whether the peak lies between 0.03 and 0.05 or 0.01 and 0.03 cannot be determined from these results). We discuss this observation further below in section 5.2.5.3.



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

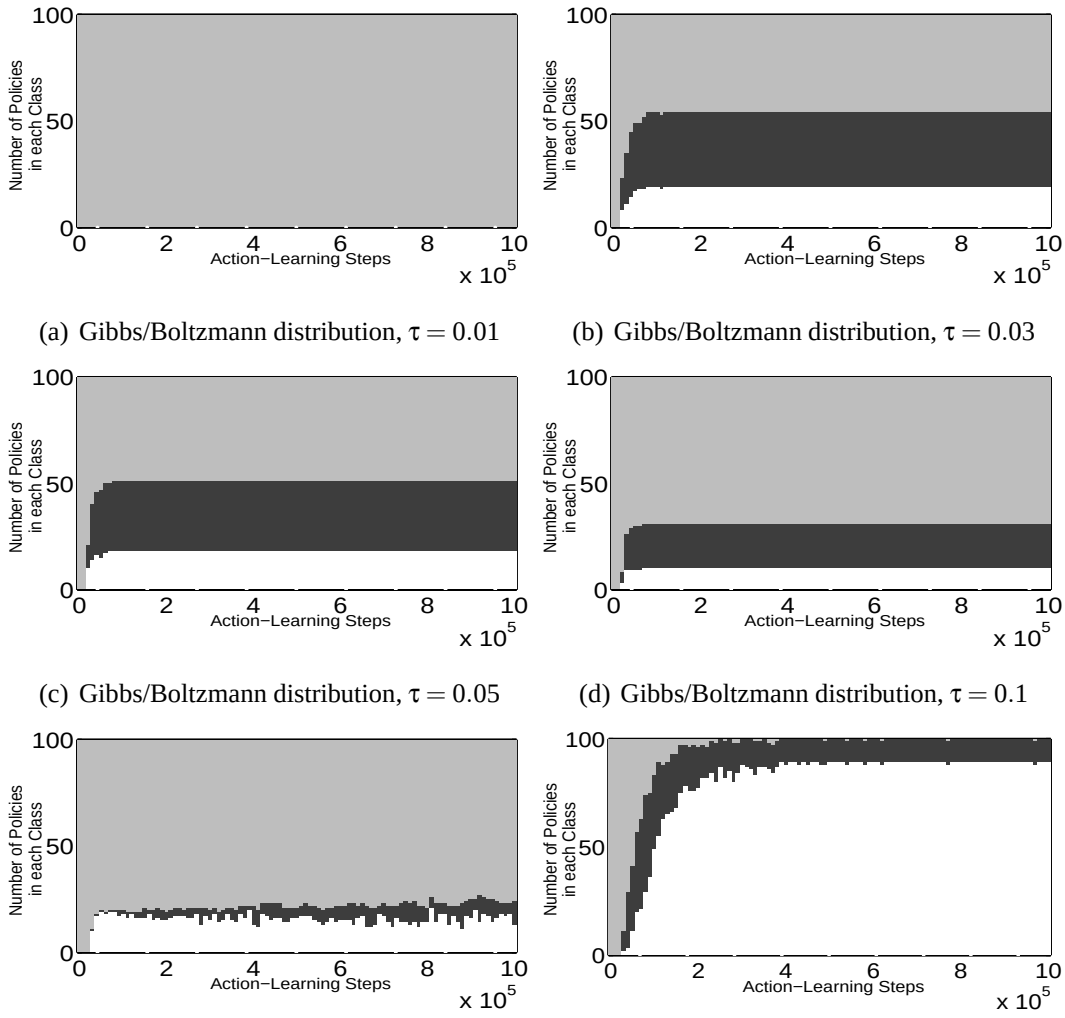
(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

(f) ε-greedy

Figure 5.11: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World using Eight Adjacent Squares Agent with $\lambda = 0.9$ and $\alpha = 0.05$. Plots 5.11(a)-5.11(e) show results when Gibbs/Boltzmann distribution is used for action selection with differing fixed temperatures τ. Plot 5.11(f) shows results when ε-greedy is used for action selection with ε initially equal to 0.2 and decaying linearly to zero by the 500,000th action-learning step. The plot for ε-greedy is reproduced from section 4.3



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

(f) ε-greedy

Figure 5.12: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World using Eight Adjacent Squares Agent with $\lambda = 0.99$ and $\alpha = 0.02$. Plots 5.12(a)-5.12(e) show results when Gibbs/Boltzmann distribution is used for action selection with differing fixed temperatures τ. Plot 5.12(f) shows results when ε-greedy is used for action selection with ε initially equal to 0.2 and decaying linearly to zero by the 500,000th action-learning step. The plot for ε-greedy is reproduced from section 4.3

5.2.5.2 Sutton's grid world using the active perception agent

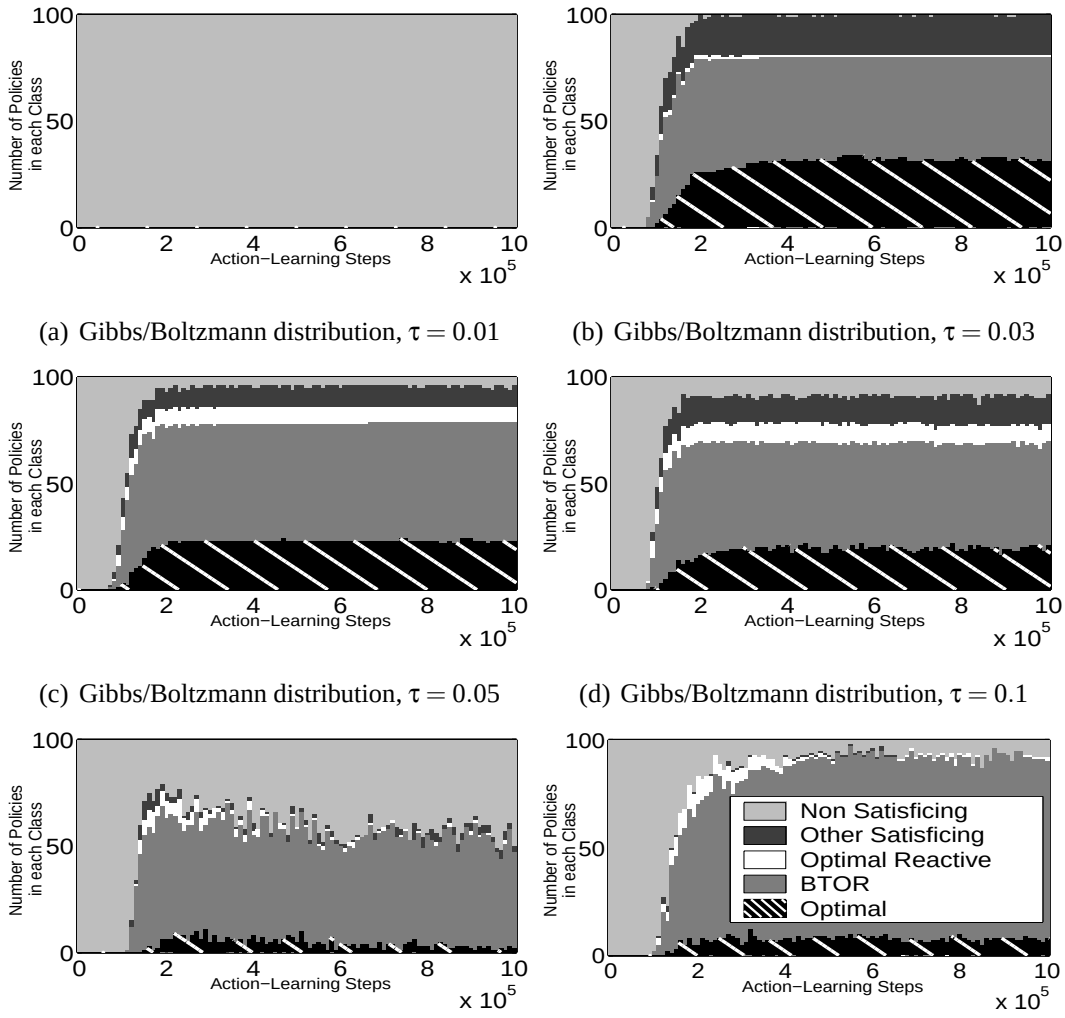
Figures 5.13 and 5.14 are for Sutton's Grid World using the Active Perception Agent. The use of this agent leaves the reward structure of the problem largely unchanged; each action executed (both perceptual and physical) costs -1 except for actions that result in it immediately arriving at the goal location, which attracts a reward of 0. The Active Perception Agent's introduction changes the grid world problem by increasing the range of actions available and the range of observations that are possible. This change increases the observation-action space that can be explored and increases the number and variety of satisficing policies that exist.

Gibbs/Boltzmann distribution for action selection performs better on this problem in terms of the number of satisficing policies learnt than it did for the Eight Adjacent Squares Agent in the previous two sets of results. It achieves 100% satisficing policies for parameter values $\lambda = 0.9$, $\alpha = 0.05$ and $\tau = 0.03$, figure 5.13(b) and 90% for parameter values $\lambda = 0.99$, $\alpha = 0.02$ and $\tau = 0.03$, figure 5.14(b). This performance is better than or comparable to the performance of the ε-greedy/greedy approach, figures 5.13(f) and 5.14(f) respectively.

The speed of convergence to satisficing policies for Gibbs/Boltzmann distribution and ε-greedy/greedy are very similar. The Gibbs/Boltzmann distribution approach is noticeably slower on this problem than it was for the Eight Adjacent Squares Agent. This is presumably due to the amount of time spent exploring the expanded observation-action space.

In terms of the classification of policies based on the number of physical actions taken to reach the goal state, there is insufficient consistency between the two sets of plots (figures 5.13 – 5.14) on which to draw conclusions. Both of the exploration approaches learn a similarly classified range of policies, the bulk of these policies being classified as optimal or better than optimal reactive policies.

In terms of variation with the value of τ, as with the Eight Adjacent Squares Agent above, we see evidence that the performance of the Gibbs/Boltzmann distribution



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

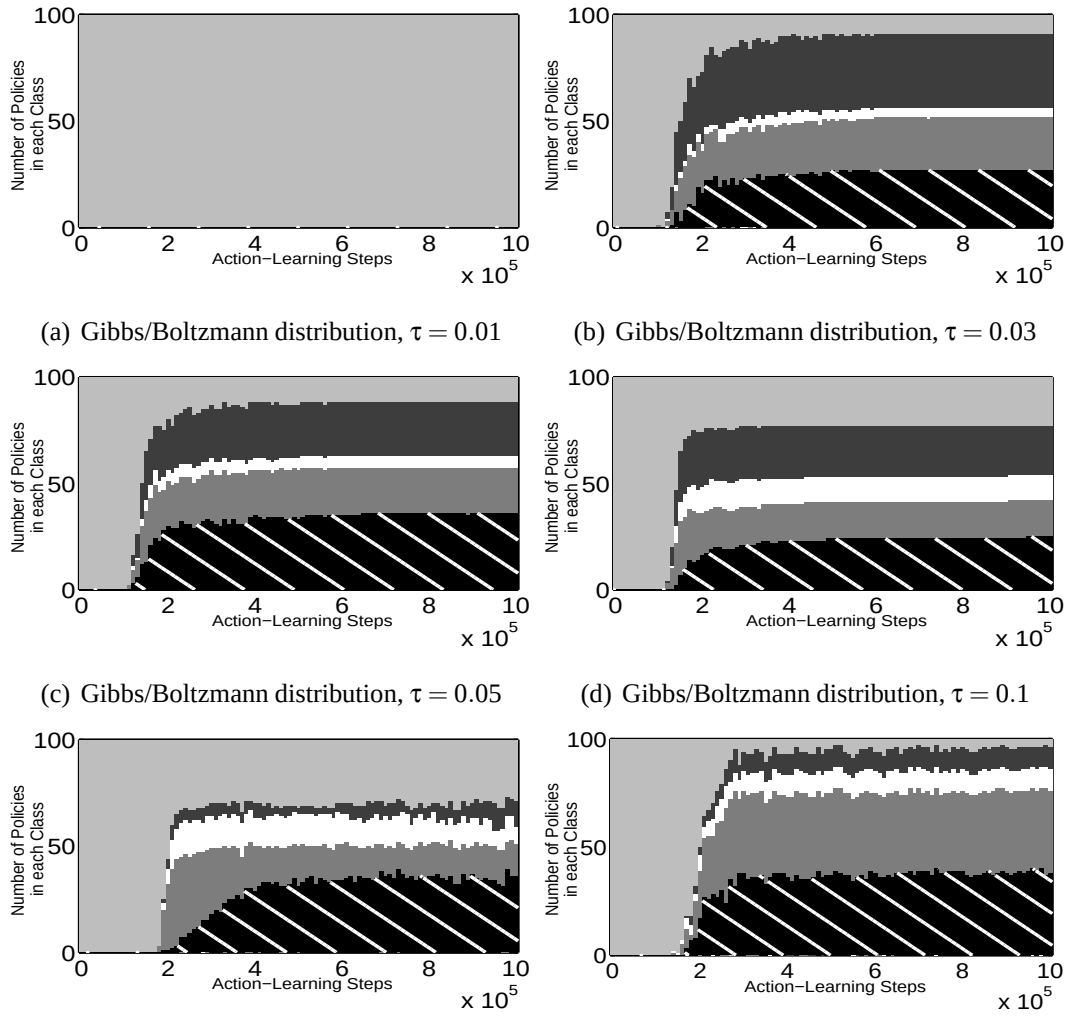
(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

(f) ε-greedy

Figure 5.13: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World using Active Perception Agent with $\lambda = 0.9$ and $\alpha = 0.05$. Plots 5.13(a)-5.13(e) show results when Gibbs/Boltzmann distribution is used for action selection with a differing fixed temperatures τ. Plot 5.13(f) shows results when ε-greedy is used for action selection with ε initially equal to 0.2 and decaying linearly to zero by the 500,000th action-learning step. The plot for ε-greedy is reproduced from section 4.3



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

(f) ε-greedy

Figure 5.14: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World using Active Perception Agent with $\lambda = 0.99$ and $\alpha = 0.02$. Plots 5.14(a)-5.14(e) show results when Gibbs/Boltzmann distribution is used for action selection with differing fixed temperatures τ. Plot 5.14(f) shows results when ε-greedy is used for action selection with ε initially equal to 0.2 and decaying linearly to zero by the 500,000th action-learning step. The plot for ε-greedy is reproduced from section 4.3

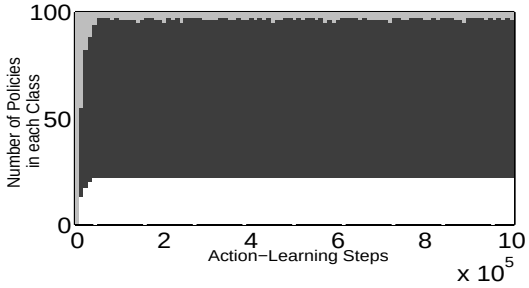
peaks somewhere around $\tau = 0.03$, with the number of satisficing solutions falling off for the two extreme values tried: $\tau = 0.01$ and $\tau = 0.3$. We discuss this further below (section 5.2.5.3).

5.2.5.3 Tuning τ to match the task

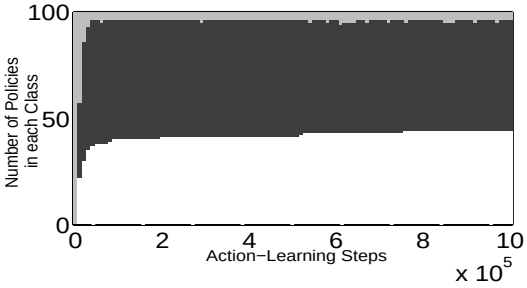
Figures 5.15 – 5.18 present the classification of policies learnt over time for Sutton's Grid World B. As before the first two sets of plots are for Eight Adjacent Squares Agent and the latter two sets are for the Active Perception Agent. No plots for the ε-greedy/greedy approach are shown as this approach was not run on Sutton's Grid World B. The aim of these runs was not to compare ε-greedy/greedy and Gibbs/Boltzmann approaches but instead to examine the effect of a change in the reward structure on learning with different values of τ.

The previous plots of policy categorisation, figures 5.11 — 5.14, demonstrate a clear relationship between the fixed temperature τ used with the Gibbs/Boltzmann distribution and the number of runs that converge to satisficing policies. For the range of parameter values tried and for both types of agent a peak in the number of satisficing policies occurs around $\tau = 0.03$. $\tau = 0.01$ produces no satisficing policies and the number of satisficing policies declines steadily as the temperature is raised above $\tau = 0.03$. This dependence on temperature lends support to our hypothesis that the fixed temperature τ needs to be tuned to the reward structure of the task.

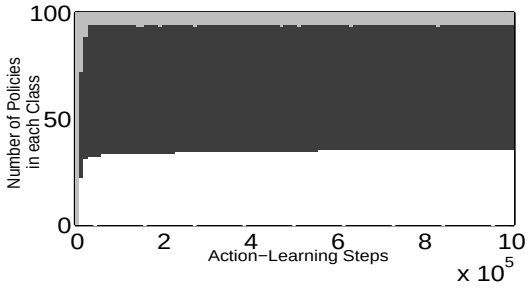
Examining the results for Sutton's Grid World B, figures 5.15 — 5.18, lends further support to our hypothesis. The change in the goal reward from zero to plus ten significantly alters the results obtained over the same range of temperatures. Focusing on the number of satisficing policies learnt, there is a much broader range of values for τ which give reasonable results on Sutton's Grid World B, compared to the single more tightly focused peak (around $\tau = 0.03$) that occurs with our original Sutton's Grid World task.



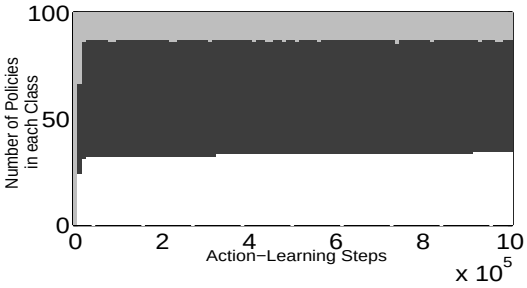
(a) Gibbs/Boltzmann distribution, $\tau = 0.01$



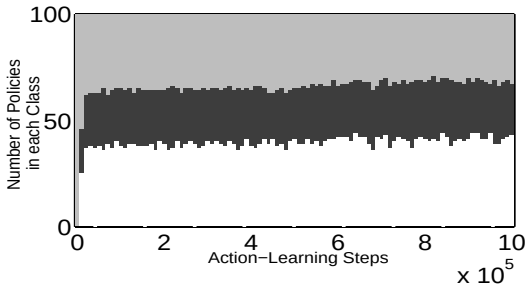
(b) Gibbs/Boltzmann distribution, $\tau = 0.03$



(c) Gibbs/Boltzmann distribution, $\tau = 0.05$



(d) Gibbs/Boltzmann distribution, $\tau = 0.1$



(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

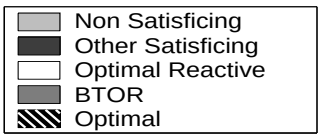
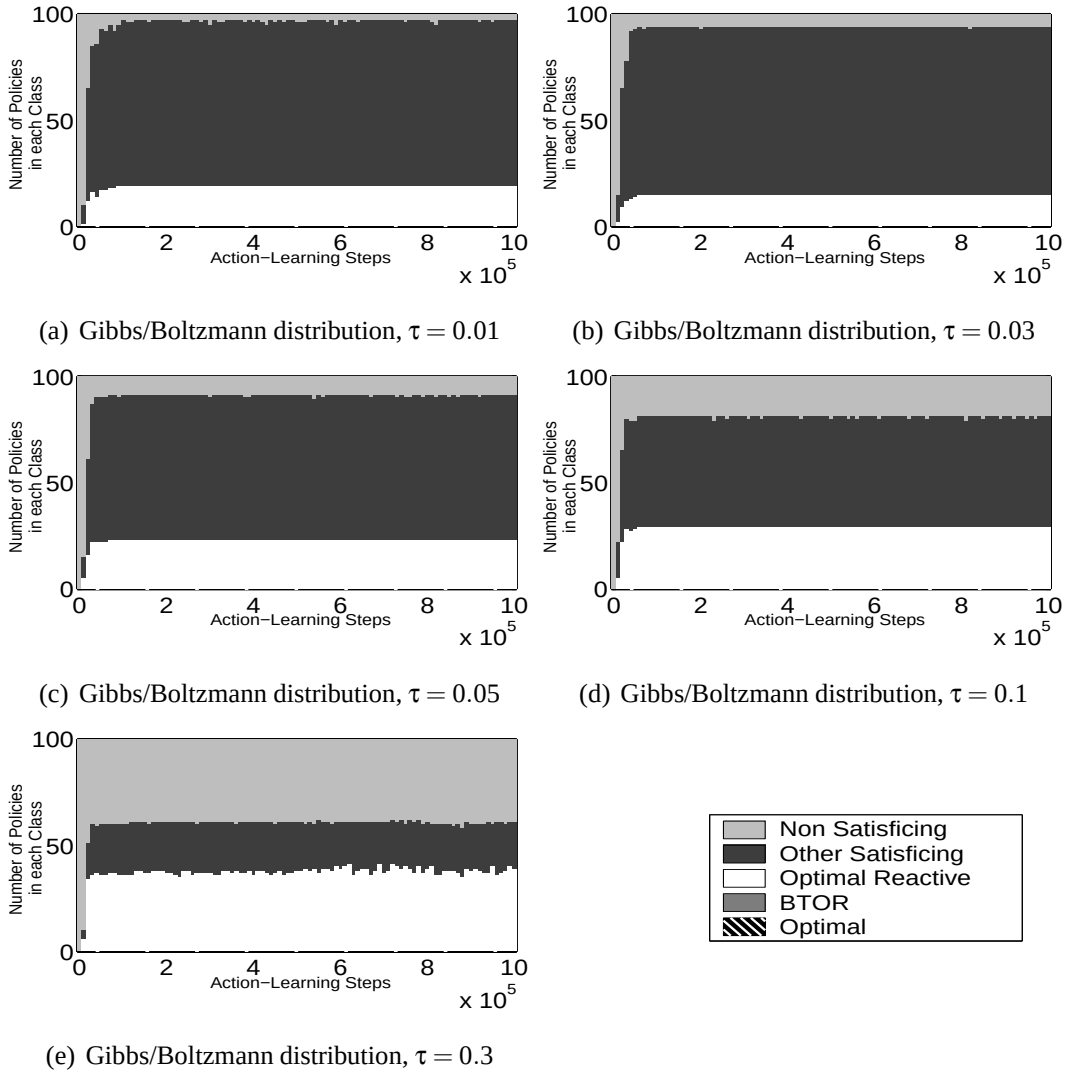


Figure 5.15: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World B using Eight Adjacent Squares Agent with $\lambda = 0.9$ and $\alpha = 0.05$ and using Gibbs/Boltzmann distribution for action selection



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

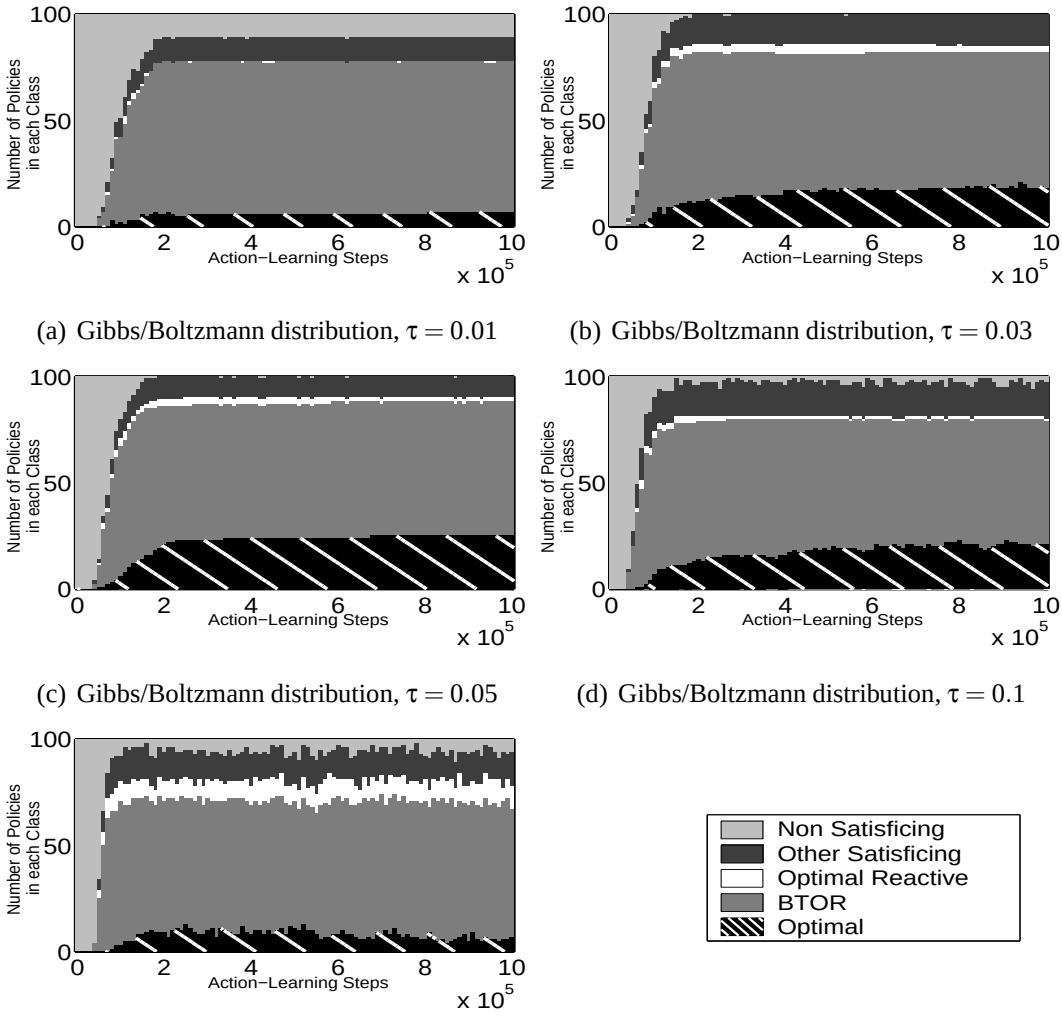
(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

Figure 5.16: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World B using Eight Adjacent Squares Agent with $\lambda = 0.99$ and $\alpha = 0.02$ and using Gibbs/Boltzmann distribution for action selection



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

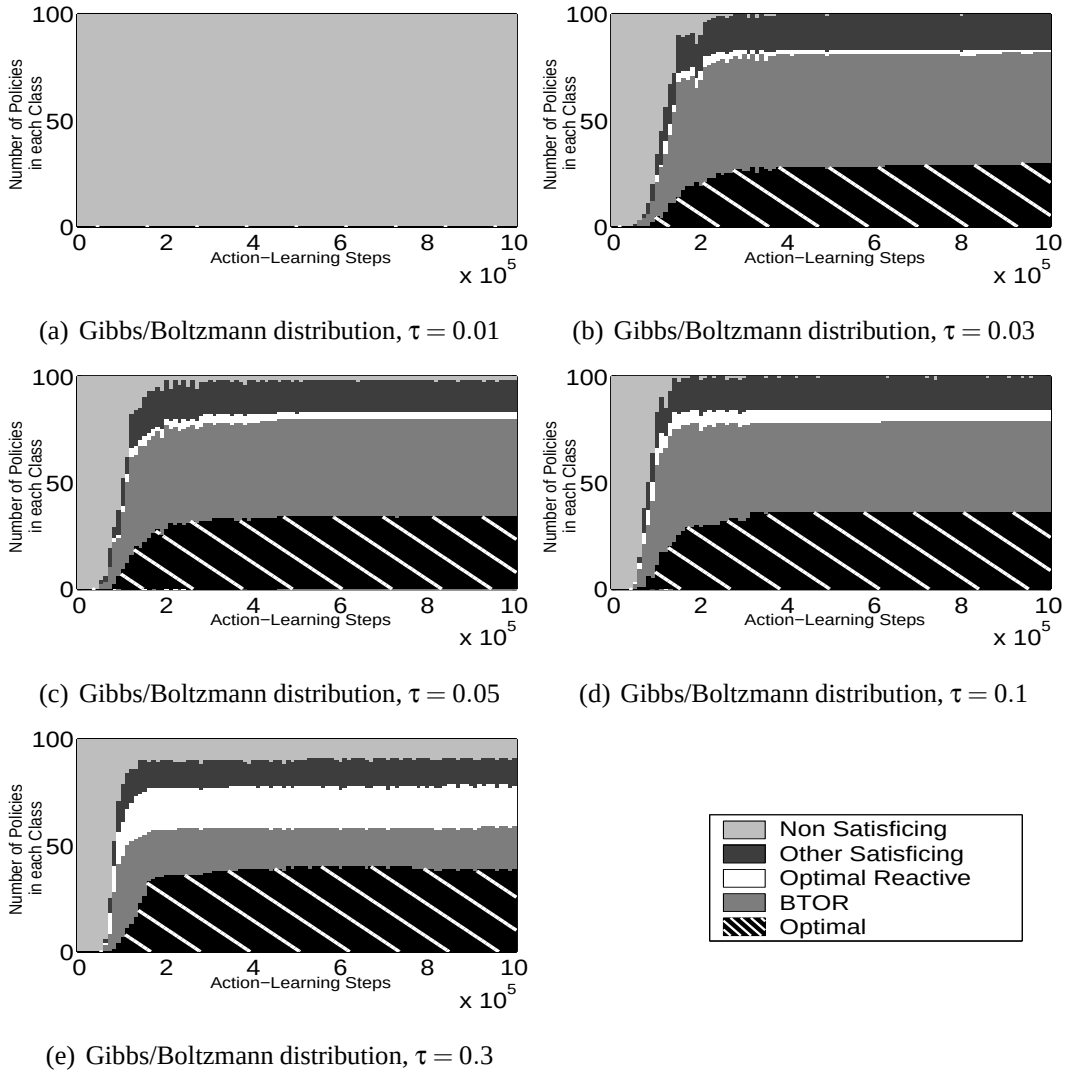
(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

Figure 5.17: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World B using Active Perception Agent with $\lambda = 0.9$ and $\alpha = 0.05$ and using Gibbs/Boltzmann distribution for action selection



(a) Gibbs/Boltzmann distribution, $\tau = 0.01$

(b) Gibbs/Boltzmann distribution, $\tau = 0.03$

(c) Gibbs/Boltzmann distribution, $\tau = 0.05$

(d) Gibbs/Boltzmann distribution, $\tau = 0.1$

(e) Gibbs/Boltzmann distribution, $\tau = 0.3$

Figure 5.18: Categorisation of policies for SARSA(λ) versus action-learning steps. Height of shaded areas indicate numbers of policies out of a total of one hundred that fall into each classification. All plots are for Sutton's Grid World B using Active Perception Agent with $\lambda = 0.99$ and $\alpha = 0.02$ and using Gibbs/Boltzmann distribution for action selection

5.2.5.4 Locally minimum policies

All of the policies that had been learnt using SARSA(λ) were tested at one million action-learning steps to see if they were locally (or globally) minimum policies. The criteria used are those set out in section 5.2.1.4. Table 5.2 collates the results of the local minimum tests for SARSA(λ).

For the Eight Adjacent Squares Agent, significantly more locally minimum policies are learnt using ε-greedy/greedy action selection than Gibbs/Boltzmann distribution, see table 5.1(a). In addition the majority of the locally minimum policies learnt by ε-greedy/greedy are categorised as optimal reactive policies which we know to be the global minimum for this problem [Littman, 1994].

The results demonstrate the existence of a non-satisficing locally minimum policy for this problem. This policy, which is illustrated in figure 5.19, has four starting locations in the bottom right hand corner that are trapped in an endless loop. Due to aliasing of observations changing any of the actions selected in such a way to break the loop leads to more starting locations becoming trapped elsewhere. For example, if the action is changed to move north for the square where 148 is observed, this frees up all four squares in bottom right corner, but creates two new loops in the two other locations where 148 is observed. The leftmost 148 location traps four starting squares, and the top right 148 location traps a further 6 starting states. The same can be shown to hold for any changes to the actions selected when observations 0 or 224 are seen. Figure 5.19 may represent the only non-satisficing locally minimum policy that exists for this problem as all 83 of the non-satisficing locally minimum policies learnt were of this general form with four starting locations trapped in a loop.

An interesting observation on the performance of Gibbs/Boltzmann distribution is that although it fails to converge on satisficing solutions for the highest value of τ that was tried, it has converged on a significant number of non-satisficing locally minimum policies. This suggest that convergence has occurred for $\tau = 0.3$, and at this value of τ the possibility of an exploratory move to break out of the loop is sufficiently high that this style of locally minimum policy is stable under SARSA(λ). For the

(a) Eight Adjacent Squares Agent

	$\alpha = 0.02$ and $\lambda = 0.99$	$\alpha = 0.05$ and $\lambda = 0.9$
Gibbs/Boltzmann, $\tau = 0.01$	0 / 0 / 0	0 / 0 / 0
Gibbs/Boltzmann, $\tau = 0.03$	39 / 19 / 10	42 / 31 / 2
Gibbs/Boltzmann, $\tau = 0.05$	35 / 18 / 4	48 / 33 / 4
Gibbs/Boltzmann, $\tau = 0.1$	26 / 10 / 5	65 / 45 / 9
Gibbs/Boltzmann, $\tau = 0.3$	41 / 13 / 27	22 / 0 / 22
ε-greedy/greedy	99 / 92 / 0	94 / 94 / 0

(b) Active Perception Agent

	$\alpha = 0.02$ and $\lambda = 0.99$	$\alpha = 0.05$ and $\lambda = 0.9$
Gibbs/Boltzmann, $\tau = 0.01$	0 / 0 / 0	0 / 0 / 0
Gibbs/Boltzmann, $\tau = 0.03$	0 / 0 / 0	0 / 0 / 0
Gibbs/Boltzmann, $\tau = 0.05$	1 / 0 / 0	2 / 0 / 0
Gibbs/Boltzmann, $\tau = 0.1$	2 / 0 / 0	5 / 0 / 0
Gibbs/Boltzmann, $\tau = 0.3$	6 / 0 / 1	0 / 0 / 0
ε-greedy/greedy	3 / 0 / 0	1 / 1 / 0

Table 5.2: Number of policies learnt that were found to be locally minimum. The figures are broken down by agent, exploration approach and parameter values. For each combination of agent type, exploration approach and parameter values, one hundred independent trials were made. Figures are: (i) total of all locally minimum policies; (ii) number included in (i) that are also globally minimum; (iii) number included in (i) which are non-satisficing locally minimum policies

ε-greedy/greedy approach, where learning continues for a sustained period while the agent acts greedily, none of the locally minimum policies are non-satisficing.

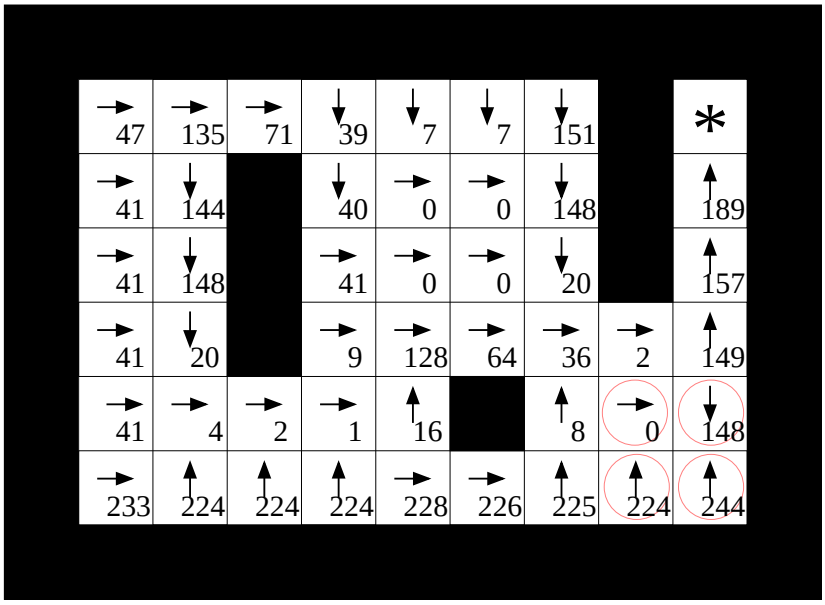
For the Active Perception Agent very few policies converge to locally minimum solutions, see table 5.1(b). This is probably due to the much enlarged observation-action space and larger range of satisficing policies that are possible with this agent. There is little to distinguish between the performance of ε-greedy/greedy action selection and Gibbs/Boltzmann distribution for action selection.

The one non-satisficing policy that is learnt is of the same form as that discussed above and shown in figure 5.19. The other result of interest is the one locally minimum policy learnt using ε-greedy/greedy action selection with parameter values of $\alpha = 0.05$ and $\lambda = 0.9$. This policy which takes a combined total of 414 steps to reach the goal when evaluated (406 physical actions and 8 perceptual actions) is the best that we have seen, and we believe that it may be the optimal solution on this problem when using the Active Perception Agent. This policy is shown in figure 5.20.

5.2.6 Conclusions for SARSA(λ)

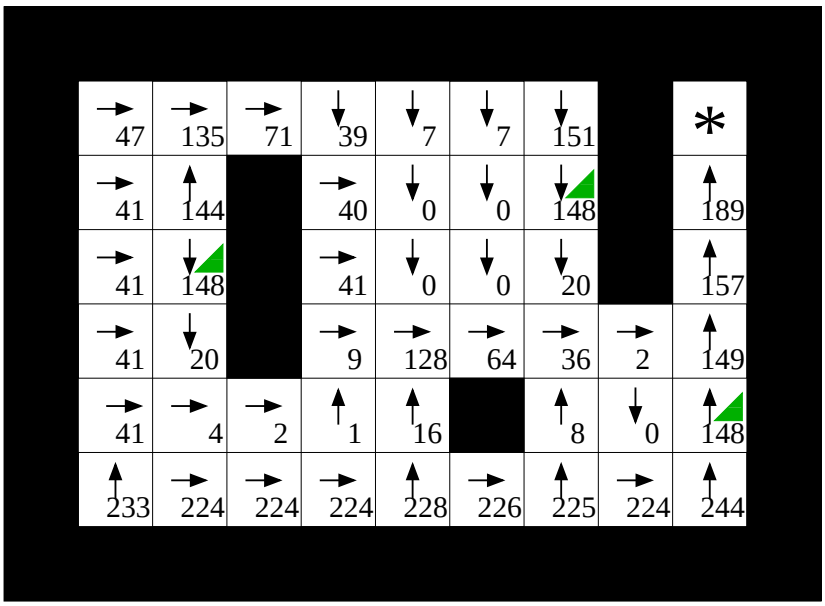
The results for Eight Adjacent Squares agent clearly show the ε-greedy/greedy approach outperforming Gibbs/Boltzmann distribution for action selection when combined with SARSA(λ). Its performance is better both in terms of the number of satisficing solutions and the number of locally/globally minimum policies which it converges to. These results are however not repeated for the Active Perception Agent where the Gibbs/Boltzmann distribution approach is equal to or marginally better than the ε-greedy/greedy approach. This lack of consistency makes it difficult to state a preference to either approach.

As for SARSA and Q-learning we see that with SARSA(λ) the tuning of τ is important in obtaining satisficing solutions. However the suggestion that arises from the locally minimum policies tests for policies learnt with $\tau = 0.3$ is that the drop off in satisficing policies might not be just a problem of convergence, but may relate to differences that



Key: 224 Observation seen by agent (converted to a decimal value)
→ Physical Action
* Goal location

Figure 5.19: Non satisficing locally optimal policy for Sutton's Grid World with Eight Adjacent Squares agent and Active Perception Agent. If an agent starts from any of the four bottom right hand locations, which are marked with faint circles, it ends up shuttling endlessly between the two locations in the bottom right corner which are numbered 148 and 244



Key: 224 Observation seen by agent (converted to a decimal value)
→ Physical Action
◢ Perceptual Action (look South East)
* Goal location

Figure 5.20: Best policy found for Active Perception agent on Sutton's Grid World. Possibly optimal policy for this problem. In locations where observation 148 is seen the agent selects perceptual action Look South East. Then, based on the new observation it obtains, it selects either physical action Move North or Move South. If the observation obtained indicates that it is in the lower right hand corner it selects Move North, otherwise it selects Move South. The combined total physical and perceptual actions is minimised by this solution; a total of 414 actions to reach the goal from all start states. This consists of 406 physical actions and 8 perceptual actions. The number of physical actions could be reduced further by selecting physical action Move East for observation 0, but this increases the number of starting locations that pass through the top right-hand 148 location and therefore increases the number of perceptual action executed

exist between the way that the problem behaves when learning and way it behaves when we evaluate policies. In the end it appears that if we intend the agent to act greedily, then on POMDPs it is worth learning for a sustained period whilst acting greedily with respect to the current observation-action values.

For all of SARSA(λ), SARSA and Q-learning we have used a static learning rate α for all the empirical examples run. However, most theoretical convergence results require that the learning rate is decayed in a reasonable manner [Watkins and Dayan, 1992; Dayan, 1992; Singh et al., 2000]. We justify our approach on the grounds that we looked at the effect of decaying α in preliminary experiments. We found that for ε-greedy/greedy approach, decaying α significantly over one million action-learning steps slowed convergence and produced worse results. This suggests that any decay in α should be very slow, and it is possible to regard the constant value used over one million action-learning steps as approximating a very slow decay which would take many millions of action-learning steps to decay significantly. This fits with theoretical results which only required α to reach zero at infinity. A further mitigating issue is that we are primarily interested in the policies learnt, not the underlying observation-action values. In many cases for both the ε-greedy/greedy and the Gibbs/Boltzmann approaches the policies learnt appear largely static, thus any underlying noise in the observation-action values is for our purposes insignificant.

5.3 Chapter Summary

In section 5.1 we examined the dependence between observation-action values and exploration. Existing results show that using ε-greedy to control exploration it is possible to create POMDPs for which there exists no stable policy. We clarify that this only holds for $\varepsilon > 0$ and take some preliminary steps towards showing that stable policies may exist on POMDPs when the learning algorithm acts greedily.

Empirically we compare an exploration approach that becomes greedy for a sustained period of time (ε-greedy/greedy) with the commonly used Gibbs/Boltzmann distribution for action selection (also known as softmax).

For the Gibbs/Boltzmann distribution, existing work has shown the existence of stationary points in the observation-action space. Our empirical results show that these stationary points do not necessarily correspond to static policies, in fact as they are stochastic stationary points there is no reason why they should do so.

Based on our empirical comparison it is difficult to make a recommendation between using ε-greedy/greedy or Gibbs/Boltzmann distribution for action selection. However, a suggestion that does arise in this chapter is that in applying simple reinforcement learning approach to POMDPs, it is possibly worth matching the conditions under which the agent learns with those under which it will be evaluated. That is, if we expect a policy to act greedily then it needs to learn whilst following a policy greedily. This point requires further investigation in order to confirm it, but if it is true then the ε-greedy/greedy approach better matches the requirements set out here than using Gibbs/Boltzmann distribution with constant τ. The later never acts completely greedily during learning where as the former does, and we always evaluate the policies greedily. To address this issue for the Gibbs/Boltzmann distribution, it would be possible to either let $\tau \to 0$ over the course of learning or slightly better computationally[4] use the Gibbs/Boltzmann distribution initially to allow exploration then switch to a greedy policy.

[4] Letting $\tau \to 0$ in the equation 5.23 results in extremely large numbers which can prove difficult for some programming languages to properly handle.

Chapter 6

Monte Carlo Approaches and Consistent Exploration

The aim of this chapter is to demonstrate the development of the idea of maintaining *consistent exploration* when learning satisficing policies for partially observable tasks. We approach this aim through an examination of the few theoretical results have been shown to apply to reinforcement learning on POMDPs, such as the use of (i) undiscounted first-visit Monte Carlo returns and (ii) the MCESP algorithm [Perkins, 2002b; Pendrith and McGarity, 1998]. We also discuss the shortcoming of Monte Carlo style approaches with regard to their possible application to real robotic platforms. These issues coupled with the concept of consistent exploration gives rise to our new reinforcement learning algorithm which we present in chapter 7.

6.1 First Visit Monte Carlo

Pendrith and McGarity [1998, 1997], as reviewed in section 3.2.3, present a proof that if actions are valued using a first-visit Monte Carlo approach with undiscounted returns, then optimal reactive policies are learning equilibria. Their paper offers no

guarantee of convergence to optimal reactive policies but optimal reactive policies, if they are arrived at, should be stable under this learning algorithm.

Showing that optimal policies are stable under a particular learning algorithm does not indicate whether that algorithm is capable of learning such policies in the first place. One factor, as discussed in the previous chapter, is the role of exploration. Some form of exploration is required as observation-action values are learnt through sampling the actions available from each observation. However, Pendrith and McGarity [1998, 1997]'s proof does not give any indication as to the effect that exploration might have on the status of optimal policies as equilibrium points when learning.

The first-visit Monte Carlo algorithm is typically coupled with *exploring starts*, as defined by Sutton and Barto [1998, chp.5] and set out in Algorithm 1 in this chapter. We examine the exploring starts approach below and conclude that the stability of optimal observation-based POMDP policies cannot be guaranteed under this approach. We conclude from the examples set out below, that we need to ensure that exploration is constrained in such a way that the learning algorithm is consistent when re-encountering the same observation. This constraint is required in order to obtain accurate samples of the value of actions. Based on this we introduce a modified form of exploring starts: *consistent exploration exploring starts*.

6.1.1 Exploring starts

We consider, through the use of two illustrative examples, the accuracy of action values as they are sampled using first visit Monte Carlo coupled with exploring starts. The example shown in figure 6.1 (a), could occur in a Markovian grid world task, that is, a MDP. The exploring start selected for the agent consists of starting in the furthest left square and selecting the action Move West. If we assume that the task is designed such that a move towards a wall results in the agent remaining where it is, then the exploratory action will result in no change of the agent's state. If the agent then follows policy it will then head east. We assume that moving east ultimately leads the agent to reach its goal and the end of the episode. The estimated value of executing Move West

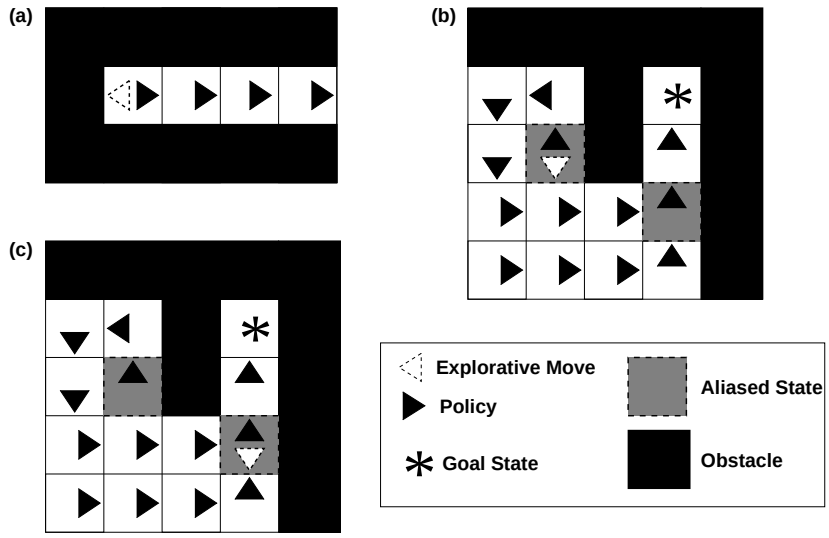


Figure 6.1: How exploring starts can arrive at inaccurate samples of action-values for both MDPs and POMDPs

in this state will, according to the first visit Monte Carlo algorithm, be the value of moving east to the goal from this state, plus whatever penalty is imposed for initially moving towards a wall. By comparison, if this action were adopted as the policy action for this state, then (assuming a deterministic task) the reward would be a sum to infinity of the penalty received for continually moving towards the wall. Thus the Monte Carlo exploring starts algorithm underestimates[1] the true cost of having a policy that selects Move West from the leftmost state. Underestimating the sampled action value in this case is not a problem if the policy update is of the form $\pi \leftarrow \arg\max_a Q(s,a)$ and either some negative reward is given to the agent for trying to move towards the wall, or future returns are discounted. In these cases the action Move West will always have a higher relative cost than the action Move East and thus will not be selected to form part of the policy.

The second example, figure 6.1 (b) and (c), arises in a POMDP task due to the aliasing of two states. This example poses much more serious problems for the first-visit Monte Carlo exploring starts algorithm. In figure 6.1 (b) the randomly chosen exploring start

[1] We assume the sum to infinity of the penalty received for moving towards a wall is greater than the cost of reaching the goal even after allowing for a discount rate $0 < \gamma < 1$.

position is the leftmost of a pair of aliased states and the exploratory action is Move South. Having executed this action the agent then follows policy and arrives at the goal state in a total of five moves. Moving South from this state thus appears to be better than moving North which takes a total of nine moves to reach the goal. Consider now figure 6.1 (c) where the agent has an exploring start in the aliased square nearer the goal and selects Move South. The agent then follows its policy and heads back north reaching the goal state in total of four moves. The learning algorithm will associate the two states with the same observation. As a result the observation-action value will be an average of the samples obtained from both states. This average will be weighted by the relative frequency with which each of the identical observation-action pairs is visited. Assuming that exploring starts selects across all states and actions with uniform probability, then the likelihood of selecting the non-policy action Move South from either of the aliased states will be equal. Thus in the two aliased states the action Move South will appear to take an average of four and a half moves to reach the goal. The apparent cost of selecting Move North from the aliased states will not be a simple average as one state is more likely to be visited as it lies on the path mapped out by the current policy to the goal. It is however possible to imagine that the weighted average of nine steps and two steps could exceed four and a half. Assuming action values correspond to path lengths, the action value learnt for Move South (as an exploratory action) could therefore appear better than the action value for Move North. If the policy update equation $\pi \leftarrow \arg\max_a Q(s,a)$ is used, moving south could be selected as the policy action. At this point the goal can no longer be reach from either aliased state when the agent follows its new policy.

We argue that value obtained for Move South as an exploratory action is in error when estimated using exploring starts. The sample value obtained should instead reflect its value if the action were adopted as part of the policy. In this example the sampled exploratory cost of Move South should not be four and a half steps but should instead be a value that reflects the impossibility of reaching the goal.

Empirical evidence that optimal policies are not stable when using undiscounted first-visit Monte Carlo with exploring starts is presented in section 6.1.3. By comparing results with an exploration strategy that is constrained to act consistently when re-

Algorithm 1 First-visit Monte Carlo algorithm using exploring starts [Sutton and Barto, 1998, chp.5, p.120]. Notation modified compared to that used in Sutton and Barto [1998] to reflect the separation between states s and observations o as used throughout this thesis. O is the observation space, $A(o)$ is the space of actions available given observation o, a is an individual action. Return R calculated as $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$ where γ is the discount rate

Initialise, for all $o \in O$, $a \in A(o)$:

$Q(o,a) \leftarrow$ arbitrary values
$\pi(o) \leftarrow$ arbitrary
$Returns(o,a) \leftarrow$ empty list

Repeat forever:

(a) To generate a complete episode using exploring starts:

Select a random start state,
Randomly execute an action which is allowed in this state,
Then follow policy π until the end of the episode.

(b) For each observation-action pair o,a appearing in the episode:

$R \leftarrow$ return following the first occurrence of o,a.
Append R to $Returns(o,a)$.
$Q(o,a) \leftarrow$ average($Returns(o,a)$)

(c) For each o in the episode:

$\pi_o \leftarrow \arg\max_a Q(o,a)$.

encountering the starting observation we lend support to the above analysis.

With respect to the work of Pendrith and McGarity [1998, 1997], these two examples do not affect their claim that, when undiscounted first-visit Monte Carlo methods are used, optimal reactive policies will be learning equilibria. However this guarantee only holds if exploration is constrained.

6.1.2 Consistent exploration

The above inaccuracy in the sampling of action values when using exploring starts can be addressed by choosing to sample the value of actions under different policies, as opposed to sampling the value of actions under different starting conditions. This can be achieved by changing the policy each time such that it incorporates the initial exploratory action, and therefore executes the same exploratory action consistently (as if it were the policy) each time the starting observation is re-encountered. We present a version of the first visit Monte Carlo exploring starts algorithm that we modified to achieve this. The modified algorithm, Algorithm 2, is similar to Algorithm 1. Both use exploring starts, however, Algorithm 2 records the initial observation and exploratory action and then selects the same exploratory action every time it re-encounters the starting observation.

Support for the approach of comparing policies when dealing with POMDPs is provided by considering the Policy Improvement Theorem of Dynamic Programming [Watkins, 1989; Sutton and Barto, 1998]. Watkins [1989, p.46] presents the policy improvement theorem as the justification for comparing alternative policies by selecting an exploratory action and then following the current policy. Briefly, the policy improvement theorem states that if we wish to compare two policies π and π', we can (for a MDP) follow policy π' for one step and then follow policy π. If π' is better than π then we will find the state-action values $Q_\pi(s,\pi') \geq V_\pi(s)$. If this is the case then the policy improvement theory guarantees that π' is uniformly better than π, that is, $V_{\pi'}(s) \geq V_\pi(s)$ for all $s \in S$. Thus policy improvement avoids the need to compute

Algorithm 2 Our consistent exploration version of first-visit Monte Carlo algorithm using exploring starts. Bold text highlights the modifications which implement consistent exploration

Initialise, for all $o \in O$, $a \in A(o)$:

$Q(o,a) \leftarrow$ arbitrary values
$\pi(o) \leftarrow$ arbitrary
$Returns(o,a) \leftarrow$ empty list

Repeat forever:

(a) To generate a complete episode using consistent exploration and exploring starts:

Select a random start state,

Record the observation seen when in this state (o_0),

Randomly execute an action which is allowed in this state,

Record the action executed (a_0),

Then **follow policy modified by executing action a_0 when o_0 is observed** ($\pi \leftarrow (o_0, a_0)$) until the end of the episode.

(b) For each observation-action pair o,a appearing in the episode:

$R \leftarrow$ return following the first occurrence of o,a.

Append R to $Returns(o,a)$.

$Q(o,a) \leftarrow$ average($Returns(o,a)$)

(c) For each o in the episode:

$\pi_o \leftarrow \arg\max_a Q(o,a)$.

$V_{\pi'}(s)$ and $V_{\pi}(s)$ over the entire state space in order to compare two policies. Unfortunately the policy improvement theorem implicitly relies on the Markovian principle and thus can not be guaranteed to hold for POMDPs. On this basis the approach of selecting one exploratory action and then following policy as a method of comparing alternative policies looks distinctly shaky. We are therefore forced back to the idea of separately computing $V_{\pi'}(o)$ and $V_{\pi}(o)$ over the observation space of the POMDP and comparing them. For a POMDP there is no guarantee that there exists *any* policy that is uniformly better over the entire observation-action space [Pendrith and McGarity, 1997, 1998], that is, $Q_{\pi'}(o,a) \geq Q_{\pi}(o,a)$ can not be guaranteed to hold for all o in a POMDP. Therefore it is not worth computing $V_{\pi'}(o)$ and $V_{\pi}(o)$ over the *entire* space of observations and actions and comparing them. Given this, the consistent exploration approach of sampling an action's value when following a policy which involves that action seems a reasonable approach, that is, we limit our sample to just those observations o that are visited under the policy, plus receive only those rewards which are possible given the combination of actions that make up the current policy.

In addition to the above, we also demonstrate in section 6.1.3 that optimal policies are more stable using this consistent exploration approach, compared to conventional exploring starts.

6.1.3 Empirical test of stability of optimal policies

6.1.3.1 Experimental setup

To support our analysis above, we compare the stability of existing optimal reactive policies using undiscounted first-visit Monte Carlo algorithms in conjunction with three different approaches to exploration:

(i) Exploring starts – Algorithm 1,

(ii) Greedily following policy. Starting locations selected at random but actions always selected according to policy. This is effectively Algorithm 1 but replacing

"Randomly execute an action which is allowed in this state," with "Select action according to policy for this state."

(iii) Consistent exploration applied to exploring starts – Algorithm 2.

Approach (ii) does not represent a realistic approach to exploration as it does not sample the whole observation-action space. It is included to demonstrate that the instability in optimal policies under (i) can be attributed to the random selection of an initial action that does not match the subsequent policy which is followed. Approach (iii) continues to allow the selection of an initial action at random, but then modifies the current policy to incorporate that action.

For these tests we use Sutton's Grid World and the Eight Adjacent Squares Agent as described in section 4.1.1. We use ten versions of the optimal reactive policy. The optimal reactive policy is the best policy that can be achieved using this agent [Littman, 1994]. The ten optimal policies used were learnt using the first-visit Monte Carlo algorithm using exploring starts (Algorithm 1) with undiscounted returns ($\gamma = 1$). These ten optimal policies were arrived at by running one hundred independent trials on this problem.

Although all ten policies take the same number of steps to reach the goal from all start locations (a total of 416 steps) they can vary in the precise actions selected as there is more than one optimal solution. They also vary in the precise value of the observation-action pairs, including variation in the samples that are stored and averaged together to calculate the observation-action values. It is because of these variations that we use a sample of ten optimal policies.

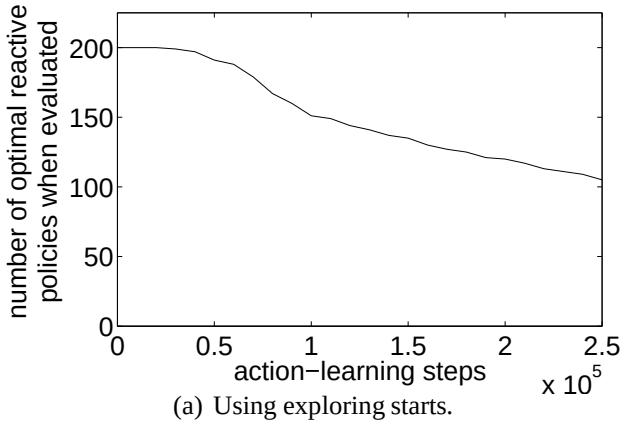
All three algorithms are initiated with the same ten optimal policies. That is say that for each algorithm, the variables of Q and *Returns* are initiated using each of the optimal policies in turn. The current policy (π) is given by $\arg\max_a Q(o,a)$ at all times. For each combination of algorithm and policy twenty independent runs are then made. Each run lasts for a total of 250,000 action-learning steps, with the policy being evaluated greedily every 10,000 action-learning steps to test whether it was still optimal. For all three algorithms $\gamma = 1$.

Note that no upper bound was placed on the length of an episode, therefore an episode only ends when the goal state is reached. This is significant when using Monte Carlo style algorithms, as updates only occur when an episode ends. By not imposing an upper bound, any policy that is updated such that it becomes non-satisficing is likely to become stuck in this state. A non-satisficing policy is unable to reach the goal state from all start locations. If during learning it starts in one such location it will subsequently never reach the goal and thus the policy will not be updated further. This will prevent a portion of non-satisficing policies recovering. The upshot is that this effect will make the test more sensitive to policy instability. This in part justifies the sampling of policies at every $10{,}000^{th}$ action-learning step. This rather coarse sampling rate may miss some transitions from optimal to satisficing policies, and back again, but will detect all transitions to non-satisficing policies when they get trapped and can never recover.

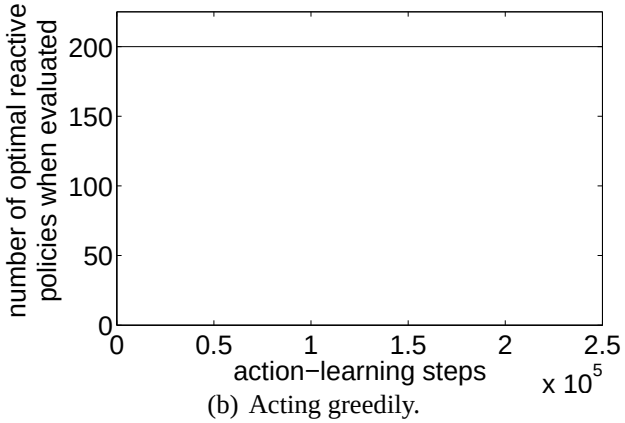
6.1.3.2 Results

The number of policies that remained optimal over time was counted for each of the three approaches to exploration. These totals, which are out of 200 (20 samples of 10 initially optimal policies), are plotted in figure 6.2. As can be seen from the plots all the policies remained optimal when the algorithm either acted greedily with respect to the policy or followed a consistent exploration policy. This contrasts sharply with the case where exploring starts was used: figure 6.2(a) In this case the sampled action values were such that the updates resulted in policies that ceased to be optimal. In examining individual runs we found that all the policies which where found to be non-optimal when policy evaluation was carried out, had become non-satisficing policies.

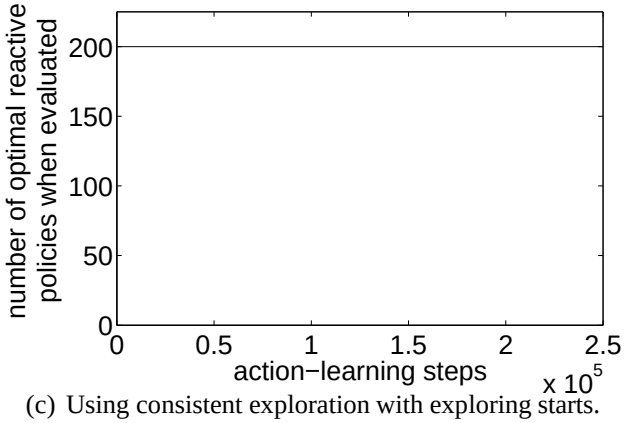
These results support the above analysis as all three algorithms are collecting *Return* values, using these to update the Q-values which in turn determine the policy using $\arg\max_a Q(o,a)$. Only in the case where an initial exploratory action is selected which is different to the policy that is subsequently followed does this apparently result in unstable optimal policies.



(a) Using exploring starts.



(b) Acting greedily.



(c) Using consistent exploration with exploring starts.

Figure 6.2: Comparison of stability of optimal policies when using different exploration approaches with first-visit Monte Carlo algorithms that are using undiscounted returns

Although these results demonstrate improved stability of optimal policies when using consistent exploration with exploring starts, they do not demonstrate whether this approach is any good at learning such optimal policies in the first place. This we explore empirically in section 6.3 where we look at the ability of undiscounted first-visit Monte Carlo algorithms to learn policies on Sutton's Grid World.

6.2 MCESP

In introducing the Monte Carlo Exploring Starts for POMDPs (MCESP) algorithm Perkins [2001, 2002b] redefine the value of an observation-action pair such that policies are stable when using discounted future rewards. Although in the published literature they do not consider the problems associated with exploring starts or the need for consistent exploration as discussed in section 6.1, they do introduce useful notation which neatly describes consistent exploration. They define $\pi \leftarrow (o,a)$ to represent a policy that is identical to policy π except that observation o is mapped to action a. As this notation concisely captures the concept of consistent exploration we adopt it for the remainder of this thesis.

The new definition of observation-action values that Perkins [2001, 2002b] introduces is:

$$Q^{\pi}_{(o,a)} = E^{\pi \leftarrow (o,a)}\{R_{post-o}(\tau)\} \tag{6.1}$$

where τ is the trajectory followed over one episode, $R_{post-o}(\tau)$ is the portion of return for trajectory τ following on from the occurrence of observation o, and $E^{\pi \leftarrow (o,a)}$ is the expected value, given a set of trajectories that pass through o, and which are following the modified policy $\pi \leftarrow (o,a)$. "In words, $Q^{\pi}_{(o,a)}$ is the portion of the expected return that follows the first occurrence of o, if the agent takes action a whenever it observes o and adheres to π otherwise" [Perkins, 2002b]. This redefinition differs from the standard form used for MDPs in three key respects:

(i) States s are replaced by observations o.

(ii) The agent is consistent in its exploration ($\pi \leftarrow (o,a)$), that is, it takes action a every time it sees observation o

(iii) the observation-action value is *the portion* of the discounted return that follows on from observing o, and not the discounted return as if starting at o.

Point (iii) makes Perkins [2001, 2002b]'s approach significantly different from the work we have considered so far. It is this point that makes a difference in how the discount factor comes into play when o occurs at different times in different trajectories. "Consider, for example, two trajectories, with reward sequences $\{r_0, r_1, r_2, \ldots, r_T\}$ and $\{r'_0, r'_1, r'_2, \ldots, r'_{T'}\}$. Suppose that in the first trajectory, o occurs on time step 2, and in the other, o occurs on time step 4. Then the first trajectory contributes $\gamma^2 r_2 + \gamma^3 r_3 + \ldots + \gamma^T r_T$ to the observation-action value, and the second trajectory contributes $\gamma^4 r'_4 + \gamma^5 r'_5 + \ldots + \gamma^{T'} r'_{T'}$. Under the standard definition, the trajectories would contribute $r_2 + \gamma r_3 + \ldots + \gamma^{T-2} r_T$ and $r'_4 + \gamma r'_5 + \ldots + \gamma^{T'-4} r'_{T'}$."

For a MDP a policy is optimal if and only if it is greedy with respect to its action values (as normally defined). This is not true for POMDPs but the above redefinition allows this property to be preserved to some degree. Perkins [2002b] proves the theorem that for all π and $\pi' = \pi \leftarrow (o,a)$,

$$V^{\pi} + \delta \geq V^{\pi'} \iff Q^{\pi}_{o,\pi(o)} + \delta \geq Q^{\pi}_{o,a} \tag{6.2}$$

A corollary of this is that a POMDP policy is locally optimal if and only if it is greedy with respect to its action values (as redefined above). A policy π is δ-locally optimal if and only if $Q^{\pi}_{o,\pi(o)} + \delta \geq Q^{\pi}_{o,a}$ for all o and a. The MCESP (Monte Carlo Exploring Starts for POMDPs) algorithm presented in Perkins [2002b], and reproduced in Algorithm 3, incorporates the above redefinition and hence the theoretical properties.

MCESP is different in the way that it treats exploration compared to the consistent exploration version of the first-visit Monte Carlo algorithm (Algorithm 2). MCESP does not necessarily select an action to explore based on its starting observation. It modifies its policy by selecting randomly among any of the observations that it has previously

Algorithm 3 Monte Carlo Exploring Starts for POMDPs (MCESP) [Perkins, 2002b]. (Note symbol δ substituted for comparison threshold in place of ε which was used in Perkins [2002b].)

Inputs:

learning rate schedule $\alpha(n,i)$,
comparison threshold schedule $\delta(n,i,j)$
discount rate γ

Initialise, for all $o \in O$, $a \in A(o)$:

$Q(o,a) \leftarrow$ arbitrary values
$\pi(o) \leftarrow$ arbitrary
$c(o,a) \leftarrow 0$
$n \leftarrow 0$

Repeat forever:

(a) To generate a complete episode τ:

Randomly choose some o and $a \in A(o)$,
Follow policy $\pi \leftarrow (o,a)$ until end of episode.

(b) For each observation-action pair o,a appearing in the episode:

$$Q(o,a) \leftarrow (1-\alpha(n,c(o,a)))Q(o,a)+\alpha(n,c(o,a))R_{post-o}(\tau)$$
$$c(o,a) \leftarrow c(o,a)+1$$

(c) For each o in the episode:

if $max_{a'}Q(o,a') - \delta(n,c(o,a'),c(o,\pi(o))) > Q(o,\pi(o))$ then
$\pi(o) \leftarrow a' \in \arg\max_{a'} Q(o,a') - \delta(n,c(o,a'),c(o,\pi(o)))$
$n \leftarrow n+1$
$c(o'',a'') \leftarrow 0$ for all arbitrary o'' and a''
end if

encountered[2] and then selecting an action at random to explore when that observation occurs. This approach has both advantages and disadvantages. The advantage of the MCESP approach to exploration is that the algorithm will work with tasks where some observations are not accessible at the start of an episode. The exploring starts approach requires that the agent can start from all possible states of the task, if not then exploring starts will not fully explore the observation-action space. The disadvantage is that the observation the algorithm has chosen to explore may not occur during the episode when it has been chosen. This will have the effect of reducing the amount of exploration, that is, depending on the task, it may spend many episodes simply following the current policy π. This will slow exploration and is likely to slow convergence of policies. In contrast, when using the exploring starts approach, the chosen observation is guaranteed to occur at least once at the start of the episode.

The definition of action values adopted for MCESP algorithm has the advantage of allowing the discounting of future rewards. In addition MCESP introduces two further innovations:

(i) The learning rate for observation-action values can be scheduled individually for each observation-action pair. The schedule can for example depend on how often an action has been tried from a given observation, and when the overall policy was last updated.

(ii) A requirement that the difference in value between alternative actions has to exceed a given threshold[3] δ before the policy is changed in favour of the dominant action value, that is, $\pi \leftarrow \arg\max_a Q(o,a)$ if $Q(o,a) > \delta + Q(o,\pi(o))$.

Of three variations of the MCESP algorithm presented in Perkins [2002b] the version that uses a constant δ performs the best, MCESP-CD[4] (CD for constant delta).

[2] This is generally implemented by recording every unique observation that has been encountered in previous episodes and selecting randomly over this set [Perkins, 2001].

[3] We have adopted the symbol δ for the minimum threshold in this thesis in place of ε which is used in Perkins [2002b]. This is to reduce confusion with the completely separate use of the symbol ε when discussing ε-greedy action selection.

[4] Perkins [2002b] refers to this algorithm as MCESP-CE (CE indicating constant epsilon). To avoid confusion with our use of CE to refer to "consistent exploration" and to maintain consistency with our use of δ in place of ε we use the name MCESP-CD.

MCESP-CD has the interesting theoretical property, that given some reasonable starting conditions (see section 3.2.4), it will, with probability 1, converge to a δ-locally optimal policy, where a policy is δ-locally optimal if and only if $Q^{\pi}_{o,\pi(o)} + \delta \geq Q^{\pi}_{o,a}$ for all o and a.

6.3 Experiment

These experiments are designed to allow comparison between the three learning algorithms presented in sections 6.1 and 6.2:

(i) first-visit Monte Carlo using exploring starts,

(ii) consistent exploration variation of first-visit Monte Carlo, and

(iii) MCESP-CD.

The results obtained can be compared to algorithms used in previous sections such as SARSA(λ), see section 4.3.

The aim is to examine the relative performance of these algorithms in terms of numbers of satisficing and optimal policies, and compare this to the best performing algorithm found so far: SARSA(λ). The experiments are also designed to test the convergence properties of the algorithms, especially MCESP-CD for which guarantees of convergence have been claimed Perkins [2002b].

6.3.1 Grid world and agent

To compare the three learning algorithms we use Sutton's Grid World as modified by Littman [Littman, 1994] and the Eight Adjacent Squares Agent, see section 4.1.1 for details.

6.3.2 Learning algorithms and parameters

6.3.2.1 First-visit Monte Carlo algorithms

The first-visit Monte Carlo algorithm using exploring starts, and the consistent exploration version of the same algorithm have very few parameters. We use a discount rate of $\gamma = 1$ as required by the theoretical findings of Pendrith and McGarity [1997, 1998], and fixed the maximum length of an episode, whilst the agent is learning, to 100 action-learning steps. Episodes normally end when the agent reaches the goal state, however, without some limitation on their length an agent which deterministically follows a policy can end up trapped in a loop. As Monte Carlo based algorithms do not update their policies until the episode ends, any loop will permanently trap the agent. To guard against this episodes are artificially curtailed and the agent restarted from a random location.

For the consistent exploration version of the Monte Carlo algorithm the amount of exploration remains constant throughout training, with the first action of each episode always being selected at random using a uniform distribution across all the actions allowed.

Analysis in section 6.1 demonstrated that optimal policies are not necessarily stable when using exploring starts without consistent exploration. Knowing this we adopt two exploration schedules for first-visit Monte Carlo algorithm using exploring starts. In one schedule the amount of exploration remains constant throughout training, with the first action of each episode being selected at random using a uniform distribution across all the actions allowed. In the second schedule, the starting action is selected at random across all available actions with a probability of ε, or, with probability $1 - \varepsilon$, the action is selected according to the current policy. The value of ε is initially set equal to 1 and then decreases linearly with the number of action-learning steps until it reaches zero at 750,000 action-learning steps. For the remaining 250,000 action-learning steps ε remains equal to zero so that the current policy action was always selected.

For both versions of the first visit Monte Carlo algorithm the start location of each episode is selected at random using a uniform distribution across all locations in Sutton's Grid World that do not contain an obstacle.

Two sets of initial values for observation-action pairs are tried. In one set all observation-action values are initiated to zero. This allows comparison with the performance of learning algorithms used in previous experiments. In the second set observation-action values are initiated to random values drawn from a Gaussian distribution with mean of zero and standard deviation of five. The aim of the latter set is to better test the convergence properties of these algorithms from different arbitrary policies. The policy π for both first-visit Monte Carlo algorithms is determined at all times as being $\pi = \arg\max_a Q(o,a)$ with ties between observation-action values being broken at random.

6.3.2.2 MCESP-CD

For the MCESP-CD algorithm we use the same basic parameters for episode length, randomised choice of starting location, and initiation of observation-action values as set out in section 6.3.2.1 above.

Exploration is constant throughout training. At the start of each episode the observation to explore is chosen at random using a uniform distribution over the set of observations that the MCESP-CD algorithm has experienced from previous episodes. It maintains a list of unique observations encountered since the beginning of each trial for this purpose. The action to explore is selected at random using a uniform distribution over the set of actions afforded when the observation to explore is encountered.

MCESP-CD maintains a policy which is independent of the action values. This policy is initiated to match the set of initial action values using $\pi = \arg\max_a Q(o,a)$, with any ties between actions values being broken at random. Thus in the case that the action values are initiated to random values drawn from a Gaussian distribution, the policy is initiated to select actions that match the randomly selected action values. In the

case when the action values are initiated to zero the algorithm initiates the policy to randomly selected actions for each observation.

The theoretical guarantees claimed for MCESP-CD hold for discounted future returns. Three values of discount rate γ are tried 0.75, 0.9 and 1.

The comparison threshold δ is fixed in this version of the algorithm (constant delta). Five fixed values are tried; 0.1, 0.01, 0.001, 0.000 01 and 0.000 000 1.

A learning rate schedule of $\alpha = \frac{0.1}{1+c(o,a)}$ is used, where $c(o,a)$ is the count of the number of times action a has been taken given observation o since the policy was last updated, see Algorithm 3. This learning schedule meets the requirement that the sequence of values of α should equal or lie between two Robbins-Monroe sequences[5] for convergence to be guaranteed [Perkins, 2002b].

6.3.2.3 Evaluation criteria

For each combination of parameters one hundred independent trials were run, each trial lasting for a total of one million action-learning steps. The policies learnt during each trial were evaluated every one thousand action-learning steps and categorised based on the number of physical action required to reach the goal. Table 6.1 summarises the categorises used. More detail on evaluation and the policy categories used can be found in section 4.1.1.4.

We also examine the policies learned at one million action-learning steps to see if they are locally minimum. We use the same criteria as described in section 5.2.1.4.

6.4 Results

The results for the first-visit Monte Carlo algorithms using the three approaches to exploration are shown in figure 6.3. The plots show the number of policies that fall into

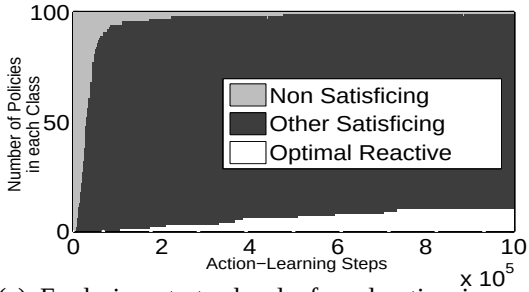
[5] $\{x_i\}_{i=0}^{\infty}$ is a Robbins-Monroe sequence if $x_i \in [0,1]$ for all i, $\sum_i x_i = \infty$, and $\sum_i x_i^2 < \infty$.

Goal Reached From All Starting Locations	Total Physical Actions	Policy Category
yes	404	Physically Optimal
yes	$405 - 415$	Better Than Optimal Reactive (BTOR)
yes	416	Optimal Reactive
yes	> 416	Other Satisficing
no	-	Non-Satisficing

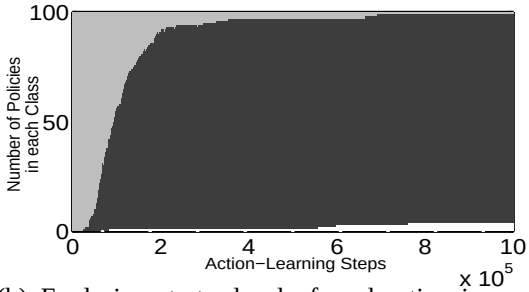
Table 6.1: Policy categories for Sutton's Grid World

the three categories, optimal reactive, other satisficing and non satisficing, against the number of action-learning steps that have passed. Initial convergence rates are similar for all three exploration approaches, though as shown in figures 6.3(a) and 6.3(b), the number of policies that converge to satisficing solutions is ultimately better for the unmodified version of the first-visit Monte Carlo algorithm using exploring starts, that is, Algorithm 1. Gradually reducing the probability of selecting an exploratory action to zero appears to offer no benefit, see figures 6.3(c) and 6.3(d). The number of satisficing and optimal reactive policies that are learnt by this approach is less than the number learnt in figures 6.3(a) and 6.3(b), where a constant probability of exploration is maintained. The result for the consistent exploration version of the first-visit Monte Carlo algorithm, Algorithm 2, shown in figures 6.3(c) and 6.3, and are particularly interesting, with the number of policies in each category varying in what appears to be a quite noisy manner. Despite evidence in section 6.1.3 that optimal policies are more stable using this approach, these results suggest that optimal policies, and some portion of satisficing policies, are unstable when this approach is used to learn policies from scratch.

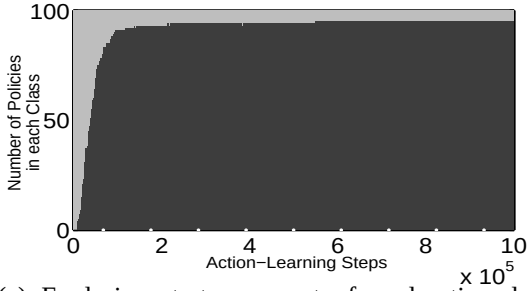
The number of satisficing and optimal policies learnt by MCESP-CD for the range of parameters tried are summarised in table 6.2. The figures presented are the number of satisficing policies, shown in bold, and the number of optimal reactive policies (bracketed figures) at one million action learning steps. Figure 6.4 presents plots for three of the parameter combinations tried, showing the change in categorisation of the poli-



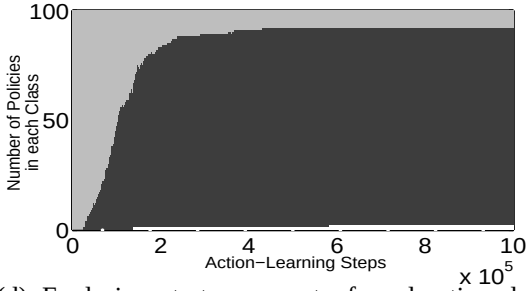
(a) Exploring starts, level of exploration is constant. Observation-action values initiated to zero.



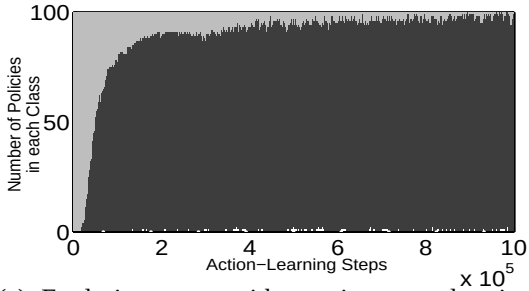
(b) Exploring starts, level of exploration is constant. Observation-action values initiated to Gaussian values.



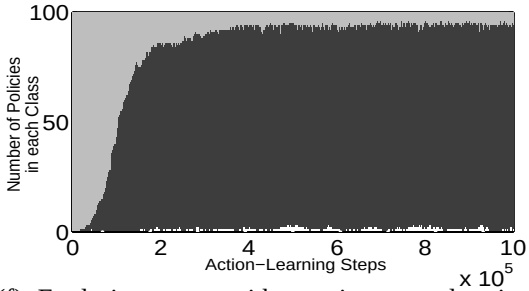
(c) Exploring starts, amount of exploration decayed linearly to zero by 750,000 action-learning steps. Observation-action values initiated to zero.



(d) Exploring starts, amount of exploration decayed linearly to zero by 750,000 action-learning steps. Observation-action values initiated to Gaussian values.



(e) Exploring starts with consistent exploration, level of exploration is constant. Observation-action values initiated to zero.



(f) Exploring starts with consistent exploration, level of exploration is constant. Observation-action values initiated to Gaussian values.

Figure 6.3: Categorisation of policies for first-visit Monte Carlo algorithms versus action-learning steps

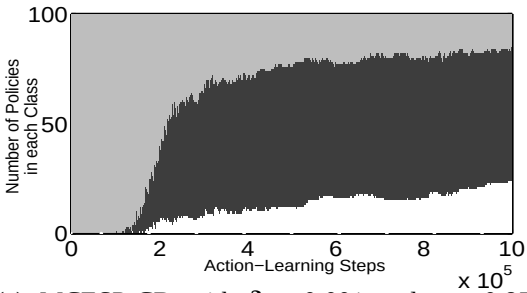
(a) Observation-action values initiated to zero.

δ \ γ	0.75	0.85	0.9	0.95	0.99	1.0
0.0000001	**0** (0)	**86** (14)	**94** (9)	**86** (0)	**91** (0)	**86** (1)
0.00001	**0** (0)	**89** (22)	**92** (6)	**90** (3)	**88** (0)	**87** (0)
0.001	**1** (0)	**85** (24)	**98** (11)	**91** (1)	**89** (0)	**89** (1)
0.01	**0** (0)	**85** (17)	**95** (6)	**96** (4)	**86** (0)	**87** (0)
0.1	**0** (0)	**76** (10)	**95** (9)	**91** (2)	**80** (0)	**85** (0)

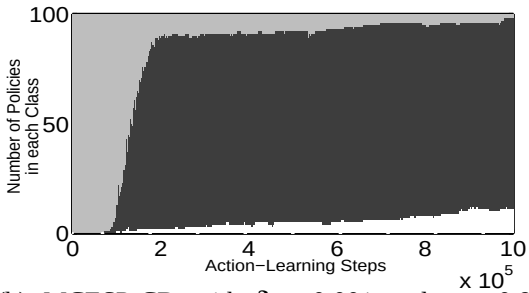
(b) Observation-action values initiated from Gaussian distribution.

δ \ γ	0.75	0.85	0.9	0.95	0.99	1.0
0.0000001	**0** (0)	**31** (0)	**80** (4)	**92** (1)	**87** (1)	**89** (0)
0.00001	**0** (0)	**27** (1)	**75** (1)	**94** (6)	**95** (0)	**91** (1)
0.001	**0** (0)	**24** (1)	**84** (5)	**92** (1)	**87** (0)	**86** (0)
0.01	**0** (0)	**20** (1)	**77** (1)	**89** (3)	**87** (0)	**85** (1)
0.1	**0** (0)	**15** (0)	**79** (2)	**94** (1)	**90** (0)	**92** (1)

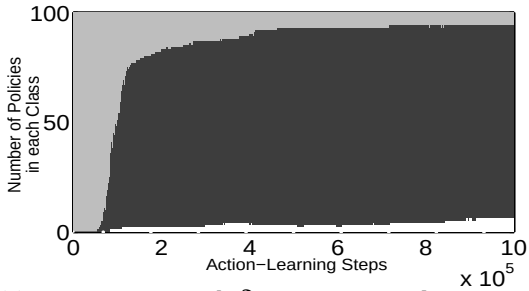
Table 6.2: Summary of satisficing and optimal policies learnt by MCESP-CD at one million action learning steps. The figures in bold are the number of satisficing policies learnt. The bracketed figures indicate how many of these satisficing policies were also optimal reactive policies. All figures are out of one hundred



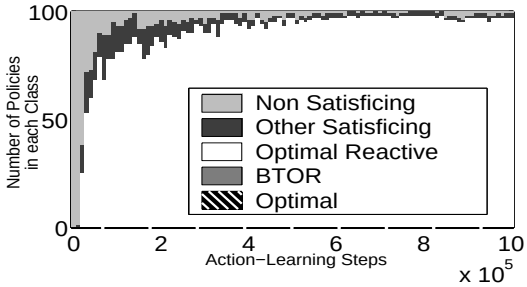
(a) MCESP-CD with $\delta = 0.001$ and $\gamma = 0.85$. Observation-action values initiated to zero.



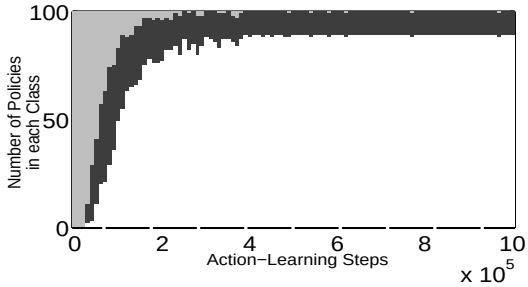
(b) MCESP-CD with $\delta = 0.001$ and $\gamma = 0.9$. Observation-action values initiated to zero.



(c) MCESP-CD with $\delta = 0.000\,01$ and $\gamma = 0.95$. Observation-action values initiated from Gaussian



(d) SARSA(λ) with $\lambda = 0.9$ and $\alpha = 0.05$. Observation-action values initiated to zero. For action selection ε-greedy is used.



(e) SARSA(λ) with $\lambda = 0.99$ and $\alpha = 0.02$. Observation-action values initiated to zero. For action selection ε-greedy is used.

Figure 6.4: Categorisation of policies versus action-learning steps for three of the best performing sets of parameters for MCESP-CD. Also shown for comparison are the results for SARSA(λ) on the same problem. These latter two plots are reproduced from section 4.3

(a) first-visit Monte Carlo algorithm

1st visit Monte Carlo	observation-action values initiated to zero	observation-action values initiated from Gaussian distribution
exploring starts, level of exploration is constant	23 / 10 / 0	9 / 4 / 0
exploring starts, amount of exploration decayed linearly to zero	10 / 0 / 0	3 / 2 / 0
consistent exploration version of exploring starts	3 / 0 / 0	2 / 1 / 0

(b) MCESP with observation-action values initiated to zero.

δ \ γ	0.75	0.85	0.9	0.95	0.99	1.0
0.0000001	0 / 0 / 0	25 / 14 / 1	29 / 9 / 0	7 / 0 / 1	3 / 0 / 0	1 / 1 / 0
0.00001	0 / 0 / 0	34 / 22 / 1	24 / 6 / 1	13 / 3 / 0	3 / 0 / 0	0 / 0 / 0
0.001	0 / 0 / 0	36 / 24 / 1	30 / 11 / 0	5 / 1 / 0	0 / 0 / 0	3 / 1 / 0
0.01	0 / 0 / 0	27 / 17 / 1	21 / 6 / 0	10 / 4 / 0	2 / 0 / 0	1 / 0 / 1
0.1	0 / 0 / 0	23 / 10 / 1	18 / 9 / 0	9 / 2 / 0	0 / 0 / 0	1 / 0 / 0

(c) MCESP with observation-action values initiated from Gaussian distribution.

δ \ γ	0.75	0.85	0.9	0.95	0.99	1.0
0.0000001	0 / 0 / 0	4 / 0 / 1	8 / 4 / 0	6 / 1 / 0	2 / 1 / 0	1 / 0 / 0
0.00001	0 / 0 / 0	4 / 1 / 0	10 / 1 / 0	12 / 6 / 0	1 / 0 / 0	2 / 1 / 0
0.001	0 / 0 / 0	2 / 1 / 1	17 / 5 / 1	11 / 1 / 0	1 / 0 / 0	1 / 0 / 0
0.01	0 / 0 / 0	2 / 1 / 0	7 / 1 / 0	11 / 3 / 0	3 / 0 / 0	2 / 1 / 0
0.1	0 / 0 / 0	0 / 0 / 0	4 / 2 / 0	9 / 1 / 1	1 / 0 / 0	1 / 1 / 0

Table 6.3: Number of local minimum policies learnt by MCESP-CD and first-visit Monte Carlo algorithms at one million action learning steps. Figures, which are out of one hundred, are (i) total of all locally minimum policies, (ii) number that are also globally minimum (that is, optimal reactive), and (iii) number in (i) that are non-satisficing locally minimum policies

cies against action-learning steps. The three plots selected show a sample of the best performing parameter combinations for MCESP-CD on this problem.

The results for MCESP-CD in table 6.2 indicate that this algorithm is sensitive to the values of γ. There is a clear peak in the number of satisficing policies learnt around $\gamma = 0.9$ when observation-action values were initially zero, or around $\gamma = 0.95$ where observation-action values are initiated from a Gaussian distribution. In both cases the number of satisficing solutions falls off dramatically for $\gamma = 0.75$ and drops slightly for γ values close or equal to 1. One might suspect that the discount rate γ might interact with the threshold value (δ) used, with the difference between competing observation-action values changing as γ changes and thus requiring a different value of δ. This, however, does not appear to be the case as the value of γ at which the peak in satisficing policies occurs does not change significantly over the range of δ values tried.

The number of satisficing policies learnt appears little influenced by the value of δ. The value of δ does however appear to affect the number of optimal reactive policies that have been learnt, with a peak around $\delta = 0.001$, then falling off as δ approaches either 0.1 or 0.000 000 1. A similar effect is observed in table 6.3 which presents the results of testing the policies to see if they are locally minimum.

Results are presented in table 6.3 for all of the policies learnt above, both by the first-visit Monte Carlo algorithms and by MCESP-CD. Three figures are presented for each combination of parameters tried. The first is the number of policies out of one hundred that are locally minimum policies. The second is the number that are also globally optimal, that is, optimal reactive policies. The final figure is the number of locally minimum policies that are non-satisficing policies. (Each of the non-satisficing locally minimum policies observed was of the same form as that described in section 5.2.5.4.)

A peak in the total number of locally/globally-minimum policies occurs around $\delta = 0.001$ for both sets of initial observation-action values, with a fall-off in the number learnt as δ is both increased and decreased. The fall-off in the number of locally minimum policies as δ increases could be caused by threshold becoming to coarse to select alternative policies which only represent a small improvement.

6.5 Discussion

The results for both first-visit Monte Carlo algorithms, and for MCESP-CD are disappointing in comparison to those obtained for SARSA(λ), plots 6.4(d) and 6.4(e). Of the parameter values tried none learnt anywhere near the number of optimal reactive policies as achieved for either of the two sets of trials shown for SARSA(λ). In addition the number of satisficing policies is generally worse for both first-visit Monte Carlo and MCESP-CD.

MCESP-CD is guaranteed to converge to δ-locally optimal policies, where a policy is δ-locally optimal if and only if $Q^{\pi}_{o,\pi(o)} + \delta \geq Q^{\pi}_{o,a}$ for all observations o and actions a. Despite this guarantee the number of policies that are locally optimal is very small, even for the smallest value of δ. MCESP-CD learnt a maximum of 36 out of 100 locally minimum policies compared to 99 or 94 out of 100 for each of the two SARSA(λ) examples.

This is not to say that MCESP-CD does not converge given sufficient time. Figure 6.5 shows the categorisation of policies for trials that lasted ten million action-learning steps. This plot, which is based on the same parameters as figure 6.4(b), shows close to 100% convergence to satisficing policies after six million action-learning steps. Despite the decaying α values there appears to be a degree of noise on this plot suggesting that some satisficing policies are not stable. The variation in the number of optimal policies, with assorted peaks and troughs, indicates that optimal policies prove even less stable. An analysis of the number of locally-minimum policies after ten million action learning steps revealed that only 37 out of 100 are locally-minimum, with 18 of those 37 being optimal policies.

Figure 6.5 illustrates the slowness with which a number of policies appear to converge when using MCESP-CD compared to SARSA(λ). To test if the chosen learning rate schedule $\alpha = \frac{0.1}{1+c(o,a)}$, could be the cause of this effect, we ran a limited number of experiments with α fixed at 0.1. An initially fixed value for α could form part of a valid Robbins-Monroe sequence provided at some point the sequence of values start to decay in the appropriate manner, that is, the initial fixed sequence of values form some

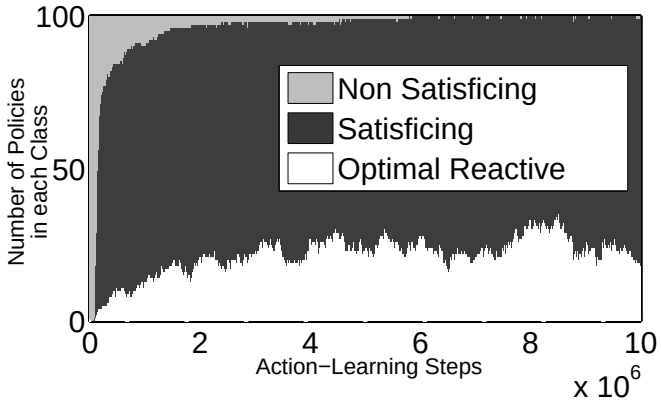


Figure 6.5: Categorisation of policies for MCESP-CD for set of trials extended out to ten million action-learning steps. Parameter values: $\delta = 0.001$, $\gamma = 0.9$, and observation-action values initiated to zero

constant offset. In this case we have taken this reasoning to an extreme by considering that the decay in sequence values starts sometime after one million action-learning steps. We take this stance as the use of a very slowly decaying learning rate should speed learning and thus affect convergence. A fixed rate represents an upper bound on this. Experiments were run for γ values of 0.85, 0.9, and 0.95, δ values of 0.001 and 0.000 01, with observation-action values initiated either all to zero, or drawn randomly from a Gaussian distribution with mean zero and standard deviation of five. The results obtained showed marginal improvement over those achieved with the schedule of α used above. For example compare figure 6.6 with 6.4(b). Both of these plots are for identical values of δ, γ and initial observation-action values. The lack of significant improvement suggest that the slowness of convergence is more fundamental and possibly linked to the way that the MCESP-CD algorithm operates.

MCESP-CD appears somewhat sensitive to the parameter values selected, as the best results were found for specific combinations of δ, γ and initial conditions. This sensitivity coupled with a slowness to converge makes it difficult to recommend it over SARSA(λ) for practical application. Nor does either of the undiscounted first-visit Monte Carlo approaches tried offer any apparent advantages on this type of problem.

An additional issue associated with any Monte Carlo approach is the need to fix an arbitrary episode length to ensure that the updating of action values and policies occurs.

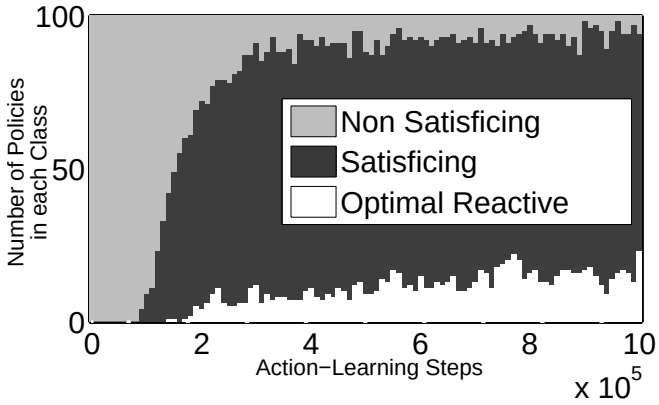


Figure 6.6: Categorisation of policies for MCESP-CD with fixed learning rate of $\alpha = 0.1$ versus action-learning steps. Other parameter values: $\delta = 0.001$, $\gamma = 0.9$, and observation-action values initiated to zero

Setting the maximum episode length to a reasonable value requires a designer to have some idea of the total length of episodes in the final solution. Setting it too short will prevent solutions from being explored, and in the extreme case may result in a failure to learn any solutions. Setting it too long will slow learning.

Finally, the concept of exploring starts does not fit very well with learning in an embedded, embodied agent. Such an agent is not going to be able to easily relocate itself to a new random starting location at the start of each episode. For such an agent a learning approach that avoids the need for discontinuities in its location would be preferable.

6.6 Chapter Summary

In this chapter we examined Monte Carlo approaches. Pendrith and McGarity [1998, 1997] showed that optimal reactive POMDP policies are equilibrium points when using a first-visit Monte Carlo approach with undiscounted returns. However, we demonstrate that such policies are not stable if exploring starts is used to search over alternative actions. We then develop the idea of Consistent Exploration, which improves the stability of existing optimal, reactive, POMDP policies when exploring starts is being used. Unfortunately, results for learning policies from scratch, using a Consistent

Exploration version of the first-visit Monte Carlo algorithm, proved disappointing.

We also demonstrate empirically results for MCESP-CD Perkins [2002b]. This algorithm redefines observation-action values such that some theoretical guarantees can be shown on POMDPs. Despite these guarantees the results obtained are still very disappointing when compared to SARSA(λ).

Chapter 7

Consistent Exploration Approach

In the previous chapter we considered Monte Carlo approaches to learning reactive policies for partially observable Markov decision processes (POMDPs). Despite claims of nice theoretical properties for these approaches, their overall performance on the Sutton's Grid World task was marginally worse than results obtained using SARSA(λ). In addition, as discussed in section 6.5, the Exploring Starts approach is not particularly practical when considering a real world robotic application. The conclusion resulting from chapter 6 is that an ideal algorithm for our task would combine the best feature of SARSA(λ) and Monte Carlo Exploring Starts for POMDPs (MCESP) approaches. In this chapter we set out an algorithm that meets these criteria and examine its comparative performance to algorithms from previous chapters.

7.1 Designing an Improved Algorithm

7.1.1 Requirements

The characteristics of SARSA(λ) and MCESP that we would wish to combine can be summarised as:

(i) Learning whilst moving through the world in a continuous manner, that is,
   (a) no discontinuities in the agent's location, such as those required by exploring starts,
   (b) updates not delayed until the end of episodes.

(ii) Minimise the number of parameters and parameter schedules that need to be specified, especially those that require knowledge of the task to tune effectively, that is, fixed value of τ when using Gibbs/Boltzmann distribution for action selection, and setting a maximum episode length for "Monte Carlo" approaches.

(iii) Ensure that backed up action-values are as accurate as possible by
   (a) using Monte Carlo returns, and
   (b) ensuring exploration is consistent, that is, employing a version of Consistent Exploration as described in section 6.1.2.

Items (i) and (ii) are most easily achieved though use of a temporal difference style algorithm, such as SARSA, Q-learning, or SARSA(λ) or Watkins's Q(λ). Given the results from chapter 4, which demonstrate the superior performance of methods that utilise eligibility traces on partially observable problems, then SARSA(λ) or Watkins's Q(λ) would seem preferable. With regard to item (iii,a), Sutton and Barto [1998, p.188] indicates that when λ is close to the value of 1.0 then eligibility traces approximate Monte Carlo style returns. Thus we can achieve this point through judicious parameter selection. This leaves the issue of achieving point (iii,b), ensuring exploration is consistent.

In section 6.1.2 we consider a modified version of the first visit Monte Carlo exploring starts algorithm. This modified algorithm records the initial observation and exploratory action and then selects the same exploratory action every time it re-encounters the starting observation. We generalise this consistent exploration by considering a reinforcement learning algorithm that can select several exploratory actions during an episode, but remain consistent with itself between exploratory actions. For example, having selected exploratory action a_1 for observation o_1 it then follows the modified

policy $\pi \leftarrow (o_1, a_1)$ until at random it selects a new exploratory action a_2 for observation o_2. From that point on it then follows policy $\pi \leftarrow (o_2, a_2)$ where π is the original policy.

For modified first visit Monte Carlo exploring starts algorithm, and MCESP, both of which also implement consistent exploration, the length of an episode had to be artificially constrained to ensure that its policy eventually changed, especially in the event of it getting stuck in a endless loop. Our generalisation of Consistent Exploration allows the observation-action pair being explored to change during an episode, and thus avoids the necessity of applying such a cap on episode length and thus makes our proposed algorithm more flexible.

7.1.2 Consistent exploration Q(λ)

To construct our new algorithm we start with a version of Watkins's Q(λ) that we have already slightly modified. This modified algorithm is presented in appendix F. Our modified version of Watkins's Q(λ), separates the updating of Q-values and the updating of eligibility trace values into two separate loops. This is in contrast to the standard version as presented in Sutton and Barto [1998, p.184], and repeated in Algorithm 9 in appendix F, where Q-values and eligibility trace values are updated simultaneously. This modification was motivated by a requirement for a version of Watkins's Q(λ) that was equivalent to Q-learning for $\lambda = 0$. The modified form has the apparent advantage that selection of the next action to be executed is done using the most up to date Q-values. However, in various empirical tests that we ran, this change does not result in consistently better performance when compared to the standard version, and for some tasks the performance was worse. Additional background information is provided in appendix F, however the details are not critical to the aims of this thesis. We modify this algorithm further to incorporate the idea of consistent exploration, resulting in our Consistent Exploration Q(λ) algorithm, or CEQ(λ), as presented in Algorithm 4.

In CEQ(λ), consistent exploration is implemented by storing the last observation o^e for which an exploratory action a^e was executed. These two variables are used to modify

the policy such that the policy being followed by the agent is $\pi \leftarrow (o^e, a^e)$. These two variables, a^e and o^e, are updated every time an action is chosen which does not match $\pi \leftarrow (o^e, a^e)$. When such an exploratory move occurs the eligibility trace is also reset. Reseting the eligibility trace ensures that values are only backed-up over observation-action pairs that occurred when a particular modified policy was being followed, that is, given that exploratory action a_1 was selected for observation o_1 then values for $Q(o_1, a_1)$ would be updated based on rewards which occurred whilst the policy remains $\pi \leftarrow (o_1, a_1)$, but not when say policy $\pi \leftarrow (o_2, a_2)$ is subsequently followed. It is for this effect that we chose a version of Watkins's Q(λ) as our base, as the backed-up values should more accurately reflect the values of the modified policy being followed.

7.1.3 Comparison with SARSA(λ)

Our contention is that CEQ(λ), in receiving more accurate estimates on the value of policy changes, will be more reliable and robust than SARSA(λ) in learning satisficing policies for POMDPs where reactive policies exist.

To test the above statement we compare CEQ(λ)'s performance with that of SARSA(λ) over a range of values of the three main parameters; the learning rate α, discount rate γ and eligibility trace value λ. We run these comparisons on three different POMDPs in order to establish an indication of the generality of the results obtained.

7.1.3.1 Parameter values

We compare both algorithms over all combinations of the following parameter values:

- learning rate (α) values of 0.01, 0.02, 0.05, 0.1 and 0.3;
- discount rate (γ) values of 0.8, 0.9, 0.99 and 1.0;
- eligibility trace (λ) values of 0.8, 0.9 0.99 and 1.0.

Algorithm 4 Our proposed POMDP reinforcement learning algorithm: Consistent Exploration Q-learning or CEQ(λ). Algorithm shown using replacing eligibility traces.

Initialise $Q(o,a) \leftarrow$ arbitrary values
Repeat (for each episode):
  Initialise $e(o,a) = 0$ for all o, a
  Initialise start state
  get observation o and select a using policy derived from Q (for example, ε-greedy)
  if $a \neq \arg\max_b Q(o,b)$, then $a^e = a$; $o^e = o$
    else $a^e = o^e =$ none
  Repeat (for each step of episode):
    take action a, obtain r and observe o'
    if $o' = o^e$, then $a^* \leftarrow a^e$
      else $a^* \leftarrow \arg\max_b Q(o',b)$
    $\delta = r + \gamma Q(o',a^*) - Q(o,a)$
    $e(o,a) \leftarrow 1$
    for all o, a:
      $Q(o,a) \leftarrow Q(o,a) + \alpha\delta e(o,a)$
    choose a' for o' using policy derived from updated Q (for example, ε-greedy)
    for all o, a:
      if $a' = a^*$, then $e(o,a) \leftarrow \gamma\lambda e(o,a)$
        else $e(o,a) \leftarrow 0$
    if $a' \neq a^*$ then $a^e = a'$, $o^e = o'$
    $o \leftarrow o'$; $a \leftarrow a'$
  until terminal state is reached

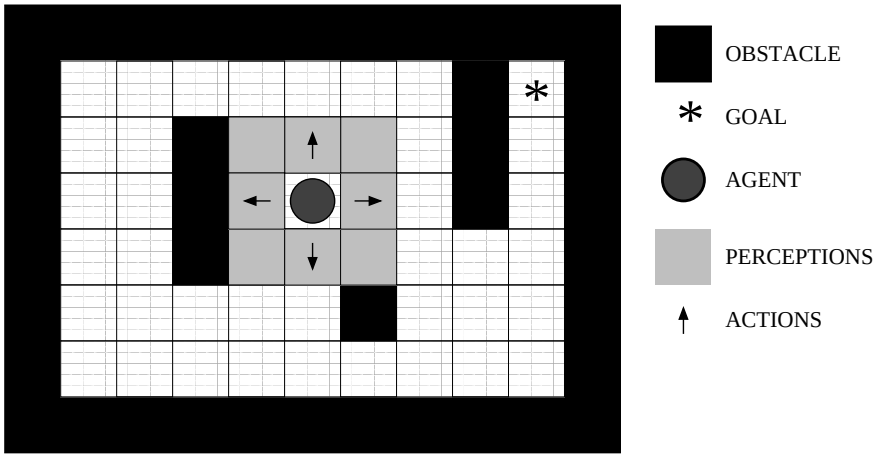


Figure 7.1: Sutton's Grid World and agent with limited local perception [Littman, 1994]

For each combination of parameter values one hundred independent trials were run allowing us to record the number of times each algorithm converged to a satisficing policy and the mean total steps required by each satisficing policy. The length of each trial was a million action-learning steps.

At the start of each trial the observation-action values $Q(o,a)$ were initiated to random values drawn from a Gaussian distribution with mean zero and standard deviation of five. Initial policies were based on these Q values $\pi \leftarrow \arg\max_a Q(o,a)$. Actions were selected in accordance with ε-greedy exploration [Sutton and Barto, 1998, p.122], with ε initially valued at 0.2 and decaying linearly to zero by the $500,000^{th}$ action-learning step. Both algorithms used replacing traces [Sutton and Barto, 1998, p.186] for their eligibility trace.

7.1.3.2 First test problem

The first two test problems use Sutton's Grid World as described previously in section 4.1.1.2 and shown in figure 7.1. This world is used in combination with two agents which have differing perceptual abilities: the Fixed Eight Squares Agent and Active Perception Agent as described in section 4.3.

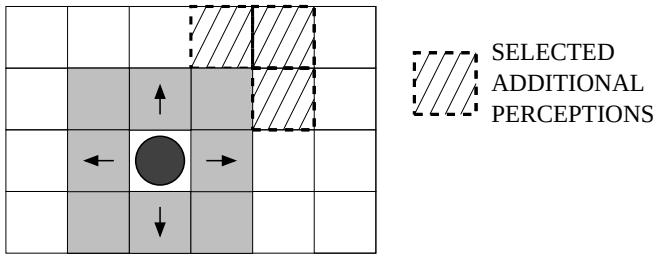


Figure 7.2: Agent selecting perceptual action “look North East”

The first test problem, using the Fixed Eight Squares Agent, is of interest as it is sufficiently complex to contain examples of the different problems that can occur when applying reinforcement learning to POMDPs. There are multiple locations where the agent obtains the same observation, and each of these aliased locations can present different challenges to reinforcement learning algorithms; see section 4.1.1.2 for more details. In addition, as found in section 5.2.5.4, there exists a non satisficing, locally minimum policy[1] for this problem to which a reinforcement learning algorithm could converge. Loch and Singh [1998] have previously demonstrated that SARSA(λ) is capable of learning satisficing reactive policies for this combination of agent and world.

7.1.3.3 Second test problem

The second test problem uses the Active Perception Agent as shown in figure 7.2. This second test world presents a more complex problem as both the input space and action space are significantly increased. In addition, the act of being able to control the agent’s perception increases the number of locations where perceptual aliasing could occur.

[1] A locally minimum policy is a policy for which any *single* change in association between observations and actions results a policy whose value is worse.

7.1.3.4 Third test problem

The third test problem is Wilson's Woods 7 as shown in figure 7.3. Unlike Sutton's Grid World which is used in the two problems above, Wilson's Woods 7 has no edge, it is instead toroidal, so for example moving off the bottom edge brings you back onto the top, and moving off the left edge brings you round to right, etc.. The agent is similar to that in the first test problem above as its perception is limited to only the eight squares which are immediately adjacent to it. There however the similarity ends, as the observations that it receives indicate not only the presence of obstacles in those eight squares, but also the presence of goals. The agent also has more physical actions available to it as it can move in any of the eight compass directions.

On reaching a goal the agent receives a reward of $+1.0$ and is relocated to a random location in the world that does not contain a obstacle. In the version of Wilson's Woods 7 that we use here, the execution of an action which does not result in the agent immediately arriving at the goal is not penalised. If the agent attempts an action which moves it towards an obstacle it remains where it currently is, and again receives no penalty or reward.

7.1.4 Results

7.1.4.1 First test problem

Table 7.1 shows the number of satisficing policies learnt by the two learning algorithms on the first test problem. The values shown in table 7.1 are the number of policies, out of one hundred, that were satisficing policies after one million action-learning steps had passed. For each combination of parameters two values are shown. The top left value, shown in bold font, is the number of satisficing policies learnt by CEQ(λ). The lower right value is the number of satisficing policies learnt by SARSA(λ). As can be seen from this table CEQ(λ) is consistently better than SARSA(λ) on this test problem, across all parameter combinations tried. More importantly CEQ(λ) shows a degree of

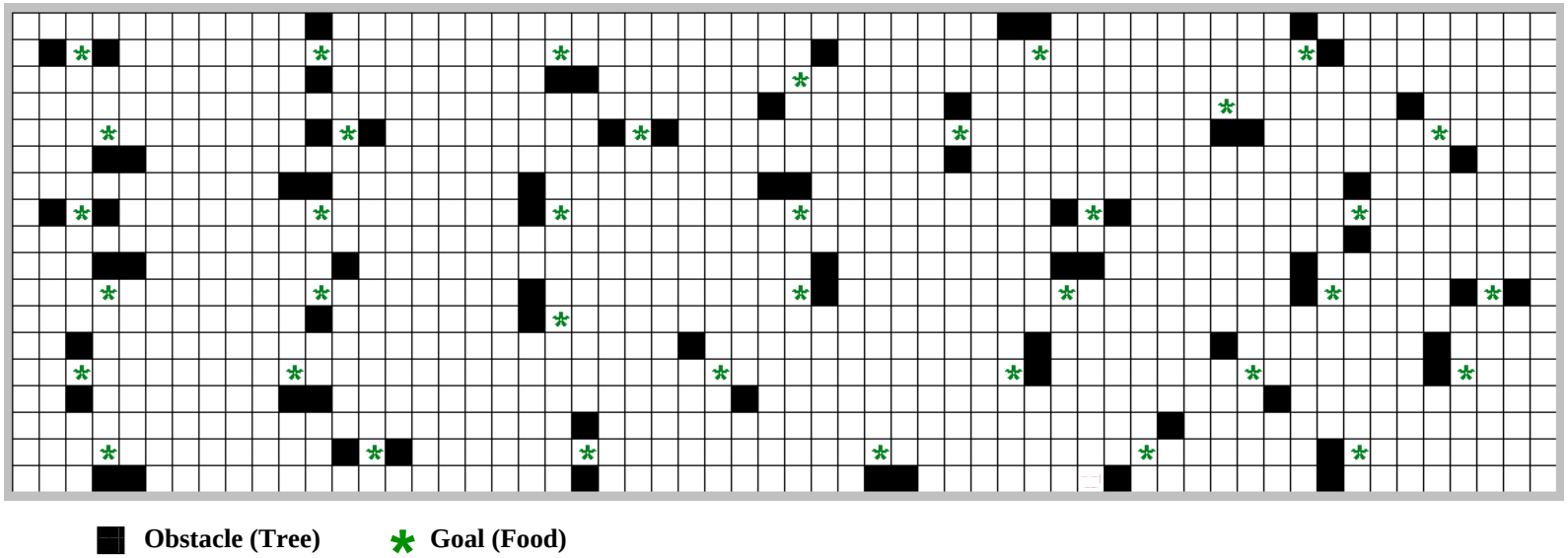


Figure 7.3: Wilson's Woods 7 grid world [Littman, 1994, 1992; Wilson, 1985]. World is toroidal as the top edge is connected to the bottom and the left edge is joined to the right

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**75** / 55	**85** / 56	**100** / 78	**100** / 86	**92** / 35
0.9	**80** / 51	**92** / 83	**100** / 92	**100** / 67	**97** / 43
0.99	**91** / 65	**96** / 93	**99** / 98	**100** / 87	**100** / 47
1.0	**83** / 80	**98** / 95	**100** / 99	**100** / 91	**100** / 57

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**89** / 78	**98** / 94	**100** / 96	**100** / 86	**94** / 33
0.9	**88** / 83	**99** / 95	**100** / 98	**100** / 85	**100** / 48
0.99	**94** / 77	**99** / 95	**100** / 100	**100** / 85	**100** / 54
1.0	**96** / 83	**100** / 99	**100** / 100	**100** / 99	**100** / 50

key:

CEQ(λ)

SARSA(λ)

(c) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**95** / 89	**100** / 99	**100** / 100	**100** / 90	**97** / 40
0.9	**98** / 92	**98** / 97	**100** / 98	**100** / 85	**100** / 54
0.99	**100** / 91	**100** / 94	**100** / 93	**100** / 72	**100** / 55
1.0	**97** / 91	**100** / 89	**100** / 86	**100** / 74	**99** / 54

(d) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**94** / 87	**100** / 97	**100** / 99	**100** / 95	**100** / 38
0.9	**95** / 88	**99** / 99	**100** / 98	**100** / 86	**100** / 55
0.99	**99** / 92	**99** / 94	**100** / 86	**100** / 83	**98** / 41
1.0	**99** / 49	**100** / 32	**100** / 14	**100** / 3	**99** / 0

Table 7.1: Number of satisficing policies out of 100 for 1^{st} test problem. Top left bold figures CEQ(λ), bottom right SARSA(λ)

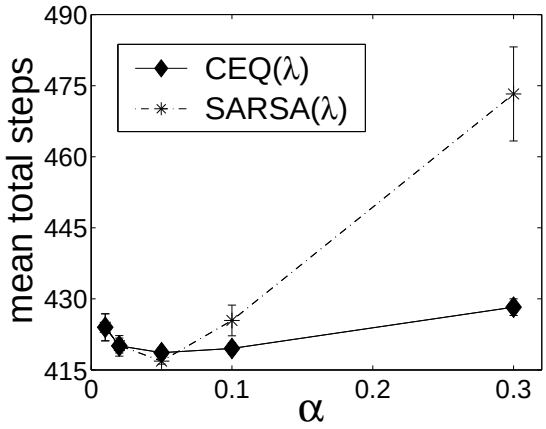


Figure 7.4: Mean total steps of satisficing policies for 1^{st} test problem versus α, with $\lambda = 0.9$, $\gamma = 0.99$. Error bars show 95% confidence intervals

robustness with respect to the variation of parameter values, with nearly all policies at one million action-learning steps being satisficing. This is in contrast to SARSA(λ) whose performance varies considerably, with clumps of good performance when either λ or γ approach 1.0 but not when both do. The mean number of satisficing policies for CEQ(λ) across the full range of parameter values is 97.6 with a standard deviation (std) of 4.8. Across the same range SARSA(λ) has a mean 75.9 (std 24.5). Student's t-test for unequal variance indicates that the probability that these results are drawn from the same distribution is very small; $p = 1.5 \times 10^{-11}$. This indicates that the result is statistically significant.

As the policies learnt are not necessarily optimal we examine the quality of the satisficing policies learnt to see if either algorithm is, on average, significantly worse than the other. The evaluation of a policy consists of placing the agent at every possible location in the grid world where there is no obstacle, and counting how many steps it takes to reach the goal state. The total steps taken by the agent is then summed up over all start states. Note that we only include policies that are satisficing.

Figure 7.4 shows a plot of the mean total steps for CEQ(λ) and SARSA(λ) as α is varied. This plot, which is for $\gamma = 0.99$ and $\lambda = 0.9$, is typical of the results seen for other combinations of these two parameters. The optimal reactive policy takes 416 steps

[Littman, 1994]. Both algorithms on average learn satisficing policies that are close to this level with SARSA(λ) performing marginally better for $\alpha = 0.05$, but performing significantly worse at higher α values where the number of satisficing policies that it has learnt also drops significantly.

The mean number of steps for all satisficing policies across the entire set of parameter values tested is 423.6 (std 14.4) for CEQ(λ), and 438.0 (std 39.6) for SARSA(λ). These averages are statistically significant with Student's t-test for unequal variance reporting the probability of them coming from the same distribution as being 1×10^{-187}. Tables which present a detailed breakdown of the mean total steps for each parameter combination can be found in appendix N.

7.1.4.2 Second test problem

Results for the second test problem, table 7.2, show a similar pattern to those for the first with the best performance for SARSA(λ), as measured in terms of the number of satisficing policies, peaking when the value of either λ or γ approaches one, but not when both do. It also shows a peak in performance when $\alpha = 0.02$. CEQ(λ) again appears consistent across the range of parameters learning close to 100% satisficing policies with the exception of the lowest α value. Examining the raw data for CEQ(λ) when $\alpha = 0.01$ reveals that at this low level of α the policies converge much slower and many have yet to converge after one million action learning steps. It appears that these policies would have eventually converged to satisficing solutions had the trials been extended.

Excluding $\alpha = 0.01$ the mean number of satisficing policies learnt by CEQ(λ) is 99.2 (std 1.2) compared to mean of 90.0 (std 9.6) for SARSA(λ). Student's t-test for unequal variance indicates that these averages are statistically significant with the probability of the two sets being drawn from the same distribution being $p = 2 \times 10^{-10}$.

Including the results for $\alpha = 0.01$ these figures drop to mean 88.7 (std 24.4) for CEQ(λ) and mean 89.5 (std 10.7) for SARSA(λ). It appears in that in including the results for

key:

CEQ(λ)

SARSA(λ)

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**6** / 54	**94** / 100	**96** / 94	**99** / 78	**99** / 82
0.9	**9** / 68	**98** / 97	**98** / 94	**100** / 84	**100** / 88
0.99	**6** / 75	**95** / 96	**100** / 99	**100** / 91	**100** / 89
1.0	**14** / 70	**98** / 95	**99** / 99	**100** / 92	**100** / 86

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**27** / 85	**98** / 100	**99** / 87	**100** / 86	**100** / 80
0.9	**26** / 97	**100** / 99	**97** / 90	**100** / 91	**99** / 87
0.99	**47** / 97	**98** / 99	**98** / 100	**100** / 94	**100** / 92
1.0	**59** / 94	**99** / 99	**97** / 96	**99** / 95	**99** / 87

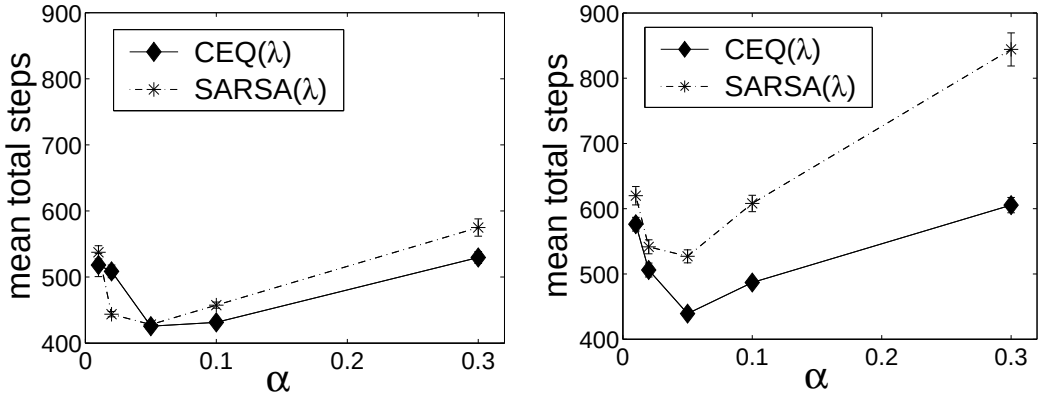
(c) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**51** / 100	**100** / 100	**99** / 96	**99** / 92	**100** / 84
0.9	**71** / 100	**100** / 100	**100** / 95	**100** / 87	**100** / 84
0.99	**79** / 93	**100** / 96	**100** / 94	**100** / 95	**99** / 78
1.0	**74** / 92	**100** / 94	**100** / 96	**99** / 92	**100** / 80

(d) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**50** / 100	**100** / 100	**99** / 90	**100** / 84	**99** / 83
0.9	**73** / 100	**100** / 100	**99** / 96	**99** / 94	**100** / 84
0.99	**78** / 92	**100** / 96	**99** / 95	**99** / 93	**100** / 69
1.0	**81** / 77	**100** / 70	**99** / 83	**100** / 74	**99** / 43

Table 7.2: Number of satisficing policies out of 100 for 2^{nd} test problem. Top left bold figures CEQ(λ), bottom right SARSA(λ)



(a) $\lambda = 0.9, \gamma = 0.9$ (b) $\lambda = 1.0, \gamma = 0.99$

Figure 7.5: Mean total steps of satisficing policies for 2^{nd} test problem versus α. Error bars show 95% confidence intervals

$\alpha = 0.01$ CEQ(λ) performs worse on average that SARSA(λ), however the difference is not statistically significant with Student's t-test for unequal variance indicating that two sets are probably drawn from the same distribution; $p = 0.80$.

Figure 7.5(a) shows the mean total steps for the satisficing policies that have been learnt by CEQ(λ) and SARSA(λ) when $\lambda = 0.9$ and $\gamma = 0.9$. The mean total steps are plotted against the values of α that have been tried. Both algorithms learn satisficing policies with the least number of steps around $\alpha = 0.05$, with CEQ(λ) performing better than SARSA(λ) for the higher values of α and performing worse for lower values of α. This plot is similar to those for other combinations of λ and γ on this problem. The only exception is when the values of both λ and γ are some combination of 0.99 or 1.0. For these values the satisficing policies learnt by CEQ(λ) take on average fewer steps to reach the goal than the policies learn by SARSA(λ). This is shown in figure 7.5(b). Tables of mean total steps for all parameter combinations tried can be found in appendix N.

We believe that the optimal reactive policy for this second test problem takes a total of 414 steps, comprising 8 perceptual actions and 406 physical actions. These results indicate that neither algorithm consistently converges on the optimal reactive policy. Mean total steps of satisficing policies across the full range of parameters is 492.0

(std 72.8) for CEQ(λ) and 533.2 (std 121.7) for SARSA(λ). The difference between these two averages is statistically significant with Student's t-test for unequal variance reporting a probability of 1×10^{-129} that the two sets of results are derived from the same distribution.

7.1.4.3 Third test problem

The results for the third test problem, Wilson's Woods 7, are presented in table 7.3. The most obvious difference to the results obtained on Sutton's Grid World (as used in the first and second test problems) is that the number of satisficing policies reduces dramatically as γ approaches and becomes equal to one. This affects both SARSA(λ) and CEQ(λ), though CEQ(λ) behaves better than SARSA(λ) at $\gamma = 0.99$. The approach of γ towards one also affects the mean total number of steps taken, with the average for both SARSA(λ) and CEQ(λ) increasing, as illustrated for parameter values $\alpha = 0.05$ and $\lambda = 0.9$ in figure 7.6. The effect is not unexpected. Unlike Sutton's Grid World above, in this variation of Wilson's Woods 7 actions are not penalised. In the absence of a penalty each time an action is executed, a discount rate $\gamma < 1$ provides the only signal to the learning algorithm as to the best action in each state. Discounting the goal reward each time a step is taken sets up a gradient that reflects the distance of each state (and observation-action pair) from the goal. In the absence of action penalties this gradient provides the only information on the best action to select. When $\gamma = 1$ there is no discounting of the goal reward and hence no gradient. In this event all states (and state-actions) will be valued the same and hence there is no signal to the learning algorithms as to which is the best action to select.

Examining table 7.3, CEQ(λ) performs better than SARSA(λ) for the higher α values; that is, 0.05, 0.1 and 0.2. For α equal to 0.01 and 0.02 then SARSA(λ) achieves better results (for $\gamma = 0.8$ and 0.9) after one million action-learning steps. However, for these low values of α the number of satisficing solutions learn by either algorithm has not yet approached the levels achieved for the higher values of α. Thus we suspect that, as was observed in second test problem, convergence has not yet been achieved at the time the runs were ended.

key:

CEQ(λ)

SARSA(λ)

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**10** / 22	**54** / 80	**99** / 98	**100** / 100	**100** / 96
0.9	**5** / 16	**57** / 79	**98** / 98	**100** / 98	**100** / 96
0.99	**4** / 18	**55** / 69	**97** / 99	**100** / 100	**99** / 91
1.0	**5** / 16	**47** / 70	**100** / 96	**100** / 99	**100** / 95

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**13** / 30	**66** / 77	**98** / 95	**100** / 98	**98** / 89
0.9	**9** / 26	**59** / 80	**100** / 94	**100** / 97	**100** / 89
0.99	**10** / 25	**55** / 75	**96** / 98	**100** / 98	**100** / 93
1.0	**6** / 26	**58** / 77	**97** / 94	**98** / 98	**99** / 96

(c) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**6** / 4	**27** / 9	**75** / 25	**85** / 26	**85** / 24
0.9	**10** / 6	**50** / 18	**94** / 43	**94** / 29	**94** / 20
0.99	**38** / 17	**73** / 55	**99** / 68	**100** / 51	**97** / 38
1.0	**38** / 44	**75** / 71	**99** / 73	**100** / 67	**94** / 62

(d) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**0** / 0	**0** / 0	**0** / 0	**0** / 0	**1** / 1
0.9	**0** / 0	**0** / 0	**0** / 0	**1** / 0	**2** / 0
0.99	**0** / 0	**0** / 0	**0** / 0	**1** / 0	**2** / 0
1.0	**0** / 0	**0** / 0	**0** / 0	**1** / 0	**3** / 0

Table 7.3: Number of satisficing policies out of 100 for 3^{rd} test problem. Top left bold figures CEQ(λ), bottom right SARSA(λ)

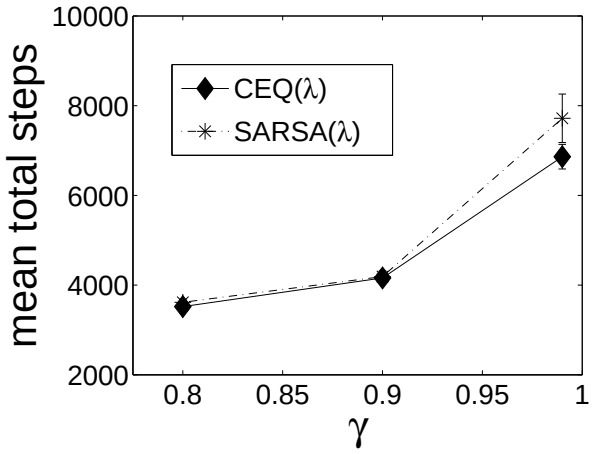


Figure 7.6: Mean total steps of satisficing policies for 3^{nd} test problem versus γ, with $\alpha = 0.05$ and $\lambda = 0.9$. No data points are shown when $\gamma = 1.0$ as no satisficing policies were learnt by either algorithm. Error bars show 95% confidence intervals. Details of the of mean total steps for all parameter combinations tried can be found in appendix N

For the Wilson's Woods 7, the optimal reactive policy is 2895 [Littman, 1994]. For the $\gamma = 0.8$ and range of α values 0.05–0.3, where convergence of both algorithms has occurred, the mean total steps of all satisficing policies are 3297.6 (std 276.6) for CEQ(λ) and 3475.2 (std 324.6) for SARSA(λ). These figures suggest that neither have consistently converged to the optimal reactive policy, though both are reasonably close with CEQ(λ) performing better than SARSA(λ). This result is statistically significant with Student's t-test for unequal variance indicating that the values making up the two averages are very unlikely to be drawn from the same distribution; probability $p = 2 \times 10^{-44}$.

7.1.5 Discussion

The results show that the probability of CEQ(λ) learning satisficing policies is better than SARSA(λ). Although it appears to be slower to converge for low values of α, its performance is generally robust over a wide range of parameter values. Robustness

to variation in parameter values is useful in practical applications, like training a real world robot, where having to repeat training with various parameter settings in order to find those that converge is time consuming and sometimes impractical.

In contrast to SARSA(λ), CEQ(λ)'s performance generally improves as both λ and γ reach one. The one exception is where the reward structure of the problem is such that using undiscounted state-action values, that is, when $\gamma = 1$, provides no information to the agent as to the best action to select. This situation can be avoided by imposing a penalty for each action taken. The accumulation of action penalties will then ensure that undiscounted returns will still provide indication of the path length to the goal.

Although having both γ and λ equal to 1.0 is unusual when applying reinforcement learning to MDPs, theoretical results from Pendrith and McGarity [1998] demonstrate that when learning policies for POMDPs, using *undiscounted Monte Carlo*[2] returns will ensure that the optimal reactive policy (where it exists) lies at a stationary point in terms of observation-action value updates. Thus it is of interest to observe the effect of setting both close to or at 1. As demonstrated by the above two sets of results CEQ(λ) performs consistently well at this extreme point, whereas SARSA(λ) performs badly.

7.1.6 Comparison with MCESP

The main reason for preferring CEQ(λ) over MCESP is that when using CEQ(λ) we avoid the issue of imposing an artificial maximum length on episodes. Consider the case when MCESP explores a poor policy $\pi \leftarrow (o, a)$ which fails to reach the goal and ends up in an infinite loop. MCESP only updates its policy and chooses a new action to explore after each episode. Hence a limit on the episode is required to ensure that loops are broken. CEQ(λ) on the other hand is free to select a new exploratory action at any point and thus will break out of infinite loops of its own accord. Imposing a maximum limit tends to require some knowledge of the potential length of solutions. Setting this limit too high will slow learning, setting this limit too low will restrict

[2] With λ equal 1.0 eligibility traces approximate to a Monte Carlo approach [Sutton and Barto, 1998, p.188].

the solutions that can be explored. We prefer to sidestep this issue by using CEQ(λ) for which no distinct segregation of the problem into episodes is required. In fact the range of problems that CEQ(λ) can be applied to is larger than MCESP as it should be applicable without difficulty to non-episodic tasks.

One would expect policies learnt by MCESP to converge more slowly than CEQ(λ). The former only updates observation-action values and policies at the end of episodes, while the latter updates both after each action-learning step. In addition, the variation of MCESP for which convergence to locally optimal policies is guaranteed, MCESP-CD (see chapter 6.2), only updates its current policy when the difference in observation-action values for competing actions exceeds some given threshold. This further slows policy updates as MCESP-CD may only update its policy after several episodes have been completed.

To test this expectation we compare their performance on the first test problem in section 7.1.3.2. Both algorithms were tested with α values of 0.01 and 0.1, with $\gamma = 1$. For CEQ(λ) $\lambda = 1$ and exploration controlled using ε-greedy; ε initiated at 0.2 and decaying linearly to zero by the $500,000^{th}$ action-learning step. For MCESP policy update threshold is 0.001 and maximum episode length is 50 steps. Policy and observation-action values were initiated at random and for each combination of parameters 100 independent runs were made. Each run was evaluated every 10,000 action-learning steps.

Results are shown in figure 7.7. In this figure we plot the mean total steps against the number of action-learning steps. The mean total steps is calculated across *all* policies. Evaluation of policies that fail to reach the goal from a given start location is curtailed after 1,000 steps. There are a total of 46 starting locations giving a maximum evaluation cost of 46,000 steps for a policy that fails to reaches the goal from every start. The optimal reactive policy takes a total of 416 steps from these 46 start locations. As can be observed from figure 7.7 the average policy learnt by both MCESP and CEQ(λ) quickly converges over time, with MCESP lagging behind CEQ(λ).

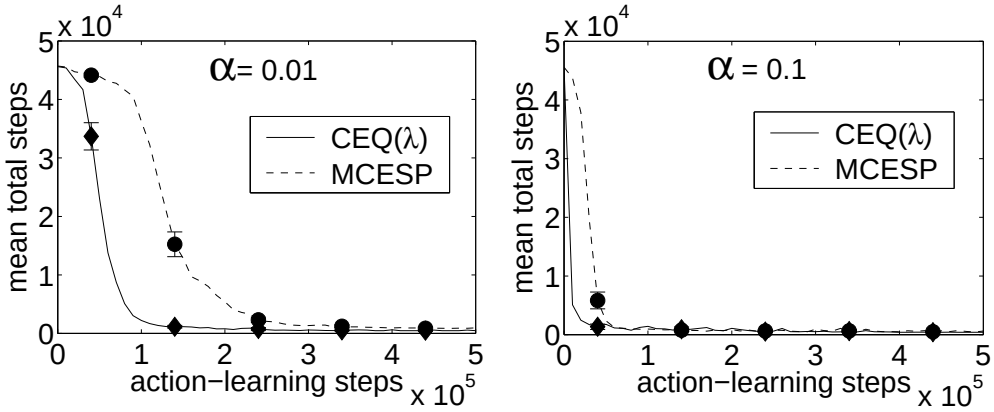


Figure 7.7: Total steps averaged across all policies versus action-learning steps. Error bars show 95% confidence intervals

7.1.7 Experiment conclusions

Where deterministic reactive policies exist in POMDPs, CEQ(λ) tends to learn satisficing policies more reliably than SARSA(λ), though convergence can be marginally slower. The policies learnt by CEQ(λ) converge quicker than those learnt MCESP, and CEQ(λ) avoids the issue of fixing the maximum episode length which is required for MCESP. These results support our more general conjecture that ensuring consistent exploration of observation-action pairs improves the performance of reinforcement learning algorithms on POMDPs.

CEQ(λ) was not specially designed to perform well on the three tasks presented here, so we believe that its performance should generalise to other problems for which deterministic reactive policies exist and also to MDPs. To this end we have successfully run CEQ(λ) on the MDP consisting of Sutton's Grid World and Absolute Position Agent as described in section 4.1.1.

7.2 Some Theoretical Limitations of CEQ(λ)

7.2.1 Optimal reactive policies not guaranteed to be stationary points for $\lambda < 1$ or $\gamma < 1$

Pendrith and McGarity [1998] demonstrate, by means of an example POMDP, that "if a TD(λ) credit-assignment method is used for direct RL of a NMDP[3], then for $\lambda < 1$, it is not guaranteed that there exists an optimal observation-based policy representing a learning equilibrium." That is to say that TD(λ) methods will not be capable of settling on the optimal reactive policy.

The example is constructed such that it applies equally to methods that use replacing or accumulating eligibility traces and methods that use on-policy or off-policy backups. A cursory examination also shows that the addition of Consistent Exploration does not affect the results drawn from the example; thus this result will hold for CEQ(λ).

The fact that this result holds for CEQ(λ) is not sufficient grounds, in our opinion, to abandon this algorithm. First, unlike SARSA(λ), CEQ(λ) shows excellent results for $\lambda = 1$ in the experiments presented earlier in this chapter. Second, we have already conceded that when we apply direct reactive reinforcement learning methods to POMDPs the best policies which can be learnt are arbitrarily worse than the optimal policy based on full knowledge of the underlying MDP Singh et al. [1994]. Instead, we are interested in learning *satisficing* policies. The main effect of this theoretical observation is to suggest that if we are interested in ensuring that convergence can occur to optimal reactive policies, then we should consider the extreme case of $\lambda = 1$.

The same paper [Pendrith and McGarity, 1998] also demonstrates the existence of a POMDP for which the discounting of future rewards ($\gamma < 1$) results in the optimal reactive policy not being a learning equilibrium for most reinforcement learning algorithms. This result is sufficiently general to include CEQ(λ). As above, we do not consider this limitation sufficient to abandon CEQ(λ) but instead it suggests that we

[3] Non-Markovian Decision Process.

need to examine undiscounted rewards ($\gamma = 1$), which maybe necessary to ensure convergence to optimal reactive policies.

7.2.2 No guarantee of convergence to single policy for $\varepsilon > 0$

Perkins [2001] present an example POMDP on which ε-greedy SARSA(λ) cannot converge to a single policy for any λ and for all $0 < \varepsilon < \varepsilon_0$ for some $\varepsilon_0 > 0$. In appendix I, we extend their analysis and show that it holds for CEQ(λ).

As described in algorithm 4 (page 235) CEQ(λ) selects the next action to execute in line with the current policy $\pi = \arg_a \max Q(o,a)$ with probability $1-\varepsilon$ and varies from policy π with probability ε. Thus the probability of selecting the next action does not take into account the modified policy $\pi \leftarrow (o^e, a^e)$ which *is* used to carry out updates and truncate the eligibility traces. This possible defect in the design of the CEQ(λ) becomes obvious when considering the probability of selecting actions in the analysis undertaken in appendix I. We therefore also analysed the theoretical performance of an extended version of CEQ(λ), which we call CEQ(λ)$^+$ (the algorithm can be found in appendix J), where the modified policy $\pi \leftarrow (o^e, a^e)$ is taken into account when selecting actions. That is, it selects an action in line with $\pi \leftarrow (o^e, a^e)$ with probability $1-\varepsilon$, where $\pi = \arg_a \max Q(o,a)$ and a^e is the exploratory action being executed when observation o^e occurs. We show that for a slightly modified POMDP problem the same non-convergence to a single policy can be shown to hold for CEQ(λ)$^+$.

One interpretation of this result is as support for our policy of switching from ε-greedy to plain greedy execution of the policy $\arg_a \max Q(o,a)$ while maintaining learning. As demonstrated in chapter 5, this often results in convergence to satisficing policies, plus the absence of random sampling of observation-action values from other parts of the problem space allows the algorithm to converge on a single policy. An alternate observation is that as the problem presented in appendix I is configured to provide the maximum reward to an agent following a stochastic policy, SARSA(λ) and CEQ(λ) are actually behaving correctly in failing to converge to a single deterministic policy.

It is important to note that failure to converge to a single deterministic policy does not imply failure to converged to a stationary point in the observation-action value space. We show in section I.3 that undiscounted first-visit Monte Carlo using exploring starts cannot be guaranteed to converge to a single deterministic policy on the same POMDP as CEQ(λ). The instability of single deterministic policies using first-visit Monte Carlo is further supported by empirical evidence from section 6.1.3. That this occurs despite the existence of a proof from Pendrith and McGarity [1998] that such an algorithm *is* guaranteed to converge to the optimal stationary point in the observation-action value space indicates that reinforcement learning algorithms can, in all probability, switch between alternative policies and still maintain a set of observation-action values which float around the optimal valued stationary point.

The same observation, that failure to converge to a single deterministic policy does not imply failure to converge to a stationary point in the observation-action value space, means that we *cannot* combine this result for CEQ(λ), which holds for $\lambda = 1$ and $\gamma = 1$ with that made in section 7.2.1, which only held for $\lambda < 1$ or $\gamma < 1$. Thus neither of these sections say anything about the ability of CEQ(λ) to converge to the optimal stationary point in the observation-action value space when $\lambda = 1$ and $\gamma = 1$.

7.2.3 Summary of theoretical observations

We have established the following limitations on CEQ(λ) performance on POMDP problems:

(i) To always allow for the possibility of convergence to the optimal valued stationary point in the observation-action space, λ and γ need to be equal to 1.[4,5]

(ii) That convergence to a single static deterministic policy is not necessarily possible for ε-greedy action selection with $\varepsilon > 0$.

[4] We have not shown that CEQ(λ) will converge to this stationary point given $\lambda = \gamma = 1$, but Pendrith and McGarity [1998] demonstrate the existence of examples on which convergence cannot occur given $\lambda < 1$ or $\gamma < 1$.

[5] Convergence to such a stationary point does not imply convergence to a single static deterministic policy.

7.2.4 Future theoretical work

Having shown some theoretical limitations of CEQ(λ), it would be useful to show some positive results indicating under what conditions convergence — for example to satisficing policies — can be guaranteed.

In terms of a guarantee of convergence to reactive policies which represent optimal valued stationary points in the observation-action space, it would be useful to show equivalence between CEQ(λ) with $\lambda = \gamma = 1$ and undiscounted first-visit Monte Carlo algorithm. Equivalence would show that the guarantees presented by Pendrith and McGarity [1998] also hold for CEQ(λ) with $\lambda = \gamma = 1$.

If equivalence cannot be shown then it may be possible to arrive at separate convergence guarantees for CEQ(λ). Examination of proofs presented by Pendrith and McGarity [1998] or Perkins [2002b] may be useful guides as to how such proofs can be constructed.

It would be useful to establish theoretically whether or not acting greedily in line with the current observation-action values whilst updating those values, always results in convergence to single deterministic static policies.

As our focus is not on convergence to *optimal* reactive policies but to *satisficing* policies, it would be useful to establish some theories of convergence to such policies. We are not aware of any theoretical analysis of reinforcement learning which looks at the likelihood of convergence to a set of satisficing policies; all existing proofs consider only convergence to optimal policies. We suspect part of the problem is in defining formally what constitutes a satisficing policy. Although there has been some work on formally defining satisficing in the control theory and multi-agent literature [Goodrich et al., 1998; Stirling et al., 2002], we do not consider that the definitions adopted fit particularly well with reinforcement learning problems considered in this thesis. Our reasoning is set out in section 2.2.1.

In terms of showing formally that convergence to satisficing solutions can occur, it may be easier to initially consider a subset of reinforcement learning problems, such

as episodic minimum-cost-to-goal tasks, for example, Sutton's Grid World. For these tasks a relatively straight forward understanding of satisficing can be established, that is, policies that reach the goal from all start states given infinite time. However, how such a definition can be exploited to provide a proof of convergence is currently not clear to us.

7.3 Chapter Summary

In this chapter we developed a new reinforcement learning algorithm: Consistent Exploration Q(λ), or CEQ(λ). This new algorithm was developed by generalising the concept of consistent exploration, as presented in chapter 6.1.2, such that it could be applied to temporal difference algorithms.

We demonstrate empirically, on three POMDP problems, that CEQ(λ) learns more satisficing policies than SARSA(λ). This is an extremely interesting result, as SARSA(λ) is generally regarded as one of the best reinforcement learning algorithms for learning reactive policies on POMDP tasks. Even if SARSA(λ) had more time or action-learning steps, we do not believe that the percentage of satisficing policies would increase. In the chapter that follows we test whether this holds for further POMDP problems.

Finally we argue for our preference of CEQ(λ) over MCESP on the grounds that there is no requirement to impose artificial episode breaks, and that updating of Q values after every step, rather than waiting for the end of episodes, results in quicker learning.

Chapter 8

Comparison of Approaches

Chapter 4 introduced the concept of active perception and showed that in extending an agent's abilities, it allowed satisficing policies to be learnt where none were possible previously. That chapter also showed that active perception allowed policies which are closer, in terms of physical actions, to optimal policies to be learnt using direct reinforcement learning algorithms. Then in chapter 7 we developed an improved direct reinforcement algorithm by applying the idea of consistent exploration to a temporal difference learning method. This chapter takes both of these developments and uses them to provide evidence for our first thesis hypothesis by showing empirically "that the performance of active perception agents is comparable with other approaches in the field of reinforcement learning" (see section 1.2).

This chapter also provides evidence in support of our second thesis hypothesis "that there is no requirement to manage perceptual actions separately through the provision of specialised reinforcement learning algorithms" (section 1.2). To this end we included an algorithm that does manage perceptual actions separately, the Lion algorithm [Whitehead, 1992; Whitehead and Ballard, 1991], and observe its performance compared to approaches which do not distinguish between types of actions.

We achieve the above through the comparison of an active perception agent with other approaches that appear in the literature and which were reviewed in chapter 3. To

make this comparison we run a selection of agent designs and reinforcement learning algorithms over the same range of problems. The selected combinations of agents and algorithms represent all three of the approaches that we identified in section 2.3, that is, *fixed-SE*, *modifiable* and *basic approaches*. The specific hypotheses that we aim to test in this chapter are:

(i) active perception's performance is at least equivalent to alternative modifiable approaches (that is, agents such as those reviewed in section 3.3 that have internal memory registers that can be manipulated directly by the agent's core[1] reinforcement learning algorithm);

(ii) there is little performance gain by treating perceptual actions differently from physical actions, as, for example, the Lion algorithm does;

(iii) fixed-SE approaches, which use state-estimation in an attempt to make the task appear Markovian (for example, Lion algorithm or U-Tree [McCallum, 1995b, 1996]) perform little better than the active perception approach advocated in this thesis; and

(iv) the active perception approach can be successfully be extended to more complex, stochastic environments that more closely resemble the real world.

A further aim of these experiments is to show that CEQ(λ) generally outperforms SARSA(λ) on POMDP problems. This is an important comparison, as SARSA(λ) is widely reported in the literature as a "high standard to compare to" [Perkins, 2002b], and "that SARSA(λ) may be hard to beat in problems where there exists a good policy that maps observation space to actions" [Loch and Singh, 1998]. To this end we aim to test a further hypothesis:

(v) of the two direct reinforcement learning algorithms tried, CEQ(λ) outperforms SARSA(λ) on POMDP tasks. (For this hypothesis, these experiments provide further evidence to that provided in chapter 7.)

[1] See section 2.3 for our definition of core-learning-algorithm.

(vi) CEQ(λ) performs as well on stochastic tasks as it does on deterministic grid world problems. That is that the introduction of stochasticity does not degrade it performance compared to that of SARSA(λ).

In the first section of this chapter we test four learning algorithms, CEQ(λ), SARSA(λ), Lion algorithm and U-Tree, in combination with a variety of agent designs on a set of randomly generated grid worlds. A wide range of parameter settings are tried and the data obtained from the broad sweep of experiments is divided up in different ways to look at each of the above hypotheses.

In the second section of this chapter we conduct a much more limited set of experiments on a task which involves a simulated world containing a robot navigating a set of corridors. This task uses a simple physics engine to provide a more realistic environment which has the additional feature of being stochastic in nature.

8.1 Randomly Generated Grid Worlds

This section compares the performance of CEQ(λ), SARSA(λ), Lion and U-Tree algorithms combined with various types of agent on ten randomly generated grid world problems.

8.1.1 Experimental setup

The sections below describe the ten random test worlds, the agent designs, the learning algorithms, and the parameter values used for this set of experiments.

8.1.1.1 Test environments used

Ten grid worlds were generated through the random placement of positive goals, penalty goals and obstacles on a 7×9 grid which is bounded by walls on four sides. The algorithm used to generate the random worlds can be found in appendix L. The ten grid

Agent \ Algorithm	CEQ(λ)	SARSA(λ)	Lion Algorithm	U-Tree
Eight Adjacent Squares (Fixed Perception)	✓	✓		✓
Active Perception	✓	✓	✓	
N-bit Memory	✓	✓		

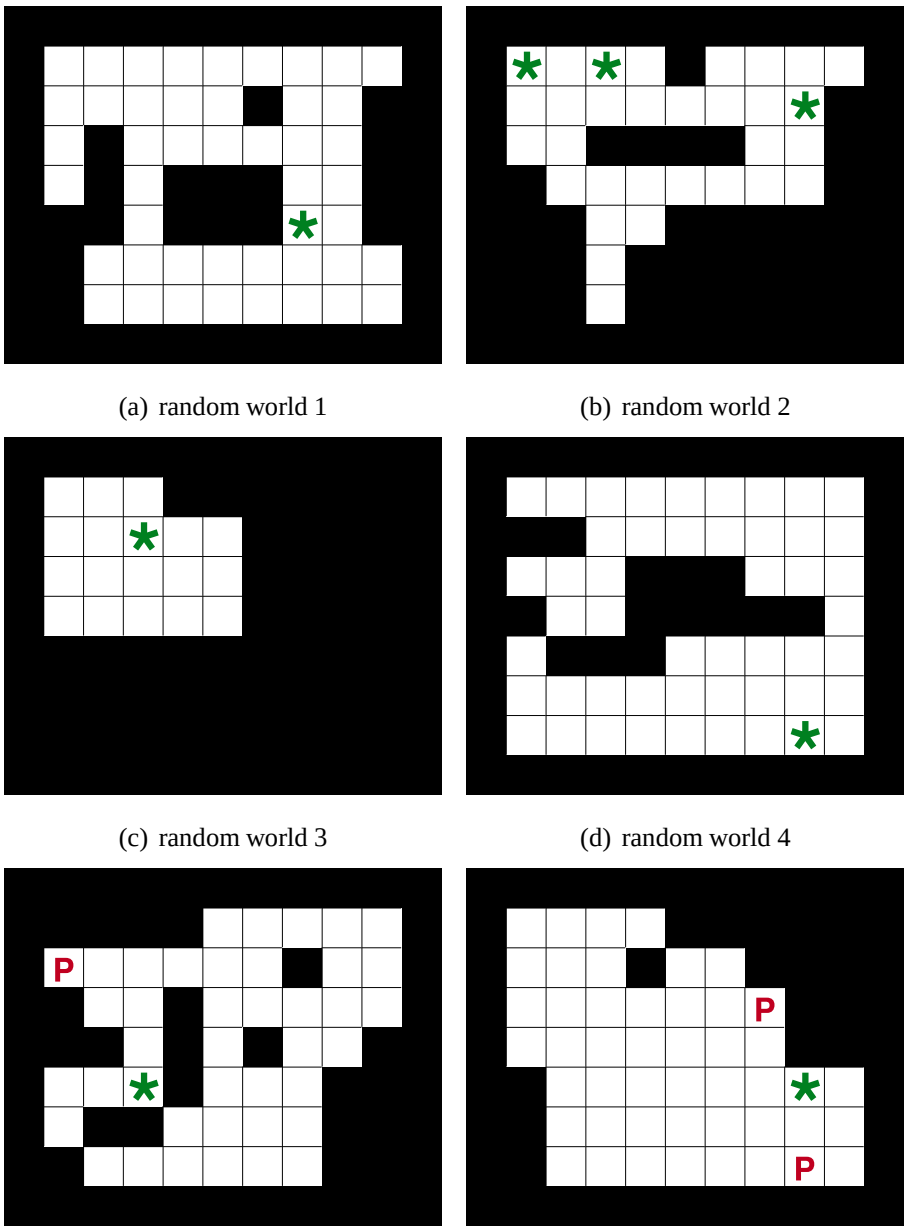
Table 8.1: Combination of reinforcement learning algorithms and agents tested

worlds used in these experiments are shown in figures 8.1 and 8.2. The same ten grid world were used for testing all agent-algorithm combinations.

An agent in one of these worlds can choose between four physical actions; move North, South, East and West. State transitions are deterministic and each action moves it one square in the appropriate direction. If an agent tries to move towards an obstacle its location remains unaltered, although it receives the same penalty as if the action had succeeded. On executing a physical action the agent receives a reward of $+10$ if that action results in it immediately arriving at a positive goal location (indicated by an asterisk), -100 if it immediately arrives at a penalty goal location (indicated by the letter P), otherwise it receives a penalty of -1. When the agent reaches a goal, it is relocated to a start location selected at random using a uniform distribution across all locations that do not contain obstacles.

8.1.1.2 Agents and learning algorithms

The combinations of agents and learning algorithms that are tested in this section are summarised in table 8.1. Learning algorithms are coupled with agents that are complimentary to their abilities. The Lion algorithm is only run with an agent that has active perception, while U-Tree, which carries out state-estimation using memory, is only run with an agent with fixed perception (the Eight Adjacent Squares agent).

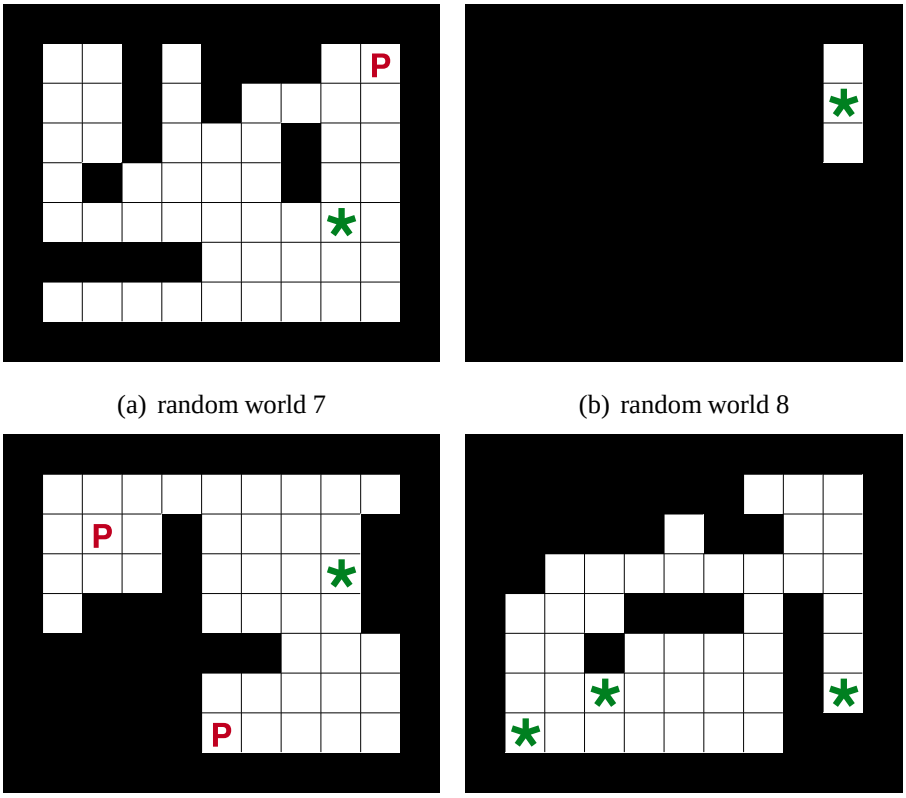


(a) random world 1 (b) random world 2

(c) random world 3 (d) random world 4

(e) random world 5 (f) random world 6

Figure 8.1: Random grid worlds 1 through 6



(a) random world 7

(b) random world 8

(c) random world 9

(d) random world 10

Figure 8.2: Random grid worlds 7 through 10

Agents

The Eight Adjacent Squares Agent is as described in section 4.1.1. It obtains an eight bit observation that represents its state in the environment. This eight bit string is made up of observations of whether or not the eight squares adjacent to its location are occupied by an obstacle.

The Active Perception Agent, which is described in more detail in section 5.2.1.3, receives the same basic observation as the Eight Adjacent Squares agent but has eight additional *perceptual actions* that it can select. On selecting one of these perceptual actions, its observation changes to an eleven bit string by adding information from three additional squares in the direction of its choosing. It receives a penalty of -1 for selecting a perceptual action.

The N-bit Memory Agent generates an observation consisting of 8+N bits, where the core 8 bits are the observation that would be obtained by the Eight Adjacent Squares Agent. The remaining N bits are states of an internal memory register which the agent can set or reset through the selection of certain actions. In practice, N is a fixed parameter of the agent which is set manually. In these experiments we only consider 1-bit memory agents.

Our chosen N-bit memory agent architecture is similar to those found in the literature, for example, Littman [1994]; Peshkin et al. [1999]; Lanzi [2000] which are reviewed in section 3.3.2. Each N-bit memory agent has a set of physical actions that allow it to move through the world. In addition it has a set of "memory + physical actions" that couple movement with an operation on the internal memory register. In our design the operation allowed is that of "flipping" the state of the memory bit, that is, a bit which has value 0 will be flipped to have value 1, and a bit which has value 1 will be flipped to have value 0. Using this architecture an N-bit Memory Agent has $M + M \times N$ actions available to it, where M is the number of physical actions available to the agent. In the test worlds used in this section $M = 4$, that is, movement is allowed in the four principle compass directions. Thus, for a 1-bit memory agent (where $M \times N = 4$) the additional actions are:

- Move North and flip memory bit,
- Move South and flip memory bit,
- Move East and flip memory bit,
- Move West and flip memory bit,

These, plus the four plain physical actions, make a total of eight actions available to the learning algorithm.

As discussed in section 2.3, N-bit memory agents of this form apply direct reinforcement learning to non-Markovian tasks. The manipulation of memory bits allowing it to change the observations it receives in a similar manner to the active perception approach. We therefore envisaged in section 2.3.1 that the performance of such N-bit memory agents may be similar to that achieved with an active perception agent.

Algorithms

SARSA(λ) is implemented as specified in Sutton and Barto [1998, p.181] and given in appendix D. **CEQ(λ)** is implemented as specified in section 7.1.2. Both these algorithms are *direct* reinforcement learn algorithms that learn reactive policies. The generality of learning reactive policies allows both these algorithms to be run with all three types of agent.

The Lion Algorithm [Whitehead and Ballard, 1991; Whitehead, 1992] is a modified form of Q-learning which uses a separate perceptual cycle to identify the current state of the world before selecting the next physical action to execute. To identify the current state of the world the perceptual cycle tries a series of perceptual actions, with the aim of finding a mapping that results in non-aliased observations for each significant world state. We review the Lion algorithm in section 3.1.1, and further details as well as the complete algorithm can be found in appendix H . We make one modification to the algorithm as originally presented by Whitehead and Ballard [1991]. In its original form, when Q-values are suppressed they are reset to zero. This approach assumes that

all rewards are positive, that is, there are no costs, or penalty goals. In the experiments presented here this is not true, and reseting a Q-value to zero might not result in its suppression, but instead promote the likelihood of it being selected. The aim in suppressing Q-values is to make their selection unlikely, but not to make it impossible for them to recover should their suppression prove unwarranted. To this end, our modification is to keep a record of the lowest Q-value generated when performing updates, and then use this value when called on to suppress other Q-values. We initiate this "suppression value" at zero, so for the case when all rewards are positive our modified Lion algorithm performs identically to the original.

U-Tree, which is reviewed in section 3.1.8.4, is specified in McCallum [1995b, 1996]. We found that parts of the specification given in these two sources was confusing or incomplete. As a result we provide supplementary notes in appendix G which also provides details of our implementation.

The U-Tree algorithm learns an internal model based on the observations that it receives, in an attempt to uncover the underlying Markov structure of the world. Given this, we consider that it would only confuse the algorithm by coupling it with either the Active Perception or N-bit Memory agents. Both of these agents allow actions which manipulate the observations seen by the learning algorithm. We therefore only run U-Tree with the Eight Adjacent Squares Agent whose observations are consistent, though not unique, for each of the underlying states of the test worlds.

8.1.1.3 Parameter values use

A broad sweep of parameter values were tried with all four algorithms. In part the aim was to give each a good chance of performing well but in addition it allows the robustness to variations in the parameter values to be judged. The parameter values used are specified below.

Common parameters

All four algorithms require a discount rate for future rewards (γ) and an action selection

method to be specified. The range of values used for γ were 0.8, 0.9, 0.99 and 1.0. The action selection method chosen, based on results in chapter 5, was ε-greedy becoming ultimately greedy. The schedule used for ε was to initiate it at 0.2 and decay it linearly to zero by the $500,000^{th}$ action-learning step.

Common parameters excluding U-Tree

With the exception of U-Tree, three of the algorithms require a learning rate (α), to be specified. The range of values used were 0.01, 0.02, 0.05, 0.1 and 0.3.

The same three algorithms, CEQ(λ), SARSA(λ) and Lion algorithm maintain Q values using a simple tabular structure[2]. In these experiments the entries in these Q-value tables are initiated to random values drawn from a Gaussian distribution, with mean zero and standard deviation of five. Initial policies π were then based on these Q values using $\pi \leftarrow \arg\max_a Q(o,a)$.

Again, with the exception of U-Tree, each trial was run for a total of one million action-learning steps. For each combination of world, agent and parameter values, twenty independent trials were run allowing us to record the number of times each algorithm converged to a satisficing policy, and the mean total steps required by each satisficing policy.

CEQ(λ) and SARSA(λ)

Both CEQ(λ) and SARSA(λ) use eligibility traces. We use *replacing* traces as described in Sutton and Barto [1998, p.186], with a range of values for λ of 0.8, 0.9 0.99 and 1.0.

Lion algorithm

Parameter values specific to Lion algorithm are.

[2] In our code we use hash tables to store the Q-values and as such we do not need to know the full observation-action space at the start. Entries are added to the hash table, which is index by the observation, as they are encountered. The initial Q-values for these new entries are drawn from the specified random distribution.

- The number of perceptual actions executed during each perceptual cycle. This is set to four perceptual actions per cycle.

- The learning rate (α') which is used to update observations associated with the world state, but which were not selected as the "lion" observation. Two values both related to α were used; $\frac{\alpha}{2}$ and $\frac{\alpha}{10}$.

U-Tree

U-Tree builds a tree, the leaves of which are the internal states of the reinforcement learning algorithm. For simplicity of coding the *Q* values for U-Tree leaves were initiated to zero rather than to random values. Initiating *Q*-values to zero may give U-Tree a small advantage as can be observed from the results in chapter 6, that convergence of policies to satisficing solutions is generally more reliable when *Q*-values are initiated with zeros than when they are initiated to random values.

Compared to the other three algorithms, U-Tree requires significantly more computational power and time. The most costly part of the algorithm is generating all possible expansions of the fringe and testing each fringe node. The run time of U-Tree increases non linearly with the number of action-learning steps the agent has taken. This is demonstrated by two trial runs shown in figure 8.3. This slow down is, in part, due to the growing size of the instance chain, but as demonstrated in figure 8.3, the processing time of the algorithm increases even more dramatically when U-Tree is adding leaves to the tree.

Two U-Tree parameters control the size of the fringe expansion, the maximum fringe depth and maximum history index. The first parameter controls how deep the expansion is, which reflects the maximum number of conjunctions that the fringe expansion can test. The second parameter controls how far back into the history of instances, that is, how far back up the instance chain, that U-Tree looks when creating fringe expansion. Increasing values of both parameters increases the expressiveness of the fringe and the depth into history that it considers when trying to distinguish utile states.

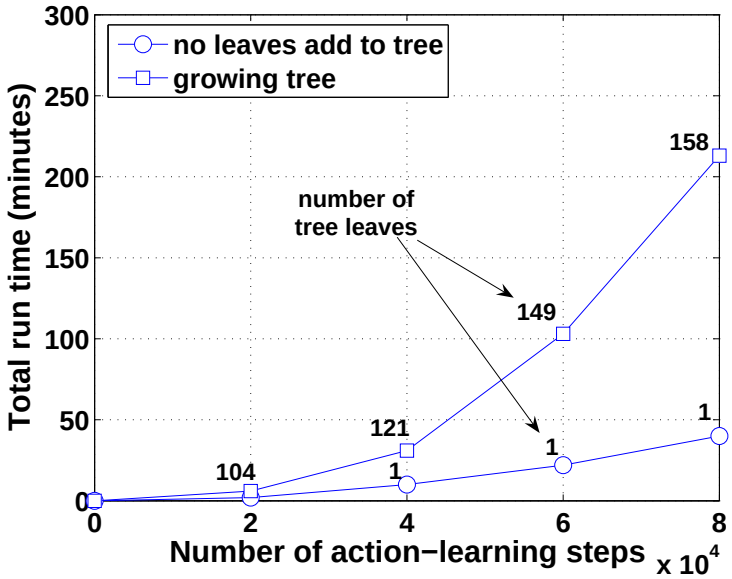


Figure 8.3: Plot of run time against number of action-learning steps for U-Tree. The two plots are for two individual runs of U-Tree algorithm with the following parameters: maximum fringe depth 2, maximum history depth 3, Kolmogorov-Smirnov probability level (p) of 0.1. One run discovered the goal and grew a tree to try and capture the dynamics of the grid world problem. The other run failed to find the goal and thus never found any utile distinctions that necessitated growth of the tree. Both runs were executed on a machine with a single 2GHz Pentium 4 CPU and 512Mb of memory

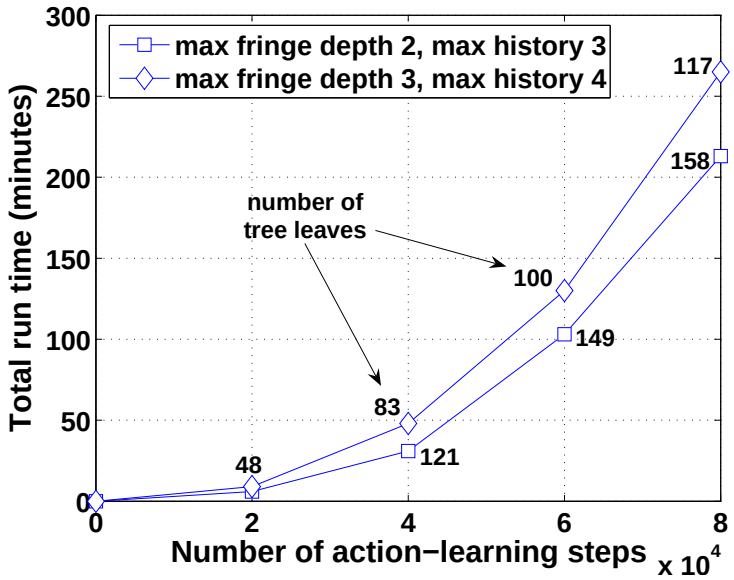


Figure 8.4: Increase in run time for increased fringe depth and history index of U-Tree algorithm. Both plots run with Kolmogorov-Smirnov probability level (p) of 0.1, on a machine with a single 2GHz Pentium 4 CPU and 512Mb of memory

The drawback is that increasing these parameters results in a significant increase in U-Tree's running time, as demonstrated in figure 8.4. For this reason the parameter values of *maximum fringe depth* and *maximum history index* used in these sets of experiments were constrained to three and four respectively.

We also constrained the branching factor of the fringe by making U-Tree treat the eight bit binary observations of the Eight Adjacent Squares as a single *feature* that was indivisible. If we allow U-Tree to treat each bit as a feature, then this increases the branching factor of the fringe expansion by a factor of eight, resulting in a further slowing of the algorithm. In addition, with the fringe depth constrained to just three layers, U-Tree is incapable of considering conjunctions of more than three bits at a time, and there is a possibility that this may not be enough to distinguish all the states. Treating the 8-bit observation as one feature allows U-Tree to discover conjunctions that it would otherwise not be able to perceive. Thus, by adopting this approach, we believe that we are aiding rather than handicapping the algorithm.

To constrain the run time further we limited each trial to 100,000 action-learning steps.

This was considered reasonable because a majority of the preliminary runs made with U-Tree ran out of memory at around ninety five or one hundred thousand action-learning steps.

The set of parameter values set out above kept the average running time to somewhere around ten hours per trial. As this still represents a reasonably long time, we also reduced the number of samples which were run for each combination of parameters from 20 to 10.

Other U-Tree parameters are listed below.

- The Kolmogorov-Smirnov test, used to indicate whether two sets of instances are the same set, returns a probability measure. U-Tree requires that some level of probability be specified below which it is deemed that the two sets of instances are independent and the fringe leaf being tested should be added to the tree. We tried a range of values for this parameter p. The values tried are 0.1, 0.01 and 0.001.
- We also need to specify the number of action-learning step which occur between each attempt to expand the tree. This was set to a 1000 action-learning steps.
- The minimum number of instances required in a fringe leaf before it was deemed reasonable to carry out a Kolmogorov-Smirnov test was set to 20.
- The minimum fringe depth expansion that was generated was set to 1 in order to match the code presented in McCallum [1995b, 1996].
- Sets of instances are compared using the Q-values for the *current policy action* of the parent leaf note, this again matches the code presented in McCallum [1995b, 1996]. The alternative would be to compare the Q-values of the instances across all possible actions.

Note that the latter two parameters are specific to our implementation of U-Tree.

In the next section we discuss results of the above trials.

8.1.2 Results

Tables 8.2 and 8.3 summarise the results for CEQ(λ), SARSA(λ), Lion algorithm and U-Tree. Table 8.2 presents the mean numbers of satisficing policies learnt, averaged across the ten random grid worlds. Results are averaged across all of the parameter combinations tried. Values are presented as percentages of the number of trials run[3]. Table 8.3, similarly presents mean number of satisficing policies learnt but these figures are for the best performing parameters values for each agent-algorithm combination. A detailed breakdown of results for each parameter value combination can be found in appendix O.

8.1.2.1 Agent-algorithm combinations

We consider the relative performance of the agent-algorithm combinations tried. Table 8.2 indicates that, in terms of the mean number of satisficing policies learnt, neither combination of Lion Algorithm and Active Perception Agent or U-Tree and Fixed Eight Adjacent Squares Agent perform particularly well when compared with the results for any combination involving SARSA(λ) or CEQ(λ). This more-or-less holds even if we select the best performing parameters values for each agent-algorithm combination, see table 8.3. Though the best results for the Lion-algorithm do approach those achieved by the SARSA(λ) Fixed Eight Adjacent Squares Agent combination. (Selecting the best performing combination of parameter values should control for any bias that may have been introduced through our choice of parameter value ranges.)

Examining the underlying data[4] we find that the Lion algorithm's performance showed a large variation across the parameter space that was tested. The algorithm performed well for values of γ close or equal to one. Its best performance occurred for $\gamma = 0.99$ coupled with small values of α and α', 0.01 and 0.001 respectively. By comparison U-Tree's performance is fairly uniform across the parameter space tested, with the

[3] The results for U-Tree are out of 10 trials, while those for SARSA(λ), CEQ(λ) and the Lion algorithm are out of 20.

[4] A detailed breakdown of results for each parameter combination tried can be found in appendix O.

Agent \ Algorithm	CEQ(λ)	SARSA(λ)	Lion Algorithm	U-Tree
Eight Adjacent Squares (Fixed Perception)	$90\% \pm 28\%$	$80\% \pm 37\%$		$48\% \pm 45\%$
Active Perception	$99\% \pm 6\%$	$88\% \pm 29\%$	$39\% \pm 46\%$	
1-bit Memory	$99\% \pm 2\%$	$90\% \pm 27\%$		

Table 8.2: Mean number of satisficing policies for each agent-algorithm combination. Figures are averaged across all parameters and test worlds. Values presented as percentages of the number of trials run $\pm$ percentage standard deviation

Agent \ Algorithm	CEQ(λ)	SARSA(λ)	Lion Algorithm	U-Tree
Eight Adjacent Squares (Fixed Perception)	100%	$92\% \pm 20\%$		$53\% \pm 50\%$
Active Perception	100%	100%	$90\% \pm 32\%$	
1-bit Memory	100%	100%		

Table 8.3: Best satisficing figures for each agent-algorithm combination. For each agent-algorithm combination the results are broken down by the parameter value combinations tried. From these results the best values achieved, in terms of the number of satisficing policies when averaged across all ten test worlds, are then selected. Values presented as a percentage of the number of trials runs $\pm$ percentage standard deviation

Agent	Approach	Number of satisficing policies
Eight Adj. Squares	basic	$85\% \pm 33\%$
Active Perception	modifiable	$93\% \pm 22\%$
1-bit Memory	modifiable	$95\% \pm 20\%$

Table 8.4: Mean number of satisficing policies for basic and modifiable approaches. Values reported are the number of satisficing policies learnt as a percentage of the number of trials run. Average calculated across all ten test worlds, both direct learning algorithms (CEQ(λ) and SARSA(λ)) and the whole parameter space. Values shown are mean $\pm$ standard deviation

number of satisficing policies learn being consistently around the level of 50% of the trials. This results is due to U-Tree performing consistently well on half of the random grid worlds and consistently badly on the others. It performs well on worlds 2, 3, 8, 9, and 10, and badly on worlds 1, 4, 5, 6 and 7. As yet we have no theory about what distinguishes these two sets of worlds as far a U-Tree is concerned.

It is relatively clear from tables 8.2 and 8.3 that the best performing algorithm is CEQ(λ) closely followed by SARSA(λ) and the best two performing agents are the Active Perception Agent and 1-bit Memory Agent. Comparing overall averages (averaged across both CEQ(λ) and SARSA(λ)) for the three agent designs, see table 8.4, indicates that performance of the two modifiable approaches (active perception and 1-bit memory) are roughly on a par. With both learning more satisficing policies, on average, than the basic approach (Eight Adjacent Squares agent).

8.1.2.2 Comparison of CEQ(λ) and SARSA(λ)

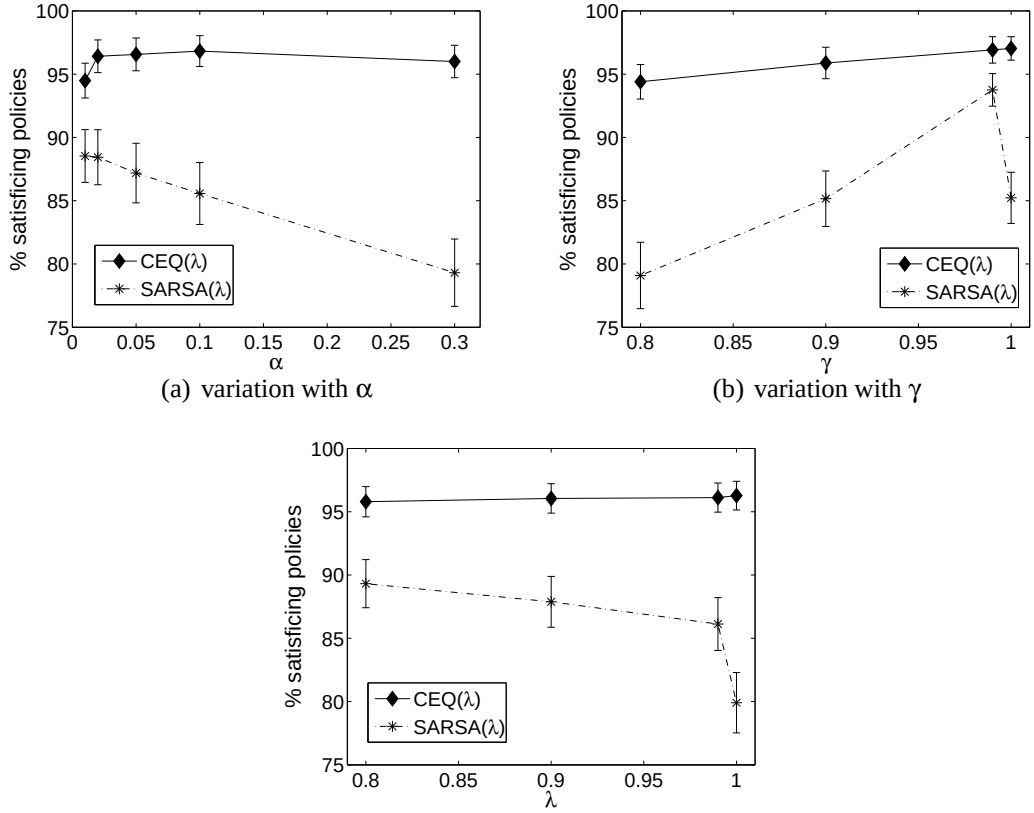
Our attention now switches to a comparison of the two direct reinforcement learning algorithms, CEQ(λ) and SARSA(λ). Averaging across across all combinations of worlds, agents and parameter values the mean number of satisficing solutions learnt by CEQ(λ) is 19.21 out of a maximum of 20 (or 96%), with a standard deviation of 3.44 (17%). This is noticeably better than SARSA(λ) which scores a mean of 17.16 (86%)

with standard deviation 6.29(31%). Student's t-test for unequal variance indicates this difference is statistically significant with the probability of both sets of results being drawn from the same distribution being equal to 2×10^{-43}.

As well as have better overall averages, CEQ(λ) averages are better than SARSA(λ) for each agent design tried, see tables 8.2 and 8.3. In fact CEQ(λ) out outperforms SARSA(λ) across most of the parameter space that we tested. Variation of both algorithms' average performance across the range of values tried for each parameter is plotted in figure 8.5. As can be observed from this figure the performance of CEQ(λ) is reasonably uniform across the parameter space tested. There is sudden fall off in its performance for the lowest value of α used and examining the raw data for CEQ(λ) reveals that at this low level of α the policies converge much slower and several have yet to converge after one million action learning steps. It appears that these policies would have eventually converged to satisficing solutions had the trials been extended and thus CEQ(λ)'s average performance would improve further.

SARSA(λ)'s performance is much more dependent on the parameter values although most of the variation is explicable. For example in figure 8.5(b) the rise in the number of satisficing policies as γ increases can be attributed to the algorithm valuing future rewards more highly which in turn emphases the value of reaching the goal state even if it is via longer routes. Of more interest is that SARSA(λ)'s performance falls off abruptly when either λ or γ are equal to 1.0. Although not directly demonstrated by these plots, we can report that the effect for λ and γ combines to produce a very large fall off in performance when both are equal to 1.0. This is in contrast to CEQ(λ) which does exceptionally well when either λ, γ or both are equal to 1.0. We comment on this further in section 8.1.3. A more detailed breakdown of of the comparative number of satisficing policies learnt for each combination of parameter values and agents can be found in appendix O.

As the satisficing policies learnt are not necessarily optimal we also examine their quality to see if either algorithm is, on average, significantly worse than the other. The evaluation of a policy consists of placing the agent at every possible location in the grid world where there is no obstacle, and counting how many steps it takes to reach a



(a) variation with α (b) variation with γ

(c) variation with λ

Figure 8.5: Mean percentage of policies which are satisficing plotted against variation in parameter values. Averages taken across all three agents, ten test worlds and the two free parameters out of α, γ and λ for each plot, for example, for figure 8.5(a) averages for each value of α are calculated over all values of γ and λ. Error bars show 95% confidence intervals

positive goal state. The total steps taken by the agent is then summed up over all start states. Note that in calculating average total path lengths we only include policies that are satisficing. Table 8.5 reports the average total path length for satisficing policies for each world (averaged over the three agent designs and all parameter combinations). Also shown is the percentage difference in mean path length for CEQ(λ) compared to SARSA(λ) and the statistical significance of comparing the CEQ(λ) and SARSA(λ) samples. As can be seen the average satisficing policy path length learnt by the two algorithms is very close. For three of the worlds, 1, 4 and 10, the results are so similar that we can not even determine with certainty that the samples for CEQ(λ) and SARSA(λ) are drawn from distinct distributions. For the remaining worlds the maximum increase when using CEQ(λ) compared SARSA(λ) is 2.3% and largest reduction is -6.6%. Overall this suggests that the improvement in the *number* of satisficing policies learnt by CEQ(λ) has negligible detrimental impact on the *quality* of the policies learnt when compared to SARSA(λ).

8.1.3 Discussion

In terms of the reliability with which the various approaches converge to satisficing policies, the results, especially those in tables 8.2 and 8.3, clearly demonstrate that the direct reinforcement learning algorithms out perform the state-estimation approaches tried. The results further demonstrate that the extension of an agent's abilities, in line with the *modifiable approach*, improves the likelihood of convergence to satisficing policies. This result we suggest is due to an increase in number of possible satisficing solutions that the agent's additional abilities bring into being. A closely related effect was reported by Lanzi [2000] as he increased N for a set of N-bit memory agents.

In terms of computational power and time, the direct reinforcement learning algorithms are much less demanding than say U-Tree. The simple tabular (or hash table approach) as used by the direct reinforcement learning algorithms, requires typically 100 kilobytes of memory to store all observation-action pairs for the test worlds used. In comparison, U-Tree's memory requirements are potentially infinite, with storage

Random World	Mean total path length SARSA(λ)	Mean total path length CEQ(λ)	percentage difference	statistical significance
1	257.0±50.9	255.2±56.7	-0.7%	0.11
2	137.1±44.7	132.9±38.8	-3.1%	9×10^{-7}
3	52.2± 9.0	51.1± 6.4	-2.1%	3×10^{-12}
4	395.2±65.4	396.2±61.2	+0.3%	0.44
5	350.6±73.1	327.3±51.9	-6.6%	3×10^{-66}
6	393.7±88.7	385.9±63.1	-2.0%	7×10^{-5}
7	277.0±49.7	283.4±53.2	+2.3%	2×10^{-9}
8	2.1± 0.3	2± 0	-4.0%	4×10^{-64}
9	258.0±53.2	261.9±56.9	+1.5%	0.02
10	170.2±31.6	170.5±35.9	+0.1%	0.73

Table 8.5: Mean total path length of satisficing solutions learnt by SARSA(λ) and CEQ(λ). The percentage difference in mean path length for CEQ(λ) compared to SARSA(λ) and statistical significance when comparing results for CEQ(λ) and SARSA(λ) is also shown. Statistical significance calculated using Student's t-test for unequal variance (results for t-test assuming equal variance are virtually identical) and the figures shown indicate the probability that the samples for CEQ(λ) and SARSA(λ) are drawn from the same distribution

required for both a tree and an instance chain which records every action, observation and reward.

Run time for the two approaches also differs massively. The direct reinforcement learning algorithms typically complete one million action-learning steps in less than 10 minutes on a machine with a single 2GHz CPU and 512M of memory. U-Tree, as discussed in section 8.1.1.3, takes ten hours to execute one hundred thousand action-learning steps on an identical machine.

The computational power required for the Lion algorithm is similar to that for the direct reinforcement learning algorithms, with run times of around 10-15 minutes on a machine with a single 2GHz CPU and 512M of memory.

For worlds where U-Tree performs well, with the exception of random world 8, we observed that it tends to continuously add leaves to the tree as more instances are added to the instance chain. Given that the worlds are deterministic, this is unlikely to be over-fitting of the data due to noise. Having examined a couple of the trees we see that this growth tends to correspond to areas where multiple aliased states are grouped together. It appears that a lot of effort is put into uniquely identify such aliased states, and we suspect that cyclic paths that result in repeated visits to aliased states may produce sufficiently confusing utility values, that U-Tree believes its internal model needs to be extended. In the runs reported here the number of leaves in a tree often exceed the number of underlying states in the world by a factor of ten, before U-Tree either ran out of memory or completed 100,000 action-learning steps. The one world where this continuous expansion was not witnessed was random world 8 which is not only the simplest world, but when combined with a Fixed Eight Squares agent, as used with U-Tree, turns out to contain no aliased states. Tree growth in this world appears (on the basis of the four successful policies that we examined) to be limited to three leaves, each leaf representing one of the observable states of the world.

We believe that the problems we experienced with U-Tree running out of memory stem from the storage of every action, observation and reward in the instance chain. The length of the instance chain needs to be curtailed in order to allow longer running

of this algorithm. However, simply truncating the instance chain raises an interesting question about what should happen to sections of tree's structure which are no longer supported by the remaining data.

8.1.3.1 CEQ(λ) performance

In terms of comparing CEQ(λ) with SARSA(λ) the results match those observed in chapter 7. This is very encouraging as a larger body of test worlds are used in this section. To reiterate the observations from chapter 7, the results overall show that the probability of CEQ(λ) learning satisficing policies is better than SARSA(λ), and with the exception of slow convergence for low α values CEQ(λ)'s performance is robust over a wide range of parameter values. Robustness to variation in parameter values is useful in practical applications, like say training a real world robot, where having to repeat training with various parameter settings in order to find those that converge, is time consuming and sometimes impractical.

In contrast to SARSA(λ), CEQ(λ)'s performance consistently improves as both λ and γ are increased towards one. Although having both γ and λ equal to 1.0 is unusual when applying reinforcement learning to MDPs, theoretical results [Pendrith and McGarity, 1998] demonstrate that when learning policies for POMDPs, using undiscounted and Monte Carlo style[5] returns will ensure that the optimal reactive policy (where it exists) lies at a stationary point in terms of *Q*-value updates. Thus it is of interest to observe the effect of setting both close to, or at 1.

As we have noted previously random world 8 as shown in figure 8.2(b) is a MDP when combined with the Eight Adjacent Squares agent. CEQ(λ) performed perfectly on this agent world combination, which is not that surprising given the world's simplicity. To further prove the performance of CEQ(λ) on MDPs we have also run a few trials on Sutton's Grid World using a agent which is aware of its precise location in the grid world (Absolute Position agent) – this is the setup as used in section 4.1.1. The

[5] With λ equal 1.0 eligibility traces approximate to a Monte Carlo style approach [Sutton and Barto, 1998, p.188].

results of the few runs we have undertaken show that CEQ(λ) reliably learns satisficing, though not necessarily optimal policies on MDPs.

8.1.3.2 Hypotheses

Returning to the hypothesis that we set out for this chapter, we observe that:

(i) Active perception's performance is very close to that achieved by 1-bit memory agents. Averaged over the parameter values and two direct learning reinforcement learning algorithms, the 1-bit memory agent performs marginally better than the active perception agent, but the difference is very small, see table 8.4.

(ii) As an example of a consistent representation approach, the Lion algorithm does not indicate that treating perceptual actions different to physical actions leads to any significant gains. In fact, given the high level of performance achieved by the two modifiable approaches (Active Perception with CEQ(λ), etc.) there appears to be little room left for improvement on these type of problems.

(iii) The fixed-SE approaches of Lion algorithm and U-Tree perform poorly compared to modifiable approaches, the latter including both the active perception and 1-bit memory agents.

(iv) Points (i) to (iii) above supports out the expectation that we raised in section 2.3.1 that differing underlying assumptions should lead to a marked difference in performance between fixed-SE and modifiable approaches, but similar levels of performance for agent extensions that fall within the same approach.

(v) These experiments provide further evidence that CEQ(λ) outperforms SARSA(λ) on POMDP problems when looking at the number of satisficing policies learnt, see section 8.1.2.2. We also show that there appears to be little difference in the quality of the policies as measured using the mean total actions taken to reach the given goals.

8.2 Simulated Robot Environment

We introduce a new environment to test the applicability of the active perception approach to a task which is closer to one that could be faced by a robot in the real world. The world that we use consists of a simulated set of corridors, which are loosely based on the corridors found adjacent to our laboratory in the James Clerk Maxwell Building (JCMB) of the University of Edinburgh. The task consist of a simulated RWI, model B21, robot learning a policy to successful navigate and enter a laboratory from random starting locations. Figure 8.6 shows the simulated corridors and robot.

The reason for considering this more sophisticated test environment is that, compared to the deterministic grid world tasks, three new factors are added by the simulator, namely:

- all actions and sensing are *stochastic*, the simulator applies simple noise models to sensor readings and to the movement of actuators and the amount of stochasticity is close to that found in the real world,
- the observations available to the simulated robot are potentially far more complex,
- the position of the agent (the simulated B21 robot) is effectively continuous[6]. Note, however, that the action and observation spaces are discrete.

8.2.1 Experimental setup

8.2.1.1 The simulator

The simulator used is the Webots mobile robot simulator [Cyberbotics, 2005], version 4.0.27, which we ran using the Linux (Fedora core 3) operating system. The physical

[6]Only "effectively continuous" as it is being simulated by a digital machine and therefore some discretization will occur but at such a fine resolution that its effects are probably insignificant.

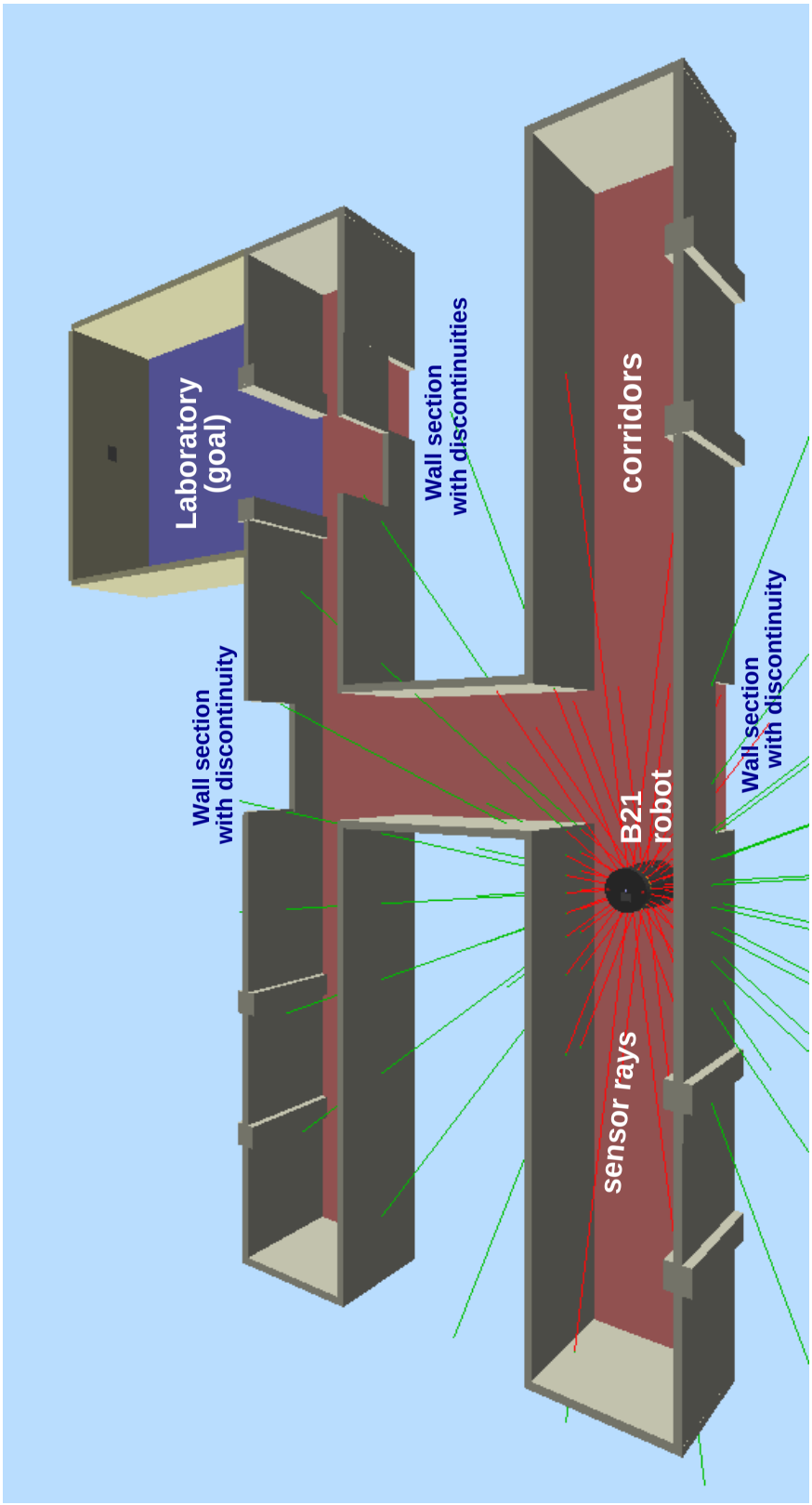


Figure 8.6: Simulated corridor environment and B21 robot

structure of the simulated environment is specified using the Virtual Reality Modelling Language (VRML). The form of VRML interpreted by the Webots simulator has been extended with the addition of objects specific to the simulation of robots, for example, various forms of sensors and wheeled bases. The properties of these objects, for example performance and noise parameters, are specified as part of the VRML description.

8.2.1.2 The world

The world consists of five sections of corridor which are 6 metres long by 1.6 metres wide. These five sections are assembled in a "H" shape which is 13.6 metres long and 6 metres wide. Eight support pillars protrude from the corridors' walls, four along each length of corridor that makes up the sides of the H shape. Along one leg of the H, between two supporting pillars, there is an opening into a laboratory space. With the exception of this opening and three discontinues, the corridor walls and supporting pillars are laid out such that the environment, when viewed from above, has rotational symmetry, and reflective symmetry along two orthogonal axes that pass through the centre of the H. Thus the environment creates naturally aliased states, for example when standing in one of the three corridor legs that do not contain the laboratory, it is difficult to know exactly where you are.

The real environment, that this model is loosely based upon, contains a variety of features that would be detectable by the robots sensors. The robot's sonar sensors can detect discontinuities caused by the presence of notice boards, posters or even changes in building materials [Marsland et al., 2000; Crook et al., 2002]. The Webots simulator only provides a simple model of sonar sensors, which give a distance reading in a straight line to the nearest object. There is no provision to model the different reflective properties of materials. To create an environment that contained some features we created discontinuities in the corridor walls. These discontinuities consist of sections of wall cut out such that the opening is visible to either the robot's ring of sonar sensors, or its ring of infra-red sensors. We create three different discontinuities at the three junctions that occur in this world; the junction at the entrance to the laboratory, and the two T-junctions where the crossbar of the H meets the two parallel corridors.

8.2.1.3 The robot

The simulated robot is based on the model B21 robot which was constructed by RWI robotics (now part of iRobots). The B21 robot is cylindrical in shape with a short base section above which the main trunk rotates. A picture of the simulated robot and the robot which was modelled is shown in figure 8.7.

Figure 8.7: Simulated model B21 robot and its real counterpart “Gillespie”

The robots mechanics are arranged such that it behaves as if it were a differential wheeled robot with a single axle, that is, it can turn on the spot rotating around the centre of its body. For the simulation we therefore model it using the *differential wheels* VRML node provided by Webots.

The other important physical features, as far as these experiments are concerned, are two rings of sensors which are mounted around the B21’s trunk. Each ring consists of 24 sensors mounted at even intervals around the body. Both sets of sensors rotate as the trunk of the robot turns such that the angle of each sensor remains fixed relative

to the direction of travel of the robot, that is, the same sensors always face forward, etc The lower ring of sensors, mounted approximately 37 cm from the ground are infra-red (IR) sensors. The IR sensors, shown as series of small grey rectangles on the simulated robot in figure 8.7, bounce infra-red off targets to obtain a distance reading. Due to background infra-red they detect objects over a relatively short distance (typically less than a metre) and suffer from a very high amount of noise. The higher ring of sensors, mounted approximately 74 cm above ground level, are sonar sensors. These are shown as gold disks in figure 8.7. Sonar sensors have a much greater range, around 6 metres, and suffer from significantly less noisy readings. The sensor ranges and noise profiles were modelled on measurements obtained from the simulator's real counterpart[7]. Details of the models used can be found in appendix P.

Internal processing of sensory data

The sensor readings obtained by the simulated robot from its environment are processed before they are presented to the learning algorithm. Two approaches to internal processing were implemented, one representing a *fixed* agent, the other a *modifiable* (specifically an *active perception*) agent.

The *fixed perception* robot design combines the sonar and IR readings into a short vector which summarises the distance to the nearest *obstruction* in four orthogonal directions; straight ahead, left, right and behind. Objects are only considered to be an obstruction if they fall inside a imaginary tunnel of space, slightly larger than the robot's dimensions, in the direction that the reading was given for. Distance readings for both sonar and IR sensors are converted to straight line distances from the sensor and then using knowledge of the robots design the values are converted into distance readings along one of the four tunnels. The distance to the closest obstruction is then taken and the values categorised into three ranges representing the classes:

- "too close" if the reading is less than 35cm,

[7] Sample readings were collected at fixed distance intervals while backing the real robot "Gillespie" away from the flat surface of a white wooden door in the laboratory.

- “near” if the reading is less than 1 metre but greater than 35cm,
- “far” if the reading is greater than 1 metre.

Note that all distances are given from the centre of the robot, the robot’s radius being approximately 25cm.

Thus the learning algorithm receives a very much compressed observation space, for example, $\langle far, near, near, far \rangle$ when standing in the middle of a corridor and facing down it. This was a deliberate design choice as it both reduces the observation space that the learning algorithm has to sample and creates many aliased locations, the latter being of great interest from the point of view of these experiments.

The combining of sonar and IR readings, as carried out above, results in a safer design than relying on either just sonar or IR. The existence of an obstacle will be flagged even if it is only detected by one of the two senor types. This design will be largely blind to the existence of the discontinuous sections of walls.

The *active perception* robot design generates an identical summary vector, as the fixed perception design. The additional active perception component consists of a single additional perceptual action, whose selection results in the gathering of additional sensory data, which is then processed to detect discontinuities between the sonar and IR readings.

When the active perception action is selected the simulated robot remaining stationary, and obtains ten sets of readings for every sensor. These ten sets of readings are then averaged to obtain a better noise to signal ratio, especially for the IR sensors. Using ten samples, the distance over which the IR sensors can reliable detect obstacles is extended from 1 metre to 1.5 metres (as measured from the centre of the robot). Once the samples have been averaged, pairs of sonar and IR readings are then compared. The sensors are paired on the basis of being mounted directly above each other on the robots external casing, as shown in figure 8.7. If either sensor in a pair detects an object which it measures as lying within 1.5 metres of the centre of the robot, then the distance registered by its counterpart is checked. If the distance readings of a pair of

sensors differ by more than 60cm then it is deemed that a discontinuity exists, and the type of sensor that registers the object as being closest is noted.

The results of these comparisons are summarised as a vector containing four entries, one for each of the four orthogonal "tunnels" that the distance measurements are reported for. Each entry in the summary vector can take one of four values:

- "None", when no discontinuity is detected for any of the sensor pairs that make up the distance reading in this direction,
- "Sonar", when, at least one pair has been found in the group of sensors that contribute to the distance reading in this direction, such that an object has been detected within 1.5 metres of the centre of the robot, and the sonar estimates its distance to be closer by more than 60cm compared to the distance estimated by the IR sensor,
- "IR", when the same conditions hold as above for "Sonar" but the IR estimates the object to be closer by more than 60cm,
- "Both", when the same considerations hold as for "Sonar" and "IR", but *both* possible discontinuities (that is, sonar closer and IR closer) have been detected among the set of sensor pairs that make up the distance reading in this direction.

Note that each sensor pair will generally contribute to two "tunnel" distance readings, that is, a front left sensor will contribute to both the straight ahead and left distance readings. So if a discontinuity is found with this sensor it will be reported in both these directions. As an example the simulated active perception robot could receive a vector $\langle None, IR, Sonar, Both \rangle$ if it chose to execute the perceptual action whilst standing in the corridor facing the open laboratory door in figure 8.6 (order of elements in the vector are front, left, right and rear).

Physical actions

Both the *active perception* and *fixed perception* versions of the robot have the same four physical actions available to them:

- turn left,
- turn right,
- turn around,
- move forward.

None of these physical action are simple ballistic actions, each represents a behaviour that is executed for a short period of time, and each behaviour is modified depending on the information obtain from the robots sensors. This is all hidden from the learning algorithm which simply selects of one of the above four physical actions (or possibly the perceptual action) to execute.

Turn left, *turn right* and *turn around* actions execute a turn on the spot of approximately 90 left, 90 right, or 180 left respectively. The robot then attempts to square itself up with reference to *all* walls that it can detect using its sonar and IR sensors. What is considered to be "square" will be a balance between the measurements made off of adjacent walls.

Selection of the action *move forward* results in the execution of a combinations of five behaviours. The selection of which depending on raw sonar and IR distance measurements. The five behaviours are:

(i) move forward and centre self in corridor,

(ii) move forward and maintain a constant distance from both walls,

(iii) move forward and maintain a constant distance from the left wall (ignore right),

(iv) move forward and maintain a constant distance from the right wall (ignore left),

(vi) move forward ballistically.

If walls are detect on both sides of the robot which are closer than two metres from the robots centre, and the walls are judged to be parallel, then the robot will travel

forward in such a manner that it moves toward the centre of the corridor. Following the execution of a series of such "move forward" actions it will end up travelling down the centre of the corridor.

If any of the conditions for being in a corridor fail, for example, if the test that the walls are parallel fails, which can happen as the robot nears junctions, then the robot attempts to maintain a constant distance to either one, or both, walls whilst moving forward. The condition that the robot applies in this case is whether the walls lie within 10cm of the target values which were set when either a turn action was completed, or when the robot still believed itself to be travelling down a corridor.

If both wall are with 10cm of their target values then it tracks both, turning slightly as it moves forward so as to minimise the error between the sensor readings and the target values for both walls. If the difference between a target value and that measured for a wall exceeds 10cm, then the robot manoeuvres so as to minimise the error on the one wall that remains within 10cm of its target reading. If the distance to both walls differ from their respective target values by more than 10cm, then the robot travels forward ballistically.

All of the above "move forward" behaviours will be halted immediately should any object be detected within 35cm of any of the forward facing sensors of the robot (as measured from the robots centre). Given the robots diameter of around 25cm, this gives a buffer zone of 10cm around the robot.

A combination of "move forward" behaviours are executed such that the robot travels forward a distance of approximately 35 cm each time the *move forward action* is selected.

8.2.1.4 The task

The task is for the robot to learn, is a policy that navigates it to the laboratory and in through the open door, when started at random locations anywhere in the corridor space. Each action executed, whether successful or not, costs the agent a penalty of

-1. This includes both physical and perceptual actions. The goal is deemed as being reached as soon as the whole of the robot's chassis has crossed over the threshold and into the laboratory. At this point the robot receives a reward of 1001, which, less the penalty of -1 for executing an action, results in a nett reward of 1000. During learning trials the robot, on entering the laboratory, is immediately relocated to a random starting location somewhere in the corridor. The robot is told that it has been relocated and uses this information to reset the eligibility trace maintained by the learning algorithm. It will, on receiving this relocation signal, also attempt to square itself up off of adjacent walls.

8.2.1.5 Learning algorithms and parameters

Each trial took around a day and a half to run[8], with this long duration being entirely down to the speed of the Webots simulator. This, coupled with a limited number of Webots licences, constrained the number of trials that we were able to run.

We restricted ourselves to two learning algorithms; SARSA(λ) and CEQ(λ) and one set of parameter values: eligibility trace decay rate $\lambda = 0.9$ (using replacement traces), learning rate $\alpha = 0.1$, discount rate $\gamma = 0.9$, and ε-greedy/greedy action selection. The scheduled used for ε starts with it equal to 0.2 and decays its value linearly to zero by the 200,000th action-learning step. From the 200,000th step onward the learning algorithm acts greedily. To avoid the eligibility trace from becoming too long it was truncated through the removal of values less than 10^{-5}.

We ran ten trials for each combination of robot design (fixed or active perception) and learning algorithms; SARSA(λ) and CEQ(λ). This results in a total of forty trials, each trial was run for 800,000 action-learning steps. At the end of each trial the policy learnt was evaluated.

[8] Running on a machine with a single 3GHz Pentium 4 CPU and 1Gbyte of memory.

8.2.1.6 Noise models

The Webots simulator has a simple built-in model of noise and wheel slippage for the *differential wheels* node. This is the node used as the robot's base. The noise model consists of adding white noise with a maximum magnitude of $\pm 10\%$ of the size of the movement to each command. Different noise being added to each wheel.

We simulate stochastic noise models for the sonar and infra-red sensors based on readings taken from the B21 robot "Gillespie". Details of the sonar and infra-red sensor models, including noise parameters, can be found in appendix P.

8.2.1.7 Evaluation

Once each trials was completed the policy learnt was evaluated. This was done by following the policy greedily with the learning rate (α) set to zero to ensure that no learning occurred. The robot was started, in turn, from ten selected locations and the number of actions taken by the policy to reach the goal was measured. If the robot failed to enter the laboratory after 4,000 action steps the run was aborted, and the figure of 4,000 steps was reported as the path length. The same ten starting locations were used for evaluating each of the forty policies. The approximate position of the starting locations are shown in figure 8.8.

The starting locations were arrived at by a process of random generation followed by manual screening to ensure a more-or-less even spread around the corridor space. The direction that the robot faces for each stating location is the angle that it would typically arrive at when squaring itself up. Intermediate angles were avoided as they would amplify the effects of random noise which will occur in evaluating these policies. For example, if the simulated robot were started at an angle close to 45, it could, depending on sensor noise, initially square itself up facing either 0 or 90.

Given the stochastic nature of the environment and thus the possibility of randomness effecting the evaluation of policies it would be preferable to record multiple runs from

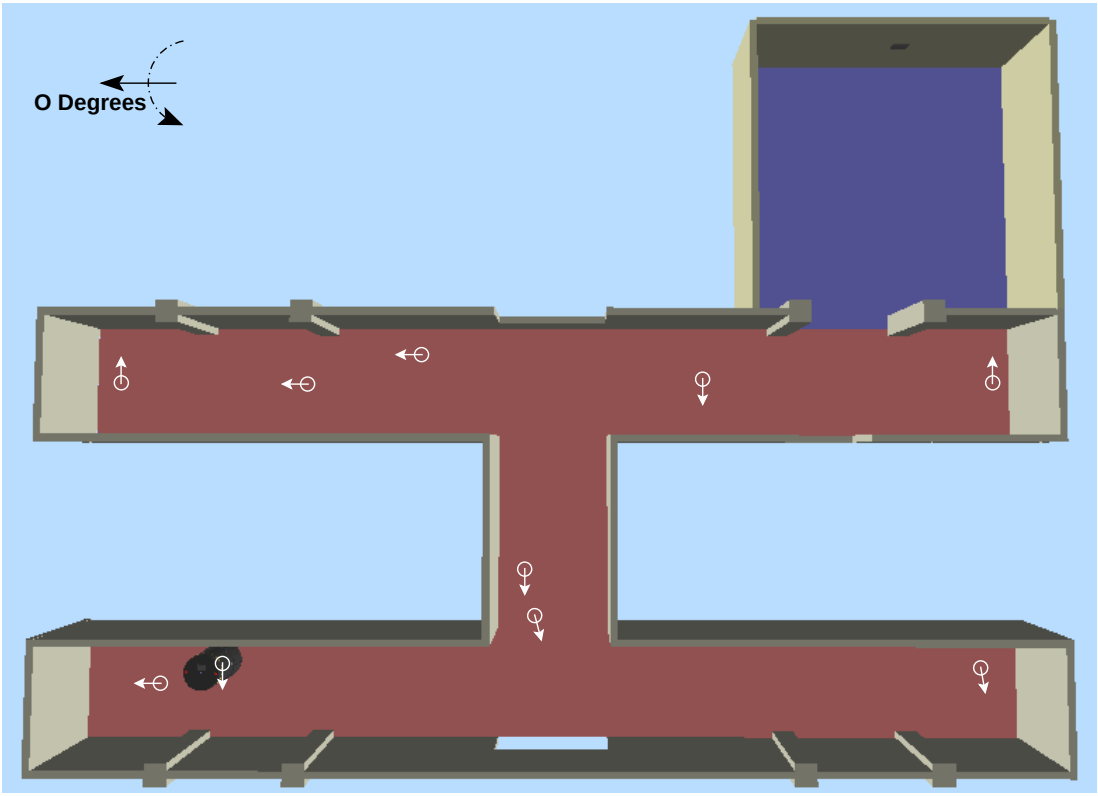


Figure 8.8: Plan of corridors showing policy evaluation starting locations

each start state. Unfortunately practical considerations, in terms of processing time and availability of Webots licences, have limited us to one run per starting location, per policy.

8.2.2 Results

Our primary interest is in the learning of satisficing policies, that is, policies that reliably achieve the desired goal. In this case our measurement of satisfiability is reduced to testing whether a particular policy can reach the goal from the ten sample start states are as discussed in section 8.2.1.7. A further constraint is that each policy evaluation run has to be completed in a reasonable time. We consider a limit of 4,000 action steps to be reasonable. Table 8.6 sets out an analysis of the results on this bases, broken down by choice of learning algorithm and design of the robot's perceptual system.

Using the same evaluation runs we also calculate mean path lengths for the policies. These results are presented in table 8.7. Note that in this table, where a run did not reach the goal in 4,000 steps, then a value of 4,000 steps is taken as the path length. A average figure close to 4,000 thus indicates that the policies, on average, performed poorly. To give a better baseline for comparison we also ran evaluation runs with the robots actions being selected completely at random. For both robot designs we ran ten evaluation runs with the robot selecting actions at random from each of the ten starting positions. The average path lengths obtained for this robot are also presented in table 8.7.

In table 8.8 we present statistics for the best satisficing policies learnt for each combination of learning algorithm and robot design.

8.2.2.1 Comparison of active perception and fixed perception

Examining the first table 8.6, we see that the active perception robot design does indeed perform better than the fixed perception design. The additional perceptual action allowing this design of robot to learn satisficing policies more easily.

learning algorithm \ robot design	fixed perception	active perception
CEQ(λ)	2	5
SARSA(λ)	0	6

Table 8.6: Number of policies (out of the 10 learnt) that reached the goal from *all* sample start positions in less than 4,000nd action steps. Results present for CEQ(λ), SARSA(λ) and both *fixed* and *active perception* robot designs

learning algorithm \ robot design	fixed perception	active perception
CEQ(λ)	2615.7±1640.9	1299.1±1436.7
SARSA(λ)	3245.4±1422.4	1369.2±1603.9
Random Actions	3300.5±1278.7	3475.9±1093.3

Table 8.7: Mean path lengths as measured using ten sample start positions. Results present for CEQ(λ) and SARSA(λ) are averaged over the ten policies that were learnt for each combination of learning algorithm and robot design. The average path length for an robot selecting actions at random is also shown for both robot designs. These latter two figures are calculated using ten runs from each of the ten start positions. Values shown are mean ± standard deviation

robot design and algorithm	mean	minimum	maximum
fixed perception & CEQ(λ)	291.8±250.6	50	764
fixed perception & SARSA(λ)	—	—	—
active perception & CEQ(λ)	87.3±42.5	30	169
active perception & SARSA(λ)	219.4±171.9	11	569

Table 8.8: Statistics for best (lowest mean path length) satisficing policy for each combination of learning algorithm and robot design. Values show are mean ± standard deviation, plus minimum and maximum path lengths when evaluated from the ten sample start states

In making this comparison we can be confident that the problem is learnable for the fixed perception design of robot in that two satisficing policies were learnt by CEQ(λ). In addition to this we can be reasonably sure that our choice of 4,000 action steps as a ceiling for evaluation has not created a significant bias against policies learnt by the fixed perception robot, in that table 8.8 shows that the maximum path length, for the best performing policy learnt by a fixed perception robot, is only 764 actions.

There is a difficulty here in testing whether a policy is satisficing or not as given the existence of stochasticity in the simulated environment there is a chance that any policy will eventually succeed given sufficient time. Given that the reward scheme used is designed to encourage learning minimal path policies, a judgement has to be made as to what is an unreasonably long path. In the case of the real world, additional considerations like the robot's battery life would help in this judgement and would form part of the definition of what was intended by satisficing.

The difference in average path lengths for fixed perception and active perception robot designs, as shown in table 8.7, emphasis the improvement in learnability of the problem with the addition of active perception. The results for the policies learn by the active perception robot are significantly better than a robot which selects actions completely at random. By comparison, the average results for the fixed perception robot, especially for the policies learnt using SARSA(λ), are close to that which would be achieved by an robot acting completely at random. Student's t-test for unequal variance indicates that there is no statistical significance between the result for SARSA(λ) and that achieved by selecting actions at random for the fixed perception robot (probability of 0.77 that they are drawn from the same distribution).

Given SARSA(λ)'s failure to achieve anything better than random action selection for the fixed perception agent we only use the results for CEQ(λ) to compare the two agent designs. Looking at the mean path lengths achieved with CEQ(λ) — that is the two results comprising the top row of table 8.7 — we find that the difference in average path length is statistically significant, with a tiny probability of 8×10^{-9} that the results are derived from the same distribution. The reduction in the mean path length can therefore be taken as an indication of the improved performance achieved though the addition of

active perception to the robot design.

The difference between the *best* achieved by the active perception robot and the fixed perception robot is shown in table 8.8. We see that the best performing *active perception* policy, learnt by CEQ(λ), had a mean path length of 87.3 steps when evaluated, with the longest path taking a total of 169 actions. The best performing *fixed perception* policy achieved an average path length of 291.8 actions, with the longest path taking 764 action steps. The difference between the best mean path length for the active perception and fixed perception robot designs is statistically significant with the probability that both samples are drawn from the same distribution being equal to 0.03 (using Student's t-test for unequal variance).

8.2.2.2 Comparison of CEQ(λ) and SARSA(λ)

Comparing the performance of CEQ(λ) and SARSA(λ) based on the results presented in table 8.6 is difficult. CEQ(λ) in conjunction with the fixed perception robot learns two satisficing policies compared to none for SARSA(λ). Conversely SARSA(λ) learns one more satisficing policy than CEQ(λ) for the active perception robot design, and summing over both robot designs leaves CEQ(λ) in the lead by one satisficing policy to SARSA(λ)'s total of six. Given the small size of this sample, a difference of one (or even two) satisficing policies is probably not significant.

If we look at table 8.7 we can compare CEQ(λ) and SARSA(λ) based on the average path lengths of the policies learnt. We observe that CEQ(λ) has the lowest mean value in the case of both robot designs. Unfortunately the result for the active perception robot design is not statistically significant. Applying Student's t-test (for unequal variances) to these figures indicates that the difference in mean path length between CEQ(λ) and SARSA(λ) is statistically significant when using the fixed perception robot ($p = 0.004$). However, when using the active perception robot there is no statistically significant difference between the two algorithms' performance ($p = 0.75$). We note that for the active perception robot the average path lengths learnt by CEQ(λ) and SARSA(λ) both differ from the result achieved when selecting actions at random

by amounts that are statistically significant ($p = 4 \times 10^{-25}$ and $p = 3 \times 10^{-21}$ respectively). Whereas for the fixed perception robot only the result for CEQ(λ) differs with any statistical significant with that achieved by selecting random actions ($p = 0.001$). We therefore conclude that the detectable difference between CEQ(λ) and SARSA(λ) for the fixed perception design is probably entirely due to the two satisficing policies learnt by CEQ(λ). Thus it is unwise to claim the existence of any clear difference between the two algorithms based solely on the above results.

8.2.3 Discussion

The simulated B21 robot experiments successfully demonstrates the extension of the active perception approach into a more complex and stochastic environment. They clearly shows that giving additional abilities to the robot can make tasks more learnable. Finally, the experiments show that neither of the perceptual spaces available to the active perception robot have to perfectly disambiguate the task in order to be useful, that is, successful policies were learnt even though both views of the world alias large portions of the environment.

These experiments also reinforce the observation made in chapter 4 that extending the abilities of an agent can result in better policies, for example, reduced path length.

Given the consistently good results achieved with CEQ(λ) in section 8.1 when both $\lambda = 1$ and $\gamma = 1$. It would be interesting to test what the results would be on this problem at this extreme point in the parameter space. Unfortunately computational resources and time did not permit this investigation.

8.2.3.1 Hypotheses

This section demonstrates points (iv) and (vi) of the chapter hypothesis. That is, "that the active perception approach can be successfully be extended to more complex, stochastic environments that more closely resemble the real world" and "that CEQ(λ)

performs as well on stochastic tasks as it does on deterministic grid world problems. That is that the introduction of stochasticity does not degrade it performance compared to that of SARSA(λ)."

8.3 Chapter Summary

This chapter presents results which show that:

(i) Active perception's performance is significantly better than either the basic or fixed-SE approaches which were tried. Active perceptions performance is comparable with the results achieved by an alternative modifiable approach (specifically 1-bit Memory agents). These results supports the expectation raised in section 2.3.1 that differing underlying assumptions should lead to a marked difference in performance between fixed-SE and modifiable approaches and similar levels of performance for agent extensions which fall within the same approach.

(ii) There seems to be little to be gained by treating perceptual actions different to physical actions as the consistent representation (CR) approach proposes [Whitehead, 1992].

(iii) U-Tree fails to learn reliably on half of the problems tried, and where it did learn it appears to be continuously adding distinctions to the tree for aliased states. In addition, U-Tree's computation requirements are excessive compared to the direct reinforcement learning algorithm approaches, and there is a need to truncate the instance chain in order to contain memory usage.

(iv) These experiments provide further evidence, in addition to that shown in chapter 7 that CEQ(λ) outperforms SARSA(λ) on POMDP problems. In addition the results show that CEQ(λ)'s performance is robust over a greater region of the parameter space than SARSA(λ). We also demonstrate that CEQ(λ) can learn successfully in stochastic environments.

(v) The simulated B21 robot experiments show that the principle of active perception can be extended successfully to complex stochastic environments which more closely resemble real world problems.

(vi) The B21 experiments also demonstrate that neither of the perceptual spaces available to the active perception agent have to perfectly disambiguate the task for the agent to be able to successfully exploit the additional perceptions that are available to it.

In demonstrating these results this chapter contributes significantly to the main hypothesis of this thesis. We show that the performance of active perception agents is comparable, and often better than other approaches found in the literature. We also show that good results can be obtained without having to distinguish between perceptual and physical actions. For this latter point we demonstrate results for algorithms that make no distinction, which exceed the results obtained for an algorithm (the Lion algorithm) that does make a distinction.

Chapter 9

Evaluation and Discussion

Chapters 4 to 8 present experimental results in support of our thesis that active perception is a useful approach to learning in POMDP environments, and that the concept of Consistent Exploration can improve the performance of reinforcement learning algorithms when learning deterministic policies for POMDPs. In this chapter we review these results before presenting other general arguments in support of our thesis.

9.1 Review of Results

The experiments presented in the previous chapters, especially chapters 4 and 8, have shown that not only is active perception a viable approach to learning in partially observable grid world, but, in terms of reliability of convergence to satisficing policies, it is only matched by internal memory agents.

The results indicate that state estimation approaches, although theoretically more powerful, are in some cases unable to converge on satisficing policies, and that in the case of U-Tree, their complexity impose an impossible burden on computational resources. By contrast the simplicity of direct reinforcement learning, which lies at the heart of the active perception approach, has a much smaller computation burden.

The results obtained also support the expectation raised in section 2.3.1 that the difference in the underlying assumptions between *fixed-SE approaches*, that is, U-Tree or Lion Algorithm, and *modifiable approaches*, such as Active Perception or 1-bit Memory Agents, should lead to a marked difference in performance between the approaches and similar performance for agent extensions which fall within the same approach. This we clearly observe in section 8.1.

We also looked at various approaches to improve the performance of direct reinforcement learning algorithms on partially observable environments in chapters 5 and 6. The approaches examined were inspired by theoretical results presented in the literature, unfortunately none appeared in practice to improve the likelihood of convergence of direct reinforcement learning algorithms on partially observable environments. We did however observe that curtailing exploration seemed to be an important factor, a fact, that apart from results presented by Lanzi [2000], has been largely ignored in the literature.

The experiments also clearly demonstrate the superiority of a new temporal difference algorithm CEQ(λ), which we introduced in chapter 7. This direct reinforcement learning algorithm can be substituted directly for SARSA(λ). We show that its performance is better than SARSA(λ) over a large portion of the parameter space, but especially at the extreme point where the discount rate and eligibility trace discount factors are equal to 1. Operation at this extreme point of the parameter space is of interest due to theoretical predictions of policy stability in partially observable environments given these parameters, see discussion in section 8.1.3.1. Not only does CEQ(λ) out perform SARSA(λ) at this extreme point of the parameter space, but it also tends to be the area of the space where it performs the best. The one slight draw back of CEQ(λ) compared to SARSA(λ) is that it converges slightly slower, especially for low values of α.

The experimental chapters also show that CEQ(λ) should be preferred over another algorithm MCESP (Monte Carlo Exploring Starts for POMDPs) [Perkins, 2001, 2002b] for which good theoretical properties have been shown. MCESP, being a Monte Carlo based approach, has a slow rate of learning with updates only occurring at the end of episodes, this compares to CEQ(λ) which updates its Q-values after every action.

There is also a problem of having to artificially end episodes to break loops in the case of MCESP and the need to move the agent to a new random location at the start of the next episode. CEQ(λ) can select a new exploratory action at any time and thus break loops naturally and explores without the need to introduce discontinuities into the agent location in the world.

The CEQ(λ) algorithms has been demonstrated learning successfully both on deterministic and stochastic POMDPs and we have reported a trial that shows it performs reasonably on deterministic MDPs.

9.2 What are the Advantages of Active Perception

9.2.1 Increase range of satisficing policies

The main aim in providing active perception to an agent is that the additional abilities should increase the solvability of POMDP tasks that the agent faces. In this regard the experiments presented in the preceding chapters demonstrate that the addition of arbitrary perceptual actions does increase agent flexibility and allow good satisficing policies to be learnt across a variety of test worlds. The results overall are comparable to those achieved with the equally simple modification of providing agents with internal memory bits which they can manipulate. Three advantages which active perception has when compared to existing techniques are:

- the agent can continue to act optimally following a 'kidnapping' or resetting event;
- no single observation-space need fully disambiguate the task;
- reduced computational burden (see sections 9.2.4 and 9.2.5).

Each of these points are discussed in detail below.

9.2.2 Acts optimally following kidnapping or resetting

A memory based method, whether building an internal model, or simply manipulating internal memory bits, will generally perform badly following a kidnapping or reset event. By kidnapping we mean the unexpected transportation of the agent from one location in the agent's world to another, independent of the agent's own volition. We assume that the location the agent is transported to lies within its normal range of experiences, that is, the location is not novel to the agent, just the means by which it arrives there. A 'reset' is where the agent's processing is interrupted in such a way that its memory of previous events is erased. Following either of these events, any memory based process will perform badly. In the kidnapping case the memories that the agent relies upon to disambiguate the current state will no longer be relevant, in the reset case the memories will be no longer be available. The result of either of these events is that the best a memory based agent can do is to execute policy actions based on its best guess at where it has been moved to, until it has seen an identifiable observation or conjunction of observations, such that it can once more accurately determine its state in the world. In following such a strategy it is unlikely that such an agent will act optimally, where optimality is measured in terms of the minimum path to some goal, following such an event. In comparison, an active perception agent has the advantage that the cues on which it relies are located in the external environment. As it maintains no memory trace, the situation following either of the above events is identical, as far as it can tell, to the situation where it has just executed a physical action and arrived at the new locations. Thus it just follows its policy as if nothing has happened, obtaining additional observations from the outside world in order to disambiguate any aliased locations. Thus provided its reactive policy is optimal it continues to act optimally following kidnapping or resetting of the agent

9.2.2.1 Vulnerable to changes in the world

As an active perception agent relies on cues that are external to it, it will be vulnerable to other events, such as the movement of the items in the world that it relies upon.

How significant this vulnerability is will depend on the characteristic of the objects that provide it with additional cues. For example, a person standing in a corridor would provide a poor cue as they are likely to move away. This contrasts with using the colour of the corridor walls which are unlikely to be repainted any more frequently than every five years and even then the colour is unlikely to change. The length of time, and number of repetitions that it takes for any reinforcement learning algorithm to converge on a satisficing policy, an issue which we touch on in section 10.11.1, will work in the favour of an active perception agent in this case. Assuming it takes an agent the equivalent of several days to learn a task, then any objects in its environment that are liable to be moved in the short term will, in all likelihood, have moved over the course of training. So even if at one point the policy had been such that it had relied on the presence of a particular object, its movement will have suppressed its utility. Objects that remained static over the course of several days are likely to remain so over the longer terms so they should retain their utility. In this way the agent should end up preferring cues which are reasonably static in its world. Demonstrating how reliably an active perception agent would perform in a world where change is ongoing is an area of future research that we consider in section 10.11.3. It should be noted that other forms of agent, for example, internal memory bit agents, will be equally vulnerable to certain changes in the world. This will depend on whether the difference is detectable given the agent's set of observations. Certain gross changes, like closing off a corridor, will typically disrupt all agents.

9.2.3 No one observation-space need fully disambiguate the task

The active perception approach can be viewed as providing an agent with multiple views, or viewpoints, of the current state of the world. The experiments with the Active Perception Agent (chapters 4 – 8) and simulated B21 robot (chapter 8) demonstrate an important point relating to these multiple viewpoints. This is that, in order for optimal or just good satisficing policies to be learnable, individually none of the viewpoints need provide a set of observations that completely disambiguate every state of the world. Each viewpoint that is available to an agent may contain observations that are

aliased, but, in the ideal case, the states where aliasing occurs for one view of the world are not the states where aliasing occurs for an alternative viewpoint. Thus by judiciously selecting alternative views, through the selection of perceptual actions, an agent can switch to a non-ambiguous view of the world for each state, and therefore select the best action.

To illustrate this point, consider the simulated B21 robot problem in chapter 8. The robot's task is to navigate to a room which is off one leg of a set of corridors. To carry out this task the robot uses readings from a set of sonar and infra-red sensors which encircle its body. The readings from these sensors are processed to produce two viewpoints. One viewpoint contains information about the proximity of objects in four directions; straight ahead, due left, due right and behind. The second viewpoint contains information about which type of sensor can detect the obstacle. Given that the sonar sensors are mounted some 37cm above the infra-red sensors, this viewpoint provides information about the shape of the objects detected. In the task presented the only objects, other than the robot itself, are the corridor walls. The wall have cutouts at various points in order to provided some locations where distinct observations exist in the second viewpoint. Neither viewpoint completely disambiguates the whole space. Most of the space looks identical to the second viewpoint except where cutouts exist. The first viewpoint can distinguish between travelling down corridors and junctions, but it cannot tell which of the four corridors it is in. Nor can it tell the difference between which of the two T-junctions it is at. The two viewpoints are however complementary as the cutouts that exist at each of the T-junctions can be distinguished from each other. Thus the agent, by selecting the second viewpoint when standing at a T-junction, can resolve the ambiguity between the two T-junctions and then select the correct direction in order to reach the room.

9.2.4 Reduced computational burden

The computational burden of the active perception approach is significantly less than approaches which attempt to learn internal models which use history to completely

disambiguate every state of the world, such as U-Tree. Active perception, in common with methods that use internal memory bits which are manipulated by the agent, has the much less ambitious aim of adding sufficient flexibility to the agent to make good reactive policies possible. Active perception and internal memory bit agents simply seek to find a sequence of actions for which the world appears to act in a Markovian manner, and which arrive at the goal state. This is a significantly simpler problem than attempting to render every state of the world Markovian. This is a reiteration of a point raised in section 10.1 where we argue that this constitutes a significant distinction between approaches like U-Tree and k-step Markov methods and approaches which learn reactive policies using, for example, CEQ(λ) or SARSA(λ) coupled with Active Perception or N-bit Memory Agents.

This reduced requirement of not having to render all observations into Markovian states, makes approaches that use active perception or internal memory bits significantly more computationally tractable than for example U-Tree. As illustrated in chapter 8 U-Tree suffers from a requirement for both massive processing and memory requirements.

9.2.5 Avoids multiplicative effect of extending observation space

Active perception can be arranged such that the increase in the size of the observation-action space which is incurred in adding additional sensory information is only additive rather than being multiplicative.

Let us assume that two complementary sets of observations from different sensors exist, Ω^x and Ω^z, which are constructed such that when aliasing occurs in one, it does not occur in the other. Let us further assume that it is possible for the agent, if it wants, to obtain an observation of each type simultaneously, that is, at time t the agent can obtain observations o_t^x and o_t^z where $o_t^x \in \Omega^x$, $o_t^z \in \Omega^z$. What advantage is gained in providing an active perception action that selects between o_t^x and o_t^z when, by combining the two observations, it is possible to arrive at unique observation for each state of the world? That is to say, we can introduce a new observation, o_t^{x+z}, equal to the tuple

$\langle o_t^x, o_t^z \rangle$, drawn from an observation space Ω^{x+z} where, by the definition of Ω^x and Ω^z, no aliased observations occur. The answer to this question lies in the mapping between the observations and the size of the observation-action space that the agent has to learn.

If we assume that observation o^x is a vector of n sensor readings each of which can take m different values, then the space Ω^x contains m^n possible observations, that is, $|\Omega^x| = m^n$. Similarly, if o^z is a vector of p sensor readings each of which can take q different values, then the size of $|\Omega^z| = q^p$. Given this, the space of the combined observation, Ω^{x+z}, has size $|\Omega^{x+z}| = m^n \times q^p$.

We note that the figure for $|\Omega^{x+z}|$ above assumes independence between the two observation spaces, which may not be the case. In addition, all possible combinations may not occur in practice. However $|\Omega^x|$ and $|\Omega^z|$ may also be reduced if the sensor readings are not independent and if certain configurations simply do not exist in the world. Thus, although the calculated increase in the size of the observation space from combining the two observations into one might not hold in all cases, there is very likely to be a significant multiplicative increase in the observation space size when combining the two sets of sensory readings.

The size of the observation space is important as reinforcement learning algorithms attempt to learn actions for each possible observation. In their most basic form this is done by storing an observation-action table with sampled values for each observation action combination that is possible. Thus, for example, if the range of actions that can be selected is contained in the set A, which is uniform across the whole of the observation space, then the number of points that the reinforcement learning algorithm has to sample for the combined observation space is $|A| \times |\Omega^{x+z}|$.

For the active perception approach we can arrange for one of the observation spaces, say Ω^x, to be the 'normal' view seen by the agent. Suppose we now add a single action to the set of actions, A, which when selected obtains the observation from the second space Ω^z. In this case the observation-action space has size $(|A|+1) \times |\Omega^x| + (|A|+1) \times |\Omega^z|$ (assuming that the set of actions is uniform across both observation spaces). Thus the increase in the observation action space is approximately additive in the size of the two sensory space readings.

For most cases the multiplicative effect of combining the two observations into a single non-aliased observation, severely increases the size of the observation-action space when compared to the additive effect of combing the two sets of observations using active perception, that is,

$$\begin{aligned} |A| \times |\Omega^{x+z}| &> (|A|+1) \times |\Omega^{x}| + (|A|+1) \times |\Omega^{z}| \\ |A| \times m^n \times q^p &> (|A|+1) \times (m^n + q^p) \end{aligned}$$

The disparity between the relative size of the observation-action space, where each observation contains all available information, and the size of the active perception observation-action space, can only be exacerbated if further sensory information is added.

Note that the grid world examples in thesis do not follow this formulation, in these we have used two spaces for the Active Perception agent design; one is an 8-bit space and one an 11-bit space. An example of where we have followed the above approach is provided by the B21 robot simulation in chapter 8. The agent has access to two sets of observations, or viewpoints. The first viewpoint contains information about the proximity of objects in four directions; straight ahead, due left, due right and behind. The readings for each direction taking one of three values; "very close", "within 1 meter", "far". The second viewpoint contains information about which type of sensor can detect the obstacle. This information is also given for the above four directions, and the values these readings can take are "sonar closest", "infra-red closest", "both closest[1]", "no difference". The theoretical size of the observation space for each of the viewpoints is therefore $3^4 = 81$ and $4^4 = 256$ respectively. Combining both viewpoints into a single observation space would result in a space with size $81 \times 256 = 20,736$. Given that the robot has four physical actions available, the resulting observation-action space would be $20,736 \times 4 = 82,944$, which represents the number of observation-action pairs that the learning algorithm needs to sample. In comparison, combining the two viewpoints using active perception results in an observation-action space of size $81 \times 5 + 256 \times 5 = 1,685$. Note that a fifth perceptual action is added to the set of actions. The observation-action space that results

[1] This indicates that between two or more pairs of infra-red and sonar sensors, there is disagreement on which type obtains the closest reading.

from using active perception results in a close to fifty-fold reduction in the number of observation-action pairs that need to be sampled. In practice not every observation occurs and examining a typical run from chapter 8 we find that the two state spaces have size 71 and 39 respectively. This results in the observation-action spaces for the two approaches having size $71 \times 39 \times 4 = 11,076$ and $(71 + 39) \times 5 = 550$ respectively. This still represents a twenty-fold reduction in the number of samples required when using active perception.

The multiplicative increase in the size of the observation-action space will have the effect of slowing learning as it will take the agent longer to sample the increased space. Thus by judicious structuring of active perception, it is possible to reduce the space that the reinforcement learning algorithm needs to be sampled, and thus increase the rate at which a given task is learnt. This could be demonstrated experimentally.

9.2.6 Separate and shared observation-action tables

The perceptual actions available to an active perception agent can map onto separate or shared observation spaces, which in turn map onto separate or shared observation-action tables. An example of a perceptual action mapping onto a separate observation space is given by the B21 robot simulation in chapter 8. Two distinct observation spaces are perceived by the agent, one observation space indicates the presence of obstacles in tightly defined regions of space around the robot, the other sets of observations flag whether there are any discontinuities between the sonar and infra-red readings when attempting to detect obstacles. Observations from the second observation space are obtained though the selection of a single perceptual action, with the second observation space mapping onto a separate observation-action table. This could be scaled up with multiple perceptual actions available to the agent, each of which maps onto a distinct observation space and hence separate observation-action tables, one for each perceptual action.

At the other extreme it is possible to arrange for all the perceptual actions to map onto a single observation space, the same observation space that is observed when executing

physical actions. An example of this would be a visual system where the perceptual actions pan and zoom a camera. In this case the observations obtain from the camera would, in all probability, map onto the same visual space.

It is also possible to envisage a mixed system, when several distinct observation-spaces exist but are shared among the perceptual actions. An example of this is given by the Active Perception Agent, see chapter 5.2.1.3. This agent has two observation spaces, one an eight bit binary string obtained after each physical action is executed, the other an eleven bit binary string which is obtained after executing any one of eight perceptual actions. The distinction between the two observation spaces results in two distinct observation-action tables, but with multiple perceptual actions mapping onto each entry in the second observation-action table.

The choice of how perceptual actions map into separate or shared observation spaces determines how much generalisation an agent makes over observations. The more perceptual actions that map to each single observation, the greater the likelihood of perceptual aliasing of that observation. On the other hand, the more perceptual actions that map to an observation, the greater the generalisation that is being made; though the generalisations made may not always be appropriate. This generalisation (provided it is appropriate), combined with the smaller size of the observation-action space, can lead to a speed up in learning. These savings, when coupled with the observation in section 9.2.3 that it is not necessary for any single observation space to fully disambiguate the task, suggest that sharing observation spaces between multiple perceptual actions, especially where those actions result in common observations, is probably a better option than creating a distinct observation space for every perceptual action. This is obviously an area for future research and we expand upon this in section 10.5.

9.2.7 More leeway in preprocessing of sensory data

It is often noted, for example, McCallum [1995b], that when dealing with robotic agents, the main issue is not the lack of sensory information, but the converse. That is, the volume of raw sensory data produced is so great that it overwhelms learning

algorithms' ability to generalise between similar situations. In developing the algorithm U-Tree McCallum [1995b] sought to deal with this issue through allowing the algorithm to select which sensor readings to sample and which to ignore.

Our proposal is somewhat simpler in that we suggest that existing preprocessing or clustering approaches should be applied to the data *before* it is presented to the learning algorithm. Many good techniques exist for the unsupervised clustering of data; some of which, for example the GWR network [Marsland, 2001], are capable of online continuous learning. The general difficulty in applying such techniques is insuring that the clustering produced contains sufficient information to allow a task to be learnt, whilst being simultaneously coarse enough to ensure the learning algorithm is not swamped. Active perception provides an alternative approach to this problem. Different sensory data, preprocessing or clustering approaches can be run in parallel, with the agent able to use perceptual actions to select between the representations available. The preprocessing or clustering methods used now require less fine tuning as no single observation set has to completely disambiguate the whole state space, as described in section 9.2.3. Provided that they do not alias the same states then a good reactive policy will exist. Even if some overlap does exist a third perceptual action and observation space could be added.

An example of this approach is shown by the simulated B21 robot in chapter 8. The simulated robot has two sets of preprocessed data available to it, neither of which completely encodes the details available from its sensors, and both of which alias many sections of the corridors. The two sets of observation do however provide complementary information, and as the experiment demonstrates, it is possible for the agent to utilise the two sets of observations to learn good reactive policies for this task. If we were to try and learn this task using, for example, raw data from the twenty four sonars, the need to sample such a massive observation space, and the lack of generalisation between similar parts of this space, would make learning so slow that effectively the task becomes impossible to learn.

9.3 Evolutionary Approaches to Active Perception

Results similar to those discussed here have been demonstrated for active perception when using evolutionary approaches, that is genetic algorithms or genetic programming, to configure the agent's control system. Evolutionary approaches consider the modification of a agent's control system through the cross-breeding and mutation of individuals who prove the most fit at some task, the aim being to achieve further improvements in fitness. A summary of the typical approaches taken in evolving robotic agents is given by Mataric and Cliff [1996]. As in the reinforcement learning examples presented in this thesis there is generally a close coupling between the perception and actions of evolutionary agents, in fact the coupling is often at a much more primitive level with the control of individual motors being linked to sensory perceptions. Suzuki and Floreano [2006] and Floreano et al. [2004] present work in which evolved controllers exert control over both perceptual and physical actions. The agents they present are equipped with either a pan/tilt camera or a movable perceptive field through which they can select parts of the visual field on which to focus. The evolved agents successfully learn to exploit the interaction between perception and physical actions in order to complete various tasks such as shape discrimination, simulated car driving and landmark navigation. In all cases the control systems exploits very simple regularities which arise through the interaction of active perception and the scene. Parallels can be seen between this dynamic exploitation of regularities and the exploitation of "Markovian" regularities which occur in the satisficing policies learnt by agents in this thesis (for further discussion of seeking Markovian paths through POMDPs see section 10.1).

We note that the control systems evolved by Suzuki and Floreano [2006] and Floreano et al. [2004] can learn non-Markovian control strategies as the evolved neural network controllers contain recurrent connections which can provide the controller with memories of previous events. This ability should allow such networks to learn better control strategies than our reinforcement learning approach which is constrained to simply reacting to its current perception.

A common criticism of both evolutionary and reinforcement learning approaches is the time required (that is, number of repetitions / generations) in order to learn control strategies. It is unclear from the current literature if either is to be favoured in this regard. There are however techniques by which reinforcement learning can be sped up, for example by providing a teacher which guides the agent through interesting parts of the space, see discussion in section 10.11.1. We are not aware of the existence of approaches which can speed up evolutionary techniques.

9.4 Structure of RL Approach Used for POMDPs

The approach that we have arrived at for reinforcement learning of reactive policies in POMDP environments, has tended to involve the selection of parameter values and learning options, such that the temporal difference (TD) algorithm used begins to resemble the first visit Monte Carlo algorithm (Algorithm 1 in section 6.1). Specifically the experimental results presented in this thesis show that it is preferable to:

(i) use eligibility traces with values of λ approaching or equal to 1. Sutton and Barto [1998] considers that eligibility traces form a continuum between one-step backup TD algorithms and Monte Carlo methods, with long traces, as produced by values of λ close to one, resembling the updates made by Monte Carlo methods;

(ii) select exploratory action in a consistent manner as per CEQ(λ). In Monte Carlo algorithms a single exploratory action is normally selected at the start and then the existing policy is slavishly followed. In applying Consistent Exploration to TD methods a similar effect is achieved.

There are other ways in which our approach results in TD methods that closely resemble the first visit Monte Carlo algorithm:

(a) The best performing algorithm in chapter 8, CEQ(λ), truncates its eligibility trace when either the chosen exploratory action changes or when the policy action changes.

(b) We chose to use replacement traces rather than accumulating traces for eligibility as the former are reported as producing better performance [Singh and Sutton, 1996]. In the same work, replacement traces are also reported as being equivalent to first visit Monte Carlo algorithm, as opposed to the every-visit Monte Carlo Algorithm.

(c) Our code resets the agent's eligibility trace when it has reached the goal and is relocated to a new random start location. Reaching goal locations would represent episode ends for Monte Carlo methods, so clearing the trace at this point enforces a clean break before the new episode and thus bears some similarities to the end of episode update that would occur for Monte Carlo methods.

The results presented in this thesis do not indicate whether any or all of these three similarities have value when learning reactive policies in POMDP environments. We suspect that they will have an effect and discuss how each could be tested in section 10.8.

It is perhaps no surprise that our TD approach has come to resemble a first-visit Monte Carlo algorithm given the theoretical results presented for the first-visit Monte Carlo algorithm by Pendrith and McGarity [1998, 1997], see section 6.1. They indicate that if actions are valued using a first-visit Monte Carlo approach with undiscounted returns, then optimal reactive policies are learning equilibria. However, as we demonstrate in sections 6.1 and 6.3, the existence of equilibrium points for reactive policies under a particular algorithm does not mean that such policies are learnable by that algorithm. We show that first-visit Monte Carlo performs poorly when exploration is introduced.

In the end our preference for a TD style algorithm, rather than Monte Carlo based approaches, is based on the crucial difference that TD approaches continuously update their observation-action values, both speeding learning and avoiding the need to impose artificial episode ends, see section 6.5.

9.5 Varying the Cost of Perceptual Actions

As explained in section 4.2.1.2, in designing the examples presented in this thesis we have deliberately chosen to ensure that the cost of perceptual actions are not less than the cost of the cheapest physical action. This choice was made to ensure that, despite the discounting of future rewards, the most valuable policies are those that reach the goal state, and not policies where the agent exclusively selects perceptual actions ad infinitum. The latter occurs as a consequence of our decision not to distinguish between perceptual actions and physical actions, which means that the learning algorithms are free to select a continual stream of perceptual actions even through their repetition will not advance the agent towards the goal. Couple this with unvarying costs for each action and a stream of cheap perceptual actions, when discounted and summed to infinity, could represent a more valuable policy than the total reward received for reaching a distance goal state, especially after deducting the cost of the finite number of more expensive physical actions which are required to get there.

It would be useful to be able to reflect the true cost of perceptual actions when forming policies, especially as perceptual actions will probably cost less in terms of time and energy than physical actions. We identify three possible approaches to achieve this:

(i) Differentiate between perceptual (information gathering) actions and physical actions (actions that potentially advance the agent towards the goal) and impose a structure which ensures the eventual selection of physical actions. Section 9.5.1 discusses why we believe such an approach to be fundamentally flawed.

(ii) Use undiscounted returns. Section 9.5.2 discusses the advantages and disadvantages of such an approach on non-episodic tasks.

(iii) Instead of having fixed costs for each action, cost actions in terms of the information gained, for example penalise repeated requests for the same information. Section 9.5.3 briefly discusses this approach.

9.5.1 Separate mechanisms for perceptual and physical actions

Unlike, for example, CS-QL or the Lion-algorithm (see section 3.1.1 and 3.1.3), the reinforcement learning algorithms we have chosen to use do not differentiate between physical and perceptual actions. This design decision is based upon the argument that it is not possible to draw a clear boundary between physical and perceptual actions. All physical actions can provide some new information to the agent, through, for example, a change of perspective, and thus can be regarded as perceptual actions. In addition many perceptual actions are made up of physical actions, for example turning one's head to obtain a different view. Only perceptual actions which are not visible to an external observer can really be regarded as not being physical acts, for example shifting visual attention to a different part of the scene without moving one's eyes. Given the above, we consider that the concept of physical and perceptual action types is flawed and any enforced segregation is potentially harmful in limiting the policies which the agent can learn. In the absence of such segregation we cannot imposing an artificial structure which forces the selection of physical actions. Thus we argue that a better approach to reflecting the true cost of actions while also ensuring the integrity of the reward structure is to either (a) signal such structure through varying the cost of actions, as set out in section 9.5.3 or (b) resolve the issue by using undiscounted returns.

9.5.2 Undiscounted returns

Using undiscounted returns removes the issue of a distance goal appearing less valuable than an infinite series of low cost actions. An undiscounted infinite series of low cost actions is guaranteed to have an unbounded cost. Therefore, for an episodic task, this will guarantee that any reward received for reaching the goal state, whatever distance it may be, will always be more attractive.

Unfortunately, undiscounted returns can introduce other problems which need to be born in mind when structuring a task's rewards. One problem is that for minimum path length tasks, a penalty has to be imposed for each action taken, as without discounting

of future rewards no gradient exists across the observation / state values to indicate in which direction an agent should move. For an example of the occurrence of this problem see Wilson's Woods 7 task in section 7.1.4.3. The other major problem is that, for non-episodic tasks, the use of undiscounted returns can lead all observation / state values becoming unbounded and thus the tasks is not well formed.

Although the active perception approach proposed in this thesis should be applicable to any reinforcement learning task, if in forming the policy it is essential that the relative cost of different actions is taken into account then its application is probably limited to episodic tasks. We note that such a limitation will apply to any approach where actions with unequal costs exist and where the selection of only the cheapest of those actions does not enable the agent to move towards the goal state.

A positive observation for the use of undiscounted rewards is that the best results obtained using CEQ(λ) were often obtained with $\gamma = 1$; see for example the experiments presented in section 8.1. There may also be theoretical support for this approach, as Pendrith and McGarity [1998] demonstrate that when learning policies for POMDPs, the use of *undiscounted Monte Carlo* returns ensures that the optimal reactive policy (where it exists) lies at a stationary point in terms of observation-action value updates. This result may apply to CEQ(λ) if, for example, it can be shown that the policy updates when $\lambda = 1$ are equivalent to Monte Carlo returns.

9.5.3 Reward structure related to information gain

In the example tasks considered in this thesis, the world is static. As a result of this, an agent which repeatedly executes the same perceptual action at the same location gains no further information than that which it obtained the first time it selected that action in that location. If the reward structure were modified such that the cost of selecting an action reflected the lack of information gained then an agent could be coerced into not selecting the same perceptual action ad infinitum.

There is a problem in implementing such a strategy which centres on the agent being able to judge the amount of information provided by each action, especially when the

observations it obtains can be aliased. For example, in moving down a long empty corridor, a stream of identical observations could be provided by the action of moving forwards. Penalising this action due to the lack of information gained would seem somewhat counter productive, especially if the goal lies at the end of the corridor. However, if we *do not* want to penalise the move forward action but *do* want to penalise an action by which an agent continually reviews the same parts of some scene, then we are back to the position of wanting to impose a separation between physical and perceptual actions, an approach which we rejected in section 9.5.1 above.

We only envisage such an approach being possible if the rewards are provided by some external observer who is monitoring the agent's actions. This is opposed to our normal conception of reinforcement leaning, where we envisage the rewards being generated internally by the agent's morphology[2]. Further, this external observer will need access to more information than that provided to the agent, such as a less aliased view of the world, in order to make judgements regarding the agent's progress.

If we were to allow such an observer to exist then the most obvious form for it to take is one of a teacher, an idea that has already been proposed to speed learning, see section 10.11.1. A teacher which is able to compute the *productiveness* of an agent's actions could provide inducements to encourage learning of better behaviours. Productiveness would need to be some combination of information gain and likelihood of progress towards the goal. It could be as simple as a measure of whether an action returns the agent to a previously visited state.

There is a further issue with dynamically manipulating the reward structure of a task. Penalising repeated behaviour will make the reward structure non-Markovian, as the rewards will become dependent on the previous actions taken. The performance of direct reactive reinforcement learning algorithms when both the world and the reward structure are non-Markovian has not been tested.

[2]The biological analogy being that the costs for selecting an action represent the losses borne by the agent's body in terms of energy and time and the rewards obtained for reaching a goal correspond to the agent finding a source of nutrition.

9.6 Continuous Observation and Action Spaces

This thesis has only considered discrete observations and actions. We deliberately constrained our research to this area as this allows a tabular representation to be used to store the observation-action values. To apply reinforcement learning (RL) to either continuous observations, continuous actions, or a continuous observation and action space without descretisation requires the replacement of the tabular storage of value estimates with a continuous value estimation function which can generalise over the observation-action space. Introducing such a generalising function even on simple Markov Decision Processes (MDPs) can frequently result in a failure of the RL approach to converge [Boyan and Moore, 1995]. The failure to converge can be due to a number of factors, some of which are: (i) the inability of the generalisation function to represent the required value function; (ii) the observation that the boot strapping updates (as used in most RL algorithms) can result in divergence of value function parameters [Baird, 1995]; (iii) systematic overestimation of state values due to the use of max function in updates [Thrun and Schwartz, 1993] and (iv) extrapolation of the value function beyond the points for which supporting data exists [Smart and Kaelbling, 2000].

There is no practical reason why the active perception approach to learning POMDPs as proposed in this thesis cannot be used with a generalising function in place of the tabular observation-action value estimates. The practical result of such a combination is unknown. However, we observe various similarities between the issues encountered in POMDP environments and those raised by the use of function approximation. When using an observation-action table, the relationship between individual table entries is that imposed by the agent's experience of the world. It is the agent's experience that relates, say, the observation-action pair $\langle observation_{1.0}, action_0 \rangle$ to $\langle observation_{1.2}, action_{0.2} \rangle$ and their proximity in the observation-action space is not significant in estimating their individual values. With the introduction of a generalisation function in place of a table, there is an assumption of some form of neighbourhood influence over the values in the observation-action space, that is, an assumption of a smooth continuum between adjacent observation-action pairs. It it relatively easy to

see that discontinuities in the true observation-action value function could exist which the generalisation function might not be able to represent. In such a case, the effect is similar to that of aliasing two different states which return identical observations. The result in both cases is that the learnt observation-actions values are distorted, which in turn causes both local and global impairment.

We show in this thesis that it is possible to learn satisficing policies in the presence of such impairment as caused by state aliasing, especially if alternative views are made available which provide consistent paths through or around aliased states. One of the keys to learning such policies is the eventual cessation of the selection of random exploratory actions. We note that Boyan and Moore [1995] adopt a similar approach to successfully learn policies with function approximation; that is, in place of one step backups, they assign values based on simulations of following *greedy* policies.

To deal with systematic overestimation errors, Thrun and Schwartz [1993] suggest using eligibility based RL techniques, as the multiple samples should reduce the overestimation. They also suggest using an undiscounted framework, as this avoids the multiplicative effect of discounting which can rapidly reduce reward signals to a level where they are smaller than the generalisation function errors. Both of these suggestions fit with our work.

Taken together, the above observations indicate that our general approach of (a) making alternative view points available through, say, active perception, (b) using the CEQ(λ) algorithm with discount γ and eligibility trace λ parameters close to or equal to 1.0 and (c) continuing learning while pursuing greedy policies, may be of use in dealing with the problems induced by function approximation.

9.7 Wider Applicability of Results

Although the tasks considered here are exclusively navigation tasks, the results are applicable more widely. It is possible to see parallels with any goal directed task that requires the selection of a sequence of actions based on observed states of the world.

This represents a large body of tasks that an embodied and embedded agent could be required to undertake, for example, manipulation or assembling of an object.

9.7.1 Providing agents with oracles

In chapter 4 we introduced Oracles as idealised active perception systems in order to initially validate our approach with 'clean' additional observations. The concept could be taken further with a 'real' oracle that provides accurate information to agents on request, the agents then making use of this information to aid their decision making processes. Our experiments demonstrate that this concept can work successfully with simple reinforcement learning algorithms such as SARSA(λ), and treating the request to the oracle as perceptual actions. This could be a useful approach in the case of multiple agents that need access to some central server, which acts as an oracle, and where central server resources are limited.

Chapter 10

Future Work

This chapter draws together various issues that we have identified in this thesis as requiring further investigation. For each issue we identify the questions raised and where possible we suggest ways of proceeding.

10.1 Seeking Markovian Paths Through Non-Markovian Worlds

In comparing Active Perception agents, N-bit Memory agents and U-Tree in chapter 8, we see a strong distinction in the results obtained between the two groupings, with modifiable approaches out performing fixed-SE approaches. The reason we believe that modifiable approaches are so successful is the constraint that exists on the policies which they can learn, that is, the policies which the algorithms can represent are restricted solely to those which are reactive. In other words, *they learn paths which act in a deterministic and Markovian manner, whilst navigating in a non-Markovian world*. The major advantage in considering only reactive policies, when compare to state-estimation approaches, is that the policy space is constrained by the observations allowed by the environment and the range of action available to the agent. Fixed-SE

approaches on the other hand have to define their own set of representations against which the policy is learnt. The resulting representations might be more compact and allow for better policies, but a significant amount of effort has to go into exploring the space of observation to internal-representation mappings before an agent can begin to form a consistent policy. It is the complexity of building internal representations while simultaneously attempting to learn and act out some useful policy that we believe causes the fixed-SE approaches to fail.

Although the possibility of limiting the range of learnable policies to the set of reactive policies was noted by Chrisman [1992a], see section 3.1.7, we have not seen it explored elsewhere in the literature. Wider recognition of what direct reinforcement learning algorithms actually attempt to learn in non Markovian environments we believe is important. It allows consideration of the issue of what policies are learnable, in terms of what paths exist that exhibit Markovian properties in a given non-Markovian world. A study focusing on the existence of such paths may give rise to insights into the nature of the tasks that are learnable. This viewpoint also opens up the possibility of considering new types of algorithms based explicitly on identifying alternative Markovian paths though the observation-action space.

10.2 Structure of Tasks Used in Our Experiments

10.2.1 Stochastic environments

We show results for a large number of deterministic grid world tasks and one stochastic environment; the B21 robot simulation. It would be useful to confirm results across a large body of stochastic worlds. The number of experiments run using the B21 robot simulation was sufficient to demonstrate that CEQ(λ) works in stochastic environments but insufficient to draw any statistical conclusions with regard to its comparative performance with SARSA(λ). Running sufficient experiments in this environment to gather meaningful statistics would be useful, however, given the the run times in-

volved when using Webots, a larger body of data could be gathered by examine the performance of CEQ(λ) on stochastic grid world problems.

10.2.2 Continuous environment

We have not examined the effect of combining our active perception approach with the use of techniques which are required to learn in continuous state or action spaces. As we indicate in section 9.6, we believe the approach that we advocate for learning POMDP tasks may also produce good results on continuous environments. This would be an interesting area to examine.

10.2.3 Tasks other than "minimum-cost-to-goal"

The problems examined in this thesis are also generally of the form "minimum-cost-to-goal" problems [Perkins, 2002a], that is, each problem has one or more goal states and the agent receives a penalty for each step it takes in reaching that goal. A reinforcement learning algorithm, in seeking to select actions worth the greatest value, will therefore seek to minimise the path length to the goal state. The reward structure of this type of problem will, to a certain extent, drive the agent towards satisficing solutions even when exploration has ceased. The returns received by an agent following a non-satisficing policy, that is, one that fails to reach the goal, will continually be negative, discouraging the selection of the actions that make up that policy. Thus, if the learning rate of the algorithm is not zero, a non-satisficing policy will be forced to change as one of the actions that make it up becomes valued less than alternative actions that have not been selected for a while. Based on this observation, it is possible that the minimum-cost-to-goal problems form a set of problems that are easier to learn than others. There is therefore a need to examine other styles of problem. Alternatives include (i) problems where there are no costs associated with actions, only positive rewards for reaching a goal state (though one example, Wilson's Woods 7, does appear in section 7.1.3.4); (ii) continuous tasks that have no goal state, for example, the New

York City driving problem [McCallum, 1995b], where the aim is to learn to steer a car which is moving at constant speed, such that the car safely overtakes slower moving traffic, whilst not obstructing faster traffic which is coming up from behind.

10.3 Comparing the Relative Difficulty of Tasks

There is a lack of understanding of what makes one POMDP task difficult to learn and what makes others easier to learn. In applying reinforcement learning to POMDP tasks, it would be useful to develop good measures which indicate the difficulty, in terms of learning stationary, deterministic, reactive policies, that a given task presents.

Some work in this area has been carried out by Littman [1992] and Hartley [2004]. They however focus on categorising tasks for which no stationary, deterministic, reactive policy exists, in terms either of the number of internal memory bits an agent requires, or of the number of distinction that the Utile Distinctions Memory (UDM) algorithm has to make, in order that such policies exist.

We suggest that it would be useful to classify tasks where stationary, deterministic, reactive policies do exist, but looking at how difficult particular tasks are to learn. A simple ordering can be established in terms of one learning algorithm's performance, for instance SARSA(λ), but of more interest would be the identification of general features that contribute to how easy or difficult a task is to learn. Part of this appears to involve both the number of aliased states, and how the aliased states are grouped. It may also be the case that the reward structure of the tasks influences learnability, for example, minimum-cost-to-goal tasks, as identified in section 10.2, may form a set of POMDP tasks for which it is relatively easy to learn satisficing solutions.

10.4 Curtailing Exploration

Results from chapters 4 and 5 suggest the hypothesis that exploration need to be completely curtailed before stable deterministic reactive policies can be reliably formed on

POMDP tasks.

There is however an alternative hypothesis, that the deterministic reactive policies learnt by reinforcement learning algorithms have adapted to the level of exploration which is present at that time and testing them under different conditions, for example, in the absence of any exploration, leads to the appearance of poor policies. Possible examples of this are provided by the policies learnt for high τ values when using Gibbs/Boltzmann action selection in chapter 5. We also observed similar results in preliminary work with the simulated B21-robot experiments, from chapter 8. In both cases SARSA(λ) appears to learn deterministic policies that rely on the occasional selection of random exploratory actions in order to reach the goal.

The experiments presented throughout this thesis test the policies that have been learnt by taking a snapshot of the current policy and then following it greedily, that is, without allowing any random exploratory actions. Under this measure it appears that the number of satisficing policies increases rapidly once exploration, during the learning phase, has virtually ceased. To test which of the above two hypotheses holds, the above tests could be repeated, but instead of following the policies greedily, their performance should be tested using the conditions that were prevalent at the time the snapshot was taken. The number of satisficing policies will of course increase under these test conditions, as any 'random walk' policy will eventually achieve the goal and thus be satisficing. As such, using the number of satisficing policies learnt as a measure will probably not provide a useful indication. Instead we should look at the average path length and the spread of path lengths to see if a significant number are performing better than a random walk could achieve. The change in the value of this measure over time, when compared to the same measure when policies are followed greedily, should give an indication if convergence to policies that exploit the existence of random exploratory actions is occurring.

In a further test, the level of exploration should be sustained at a constant level and the stability of individual policies should be monitored. This should indicate whether the policies learnt are stable in the presence of exploratory actions. If they are, it would indicate that they have reached some equilibrium point. It is also possible that

an algorithm alternates between policies around some equilibrium point. Long term tracking of Q-values while executing this latter test should provide an indication if this is the case.

It would also be useful to pursue a theoretical approach as suggested at the end of section 5.1.2.2. It would be of great interest to show formally (i) that stable *greedy* policies always exist for POMDPs or some sub-class of POMDPs, (ii) a proof of convergence to such greedy policies, including conditions required for convergence such as the learning rate schedule, (iii) the quality of such greedy policies, that is, for POMDP minimum-cost-to-goal tasks, will greedy action selection always converge to an reactive policy that is satisficing?

10.5 Separate and Shared Observation Spaces

As discussed in section 9.2.6 it is possible to construct active perception agents that either (i) map every perceptual action onto a single shared observation space, and thus one observation-action table, (ii) map each perceptual action onto its own observation space, and thus maintain a separate observation-action table for each, or (iii) some mixture of (i) and (ii). Shared observation spaces, and thus observation-action tables, when done appropriately will speed learning through generalisation. Separate observation spaces will reduce the number of many-to-one mappings between actions and observations and therefore reduce the amount of perceptual aliasing introduced by the perceptual actions. Which approach should be preferred, either in terms of learning better policies or speeding learning, is an area requiring further investigation.

10.6 Generalising Our Active Perception Approach

Our current implementation of active perception agents enforces an ordering over observation spaces, with one observation space being picked out as the default observa-

tion space, that is, the space where observations are drawn from following a physical action. In this design, single observations are drawn from alternative observation spaces when an agent executes a corresponding perceptual action. It is possible to conceive of alternate implementations where, for example, the perceptual actions switch between the observation spaces, and the agent continues to obtain observations from the most recently selected observation space until the next perceptual action is selected. This design would have interesting implications. Allowing the agent to maintain the same selected observation space over a series of physical actions gives the agent an internal memory of the most recently selected perceptual action. In this design, perceptual actions, as well as changing the agent's viewpoint, can be seen as acting equivalent to an action that sets a memory bit of an internal memory bit agent.

Another design possibility is an agent which executes a series of perceptual actions in order to obtain the best representation of the current space before selecting a physical action. This latter design would match the consistent representation (CR) approach advocated by Whitehead [1992]; Whitehead and Lin [1995]. This approach assumes that perceptual actions are intrinsically different from physical actions and as such there should exist a specialised mechanism for selecting these actions. In this thesis we maintain that there is not a clear distinction between a perceptual and physical action and as such there is no need for separate selection mechanisms. This view was adopted by McCallum [1995b], and is supported in part by the views of Allport [1989]. It is, however, possible to envisage perceptual actions that only select between different computational processes which are internal to an agent. Such perceptual actions could be safely segregated from physical actions in that they do not change the world state. The question then arises as to whether it would be worth adopting a CR approach for such actions. The Lion Algorithm [Whitehead, 1992] is a CR algorithm. The results that it achieves in chapter 8 very much leave this question open. The Lion Algorithm performs relatively poorly compared to CEQ(λ) or SARSA(λ) but this does not rule out the existence of a CR algorithm which can better exploit the perceptual actions available to the agent.

An area of further investigation would be to examine the performance of active perception agents structured according to either of the above designs.

There are alternative ways, other than active perception, in which an agent's abilities can be extended to help learn in POMDP worlds, for example, internal memory bits. The common link between these approaches is not that they attempt to render every state of the world Markovian, but that they increase the range of actions and observations available to an agent, potentially increasing the number of satisficing reactive policies that exist, and thus increasing the chances of a satisficing policy being learnt. Future work should aim at developing a generalised framework for such approaches.

10.7 Combining Active Perception with Other Approaches

It is not our intention to say that active perception should be the only solution to reinforcement learning of partially observable tasks. Where other approaches offer advantages there is no reason why they should not be combined. As an example a natural synergy exists between active perception and agents that have internal memory bits that they manipulate through action selection, that is, both are modifiable approaches. As indicated in chapter 8 and section 10.1 agents that use active perception and internal memory bit agents face a very similar problem. Both types of agent are still operating within partially observable worlds, and both are seeking Markovian paths through such worlds. This similarity is underlined by the improved performance of both types of agent when coupled with CEQ(λ) in chapter 8. Thus it is easy to imagine an agent that has both active perception and internal memory bits.

10.8 Required Aspects of RL Approach for POMDPs

Section 9.4 discusses the parallels between the first visit Monte Carlo algorithm and the temporal difference approach that we find to work most successfully in this thesis. Our results demonstrate that (i) adopting Consistent Exploration, and (ii) using λ close or equal to one, gives the best performance on the range of problems tried. There remain points of our approach whose impact upon learning have yet to be tested. They are:

(a) Truncation of eligibility traces when the agent's policy changes or an exploratory action is selected.

(b) The use of replacement rather then accumulating traces.

(c) Resetting of the agent's eligibility trace on reaching the goal location.

The behaviour in (a) is exhibited by CEQ(λ) by virtue of it being based on Watkins's Q(λ). It is possible to envisage a Consistent Exploration algorithm that is based upon SARSA(λ) which would not exhibit this behaviour. We present such an algorithm in appendix K. This algorithm, which we call CE-SARSA(λ), maintains a continuous eligibility trace irrespective of whether the next action selected is greedy or exploratory. Unfortunately a straight comparison between CEQ(λ) and CE-SARSA(λ) will not isolate performance differences which are solely due to (a). This is because we identify a anomaly in how exploration is handled in CEQ(λ) in section 7.2 and appendix I. To isolate the contribution of (a) towards the successful learning of reactive policies it would be better to compare the performance of the revised algorithm CEQ(λ)$^+$ (found in appendix J) and CE-SARSA(λ) .

Items (b) and (c) are relatively easy to assess, though each would require a substantial number of experiments to be run in order to arrive at a meaningful comparison. Item (b) requires a simple code change in the reinforcement learning algorithm such that eligibility trace values are accumulated rather than being reset each time an observation is re-observed, see Sutton and Barto [1998]. Note that work on comparing replacement and accumulating traces on MDPs was done by Singh and Sutton [1996] where more favourable results were reported for replacement traces. Item (c) requires a programmatic change in the code used to manage the agents and world such that resetting the eligibility trace on reaching the goal state is omitted.

10.9 CEQ(λ)

CEQ(λ) was not specially designed to perform well on grid worlds, and therefore we believe that its performance should generalise to other problems for which determin-

istic reactive policies exist. We demonstrate this for one non-grid world problem, the simulated B21 robot task in chapter 8. An area of future work would be to further establish the generality of these results on other tasks.

In section 7.2 we demonstrate some theoretical limitations for CEQ(λ). These are however very limited results and it would be useful to demonstrate some positive results indicating under what conditions convergence to satisficing deterministic policies can be guaranteed. Section 7.2.4 provides some pointers indicating how the theoretical analysis could be moved forward.

As a result of our theoretical analysis in section 7.2 and appendix I we uncovered what we considered to be an anomaly in the exploratory behaviour of CEQ(λ). This anomaly is addressed in the algorithm CEQ(λ)$^+$ which can be found in appendix J. It would be interesting to measure the difference in performance between CEQ(λ) and CEQ(λ)$^+$.

Finally, the concept of Consistent Exploration can be applied to other reinforcement learning algorithms, for example, a consistent exploration version of SARSA(λ), as detailed in appendix K, where unlike CEQ(λ) and CEQ(λ)$^+$ the eligibility trace is not reset when a new exploratory action is selected. It would be interesting to test how such alternative algorithms perform.

10.10 Further Comparisons

As mentioned in section 3.1.9 of the literature review, recent interest in reinforcement learning in non Markovian domains has focused on learning Predictive State Representations (PSRs) in order to aid state disambiguation. Such algorithms are the inheritors of U-Tree's ambition which is to build internal Markovian models of the world. Empirical comparisons between such approaches and the approach presented in this thesis would be very useful in order to establish the success of such predictive approaches.

Similarly, given the good results that have been claimed for recurrent neural networks using Long Short-Term Memory (LSTM), see section 3.1.5, it would be useful to compare our approach against this form of state estimation.

10.11 Implementation on a Real Robotic Platform

Ultimately, we would like to prove our approach using for example a mobile robot navigating a building, where the robot's learning algorithm has some direct control over the observations it obtains. Control could be expressed either in terms of selecting how the data is processed, for example, selecting the method used to categorise sonar data, or in overtly directing of sensors, for example, panning or zooming a camera to obtain different views.

10.11.1 Speed of learning

The main limitation in testing such an approach is the speed with which the policies are learnt. This is an issue which bedevils most reinforcement learning approaches. One promising approach is to externally direct the exploration of the robot by having it follow or imitate a teacher who knows the task [Marom, 2003]. The guidance provided by the teacher can the shape the agent's initial policy and help it to find the goal state. A lot of the initial exploration carried out by reinforcement learning algorithms involves the location of the goal and then the slow propagation of the reward back from the goal through the rest of the observation-action space. A second function of the teacher could be to manipulate the rewards that the agent receives in order to punish poor policies, see section 9.5.3.

Other approaches to speed learning could involve the learning agent better directing its exploration though either maintaining (i) counts of each observation, and preferring to sample those rarely encountered, or (ii) confidence intervals on observation-action values and re-sampling those observation-action pairs that have greatest uncertainty. To our knowledge there has been little work on applying either of these techniques when learning reactive policies on POMDPs, so their effectiveness in POMDPS is unknown.

10.11.2 Density of cues available

An issue we have not examined is the density of cues required in order to make learning good satisficing policies reliable. For example, in the B21 robot simulation, in chapter 8, observations that can be seen by selecting the active perception action are placed at the three aliased locations that we consider critical to the problem; the two T-junctions and the entrance to the room. If the number of cues were reduced to say only one, would this be sufficient to allow good satisficing policies to be learnt, and how reliably could such policies be learnt? Alternatively, if the number of cues available throughout the corridors were increased, would this result in confusing the learning algorithm? Answers to these questions would help a designer to determine the sensors to use and the preprocessing of sensory data that should be required.

10.11.3 Learning cues in an ever changing world

As active perception agents are reliant on external cues in the world to disambiguate states, they are vulnerable to changes in the world that affect those cues. As we discussed in section 9.2.2.1 any cue that changes with a time period much shorter than the time period taken by the agent to learn a satisficing policy will probably be disregarded in favour of cues that remain static over this period. It would be useful to demonstrate this conjecture empirically.

Other interesting questions in this area that are worth further study are:

- how detrimental is the process of ongoing change to learning of satisficing policies by active perception agents?
- what level of change can be tolerated and still allow satisficing policies to be learnt?
- is there a need for on going learning to allow for changes in the environment?

For minimum-cost-to-goal tasks it would appear to be relatively easy to achieve continuous learning as it would only require that some minimum positive level of learning rate is maintained. Given our discussion in section 10.2, to maintain learning for a minimum-cost-to-goal task, the learning algorithm would probably not have to continue to select random exploratory actions, as the rewards received whilst following non-satisficing policies should be sufficient to drive exploration.

An understanding of the above issues is of interest when attempting to implement active perception on a real robotic platform, as they would affect the robot's performance when learning in the real world.

10.12 Chapter Summary

Although this work provides some pointers, the field still lacks a clear formulation of what factors govern the learnability of satisficing reactive policies for given POMDP tasks. Useful future developments would include techniques which identify when satisficing observation-base policies exist for a given POMDP task, and the conditions under which such policies are learnable.

We have investigated one particular extension of an agent's abilities. There exist alternative designs of active perception, as well as other ways in which an agent's abilities can be extended. We consider the additional abilities given to agents as different from those given to learning algorithms, as the former do not attempt to render every state of the task Markovian. Instead they increase the range of actions and observations available to an agent, with the aim of making it easier to learn satisficing reactive policies. We hope that in drawing this distinction we will stimulate investigation into methods similar to active perception which do not rely on rendering every state of a task Markovian.

Ultimately we would like to see the development of a generalised framework for extending an agent's abilities which, in conjunction with work on learning reactive poli-

cies in POMDPs, provides a set of tools for engineers wishing to design agents which are able to operate in the everyday non-Markovian world in which we all live.

Chapter 11

Conclusions

Understanding how and why animals, or humans, exercise control over their perception of the complex world in which they live is a deeply fascinating issue. We do not pretend to have addressed this issue in any depth, but we hope the results that we found in giving artificial agents some control over their perception indicate ways in which it is useful to control perceptions, especially when reactions are tied into what is perceived.

11.1 Review of Thesis Aims

11.1.1 First hypothesis

The main hypothesis of this thesis is that in the field of reinforcement learning, the act of equipping an agent with active perception can prove an effective approach to learning in partially observable worlds. In chapter 1 we set out a list of supporting evidence which we aimed to show through this work. For convenience the original list is repeated again below, following which we present a summary of the evidence for each point. The evidence that we provide in support of the first hypothesis is:

(i) that the addition of active perception can allow an agent to learn satisficing, deterministic, reactive policies in many partially observable worlds, where no satisficing policies were possible before,

(ii) a demonstration that enhancing an agent with active perception will allow it to learn satisficing reactive policies that are better, according to some quality measure, for example, the number of physical steps taken to reach the goal, than the optimal reactive policy achievable by an non-enhanced agent,

(iii) the presentation of techniques that improve the reliability of learning satisficing, deterministic, reactive policies in partially observable worlds,

(iv) an empirical demonstration that the performance of active perception agents is comparable with other approaches in the field of reinforcement learning.

11.1.1.1 Deterministic reactive policies exist where none were possible before

Using McCallum's M-maze, we demonstrate in chapter 4 an example of a task where no satisficing, deterministic, reactive policies existed, but where it is possible to learn such policies when active perception is added.

This is an important result, but one whose occurrence is relatively straightforward to envisage. Tasks for which *no satisficing*, deterministic, reactive policies exist contain a minimum of two states that provide the same observation yet required different actions to be executed. In addition, there has to be no route which circumnavigates these two states, or the possibility, when executing the same action in both, of reaching the goal via an alternative route. The addition of active perception to an agent can make such tasks possible by changing how the agent perceives its world and expanding the number of observations that are possible for each state. This expansion in the set of observations available to the agent for each underlying state increases the likelihood that some deterministic path to the goal can be learnt. When active perception is added to the agent's abilities, the typically satisficing, deterministic policies that are brought into existence are ones where the same perceptual action is selected in the two states

that produce the same observation. As a result of selecting this perceptual action, new observations that differ for the two states are seen by the agent. Based on these new observations the agent can then select a different action to execute for each state and thus find a deterministic route through to the goal. Alternatively, other paths may now exist, which, as a result of the addition of active perception, can circumnavigate the two states that present the same observation.

No specific guarantee can be given as to what active perception needs to be added in order that satisficing, deterministic, reactive policies will exist for specific tasks. The abilities required by an agent can realistically[1] only be determined by a designer who has knowledge of the agent's sensors and the task. This situation is, however, no different from that which occurs when adding internal memory bits to an agent, which the agent itself manipulates, as in work by Lanzi [2000]. The internal memory bits are perceived by the agent as part of the observation it receives from the world. As for active perception above, this increases the number of ways each state can be represented, thus increasing the probability of the existence of satisficing, deterministic, policies. However, no method has been presented in the literature for guaranteeing that satisficing, reactive policies exist for particular internal memory bit agent designs. It is again down to the designer to determining the number of internal memory bits and the actions required to manipulate those bits. In designing the experiments for this thesis, our experience suggests that the job of the designer is not particularly difficult: in most cases we were able to suitably augment the agents with useful active perceptions based on a cursory examination of the tasks.

It is also worth noting that for agents with internal memory bits, the observation space as perceived by the learning algorithm doubles with *each* memory bit added. Which in turn doubles the observation-action space that the agent need to sample. As we demonstrate in section 9.2.5 the increase state space for active perception can be additive rather than multiplicative. As a result the increase in the size of the observation space might be smaller for active perception, which would theoretically result in quicker learning.

[1] The designer could be removed from the problem of designing agents through use of genetic algorithms, but this would probably be a significantly slower approach.

In arguing the above point we do not want to give the impression that there is a strict choice between active perception or internal memory agent designs. As both approaches fall into the same category, that of modifiable approaches, they share a lot of similarities. Thus one can envisage a hybrid agent which has both active perception and internal memory registers. The two techniques are complimentary; internal memory would be useful when external cues in the environment are sparse, and the active perception system can, by allowing different representations, reduce the amount of internal memory required and provide robustness to changes that are not captured by internal memory, such as unexpected relocation of the agent.

11.1.1.2 Improved policies

This result is demonstrated in chapter 4 using a modified form of Sutton's grid world as proposed by Littman [1994]. On this task, one agent enhanced with active perception is shown to learn reactive, deterministic policies that take fewer physical actions to reach the goal state than the number of physical steps in the optimal reactive, deterministic policy that is possible for an non-enhanced agent.

A possible criticism of these results is that for the majority of policies learnt, the total actions executed, both physical and perception, may not be less than the total physical actions for the optimal policy of an non-enhanced agent. However, in general the cost of perceptual actions, such as looking around or focusing on an object, will be less than the cost of physical movements. It would be possible to reflect these relative costs in the reinforcement learning penalties imposed for executing each action, and thus ensure that the agent balances these factors as part of its learning. In this thesis we do not use this approach, as differing rewards for actions in minimum-cost-to-goal[2] tasks, when coupled with discounting of future rewards, can create the situation where the cost of continuously executing the cheapest action appears more attractive than seeking routes to the goal. This is discussed in section 9.5.

[2] See section 10.2 for definition.

The improvement of policies by the addition of active perception is further demonstrated in the B21 robot simulation problem presented in chapter 8, where the policies learnt by a robot equipped with an additional active perception action are better than the policies learnt by a robot not equipped with these abilities.

11.1.1.3 Improvement in reliability of learning satisficing, deterministic, reactive policies

Chapters 5 and 6 examined recent theoretical results for the convergence of reinforcement learning policies on partially observable tasks.

The first of these chapters considered action selection. Although our results do not threaten the theoretical finding of Perkins and Pendrith [2002], that stationary points exist in the state-action value (Q-value) space, when a continuous function is used for action selection, it appears that such stationary points do not necessarily correspond to locally minimum or satisficing policies.

The chapter provides an indication that stable policies may exist provided all exploration has been curtailed, that is, the agent acts greedily with respect to its Q-values.

The second of these chapters compares the convergence of policies using Monte Carlo based algorithms with that using temporal difference algorithms. As a result of this comparison we set out the concept of Consistent Exploration which is used to modify an existing temporal difference algorithm to obtain better performance. The modified algorithm, CEQ(λ), is presented in chapter 7, wherein we present evidence of it outperforming SARSA(λ) in terms of the probability of convergence to satisficing policies, in nearly every case. Further results that support the preference of CEQ(λ) over SARSA(λ) for a much larger body of problems are presented in chapter 8.

Overall, CEQ(λ) combined with an exploration strategy that ultimately becomes greedy with respect to its Q-values, appears to provide increased reliability for convergence on satisficing, deterministic, reactive policies for partially observable worlds.

11.1.1.4 Performance comparable with other approaches

Chapter 8 presents results showing that the active perception approach proposed, combined with the new algorithm CEQ(λ), and an exploration strategy that ultimately becomes greedy with respect to its *Q*-values, performs at least as well as alternative techniques which we have carried out comparisons against, on all tasks that we tested.

11.1.2 Second hypothesis

Our second hypothesis is that there is no requirement to manage perceptual actions separately through the provision of specialised reinforcement learning algorithms. This hypothesis is supported by:

(i) the comparison of results obtained in chapter 8 using either SARSA(λ) or CEQ(λ) (neither of which distinguishes between perception and physical actions) against the Lion Algorithm [Whitehead and Ballard, 1991; Whitehead, 1992] which maintains a strict distinction. The results show that the Lion Algorithm performs poorly compared to the other two,

(ii) the general result, which can be observed in all of the experiments where active perception has been used, that active perception agents have learnt good, satisficing, reactive policies without needing to employ learning algorithms that distinguish between the two types of actions.

11.2 Concluding Remarks

Overall the work presented in this thesis stands in support of our two hypotheses presented above. It demonstrates that active perception is a useful approach when applying reinforcement learning to partially observable tasks. It is not our intention to say that active perception is the sole solution, or that it should stand alone. As we

emphasise in section 10.7, active perception should be seen as a technique to be considered alongside other approaches, and possibly combined with them, when designing systems which operate in partially observable environments.

Appendix A

Best Parameter Values Found

The table below summaries the best parameter values found for each learning algorithm on the various tasks tested in this thesis. The best parameters were judge primarily on the percentage of satisficing policies learnt with a secondary measure of minimum path length of the policies learnt. We cannot claim that these values are optimal as (i) in a few cases the range of values tested was quite limited, and (ii) statistically similar results can be observed over a range of parameter values. They do however offer a useful guide as to where in the parameter space good results can be found.

Table A.1: Best parameter values found

Algorithm	Task	Best Parameter Values
Q-learning	**Simple 1-D example**	$\alpha = 0.1, \gamma = 0.9$, ε-greedy/greedy[1]
	Sutton's Grid World using:	
	• Absolute Position Agent	
	• State Oracle Agent	
	• Action Oracle Agent	
	• Active Perception Agent	
	• Eight Adjacent Squares Agent	$\alpha = 0.5$, $\gamma = 0.9$, ε-greedy/greedy[1]
SARSA	**Simple 1-D example**	$\alpha = 0.1, \gamma = 0.9$, ε-greedy/greedy[1]
	Sutton's Grid World using:	
	• Absolute Position Agent	
	• State Oracle Agent	
	• Action Oracle Agent	
	• Active Perception Agent	
	• Eight Adjacent Squares Agent	$\alpha = 0.7$, $\gamma = 0.9$, ε-greedy/greedy[1]
CEQ(λ) with replacing eligibility traces	**Sutton's Grid World using:**	
	• Eight Adjacent Squares Agent	$\alpha = 0.1$, $\gamma = 0.9$, $\lambda = 0.99$, ε-greedy/greedy[1]
	• Active Perception Agent	$\alpha = 0.05, \gamma = 0.8, \lambda = 0.99$ ε-greedy/greedy[1]
	Random Grid Worlds using:	
	• Active Perception Agent	
	• 1-bit Internal Memory Agent	$\alpha = 0.05$, $\gamma = 0.8$, $\lambda = 1$, ε-greedy/greedy[1]
	• Eight Adjacent Squares Agent	$\alpha = 0.3$, $\gamma = 1$, $\lambda = 1$, ε-greedy/greedy[1]
	Wilson's Woods 7	$\alpha = 0.3$, $\gamma = 0.8$, $\lambda = 1$, ε-greedy/greedy[1]

[1] ε-greedy exploration with initial ε value of 0.2 decaying linearly to zero by midpoint of run, thereafter acting greedily.

Table A.1: Best parameter values found

Algorithm	Task	Best Parameter Values
SARSA(λ) with replacing eligibility traces	**Simple 1-D example**	$\alpha = 0.1, \gamma = 0.9, \lambda = 0.9$ ε-greedy/greedy[1]
	Sutton's Grid World using:	
	• Absolute Position Agent	
	• State Oracle Agent	$\alpha = 0.01, \gamma = 0.9, \lambda = 0.9$ ε-greedy/greedy[1]
	• Action Oracle Agent	
	• Eight Adjacent Squares Agent	$\alpha = 0.05, \gamma = 0.9, \lambda = 0.99$ ε-greedy/greedy[1] or softmax[2] $\tau = [0.03, 0.05]$
	• Active Perception Agent	
	Sutton's Grid World B	as above except $\tau = [0.01, 0.03]$
	Random Grid Worlds using:	
	• Active Perception Agent	$\alpha = 0.05$, $\gamma = 0.99$, $\lambda = 0.8$, ε-greedy/greedy[1]
	• 1-bit Internal Memory Agent	$\alpha = 0.02$, $\gamma = 0.9$, $\lambda = 0.8$, ε-greedy/greedy[1]
	• Eight Adjacent Squares Agent	$\alpha = 0.1$, $\gamma = 1$, $\lambda = 0.99$, ε-greedy/greedy[1]
	Wilson's Woods 7	$\alpha = 0.1$, $\gamma = 0.8$, $\lambda = 0.8$, ε-greedy/greedy[1]
Watkins's Q(λ) with accumulating eligibility traces	**Simple 1-D example**	$\alpha = 0.1, \gamma = 0.9, \lambda = 0.9$ ε-greedy/greedy[1]
	Sutton's Grid World using:	
	• Absolute Position Agent	
	• State Oracle Agent	$\alpha = 0.01, \gamma = 0.9, \lambda = 0.9$ ε-greedy/greedy[1]
	• Action Oracle Agent	
	• Eight Adjacent Squares Agent	

[2]Softmax or Gibbs/Boltzmann action selection as given in equation 5.23 on page 164.

Table A.1: Best parameter values found

Algorithm	Task	Best Parameter Values
1st visit Monte Carlo with exploring starts	**Sutton's Grid World using:** • Eight Adjacent Squares Agent	$\gamma = 1$ constant level of exploration; *i.e.* initial state and action always selected at random
MCESP-CD	**Sutton's Grid World using:** • Eight Adjacent Squares Agent	$\alpha = 0.1$, $\gamma = 0.9$, $\delta = 0.001$
Lion Algorithm	**Random Grid Worlds using:** • Active Perception Agent	$\alpha = 0.01$, $\gamma = 0.99$, $\alpha' = 0.001$, ε-greedy/greedy[1], 4 perceptual actions per step
U-Tree	**Random Grid Worlds using:** • Eight Adjacent Squares Agent	$\gamma = 0.99$, $p = 0.001$, ε-greedy/greedy[1], max. fringe depth = 3, max. history index = 4, min. 20 instances for K-S test 1000 steps between tests

Appendix B

Q-learning

Algorithm 5 Q-learning: A 1-step backup, TD, off-policy control algorithm [Sutton and Barto, 1998, p.149]

Initialise $Q(s,a) \leftarrow$ arbitrary values

Repeat (for each episode):

Initialise s

Repeat (for each step of episode):

choose a from s using policy derived from Q (for example, ε-greedy)

take action a, obtain r and observe s'

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

$s \leftarrow s'$

until s is terminal

Appendix C

SARSA

Algorithm 6 Sarsa: An on-policy TD control algorithm [Sutton and Barto, 1998, p.146]

Initialise $Q(s,a) \leftarrow$ arbitrary values

Repeat (for each episode):

Initialise s

Choose a from s using policy derived from Q (for example, ε-greedy)

Repeat (for each step of episode):

take action a, observer, s'

Choose a' from s' using policy derived from Q (for example, ε-greedy)

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

$s \leftarrow s'$; $a \leftarrow a'$

until s is terminal

Appendix D

SARSA(λ)

Algorithm 7 Tabular Sarsa(λ) [Sutton and Barto, 1998, p.181] using replacing eligibility trace. Note algorithm corrected in line with errata [Sutton and Barto, 2006] such that eligibility traces are reset at the start of each episode

Initialise $Q(s,a) \leftarrow$ arbitrary values

Repeat (for each episode):

Reset $e(s,a) = 0$ for all s, a

Initialise s, a

Repeat (for each step of episode):

take action a, observe r, s'

Choose a' from s' using policy derived from Q (for example, ε-greedy)

$\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$

$e(s,a) \leftarrow 1$

For all s, a

$Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$

$e(s,a) \leftarrow \gamma\lambda e(s,a)$

$s \leftarrow s'; a \leftarrow a'$

until s is terminal

Appendix E

Watkins's Q(λ)

Algorithm 8 Tabular version of Watkins's Q(λ) [Sutton and Barto, 1998, p.184] using accumulating eligibility trace. Note algorithm corrected in line with errata [Sutton and Barto, 2006] such that eligibility traces are reset at the start of each episode

Initialise $Q(s,a) \leftarrow$ arbitrary values

Repeat (for each episode):

- Reset $e(s,a) = 0$ for all s, a
- Initialise s, a
- Repeat (for each step of episode):
  - take action a, obtain r and observe s'
  - choose a' from s' using policy derived from Q (for example, ε-greedy)
  - $a^* \leftarrow \arg\max_b Q(s',b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
  - $\delta = r + \gamma Q(s',a^*) - Q(s,a)$
  - $e(s,a) \leftarrow e(s,a) + 1$
  - for all s, a:
    - $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
    - if $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
      else $e(s,a) \leftarrow 0$
  - $s \leftarrow s'$; $a \leftarrow a'$
- until s is terminal

Appendix F

Modified Watkins's Q(λ)

F.1 Eligibility Traces

In reinforcement learning, the application of eligibility traces to one-step temporal difference (TD) methods creates a broader range of algorithms. This range spans from one-step TD methods at one end of the spectrum to Monte Carlo like methods at the other [Sutton and Barto, 1998, chp.7]. An eligibility trace controlling how many state-action values (Q values) which preceded the current state are updated in anyone time step. At one extreme, one-step TD algorithms only update one Q value, that of the most recent state-action pair to occur, at the other extreme Monte Carlo like methods update complete episodes of Q values, that is, the most recent reward is backed up over all the state-actions pairs encountered since the start of the episode. The number of states which are backed up at any one step is typically controlled by a parameter λ with λ values close to 1 resulting in Monte Carlo like methods, and $\lambda = 0$ resulting in one-step TD algorithms.

SARSA [Sutton and Barto, 1998, p.146] is a one-step TD learning algorithm which is classed as an *on-policy* algorithm. On-policy implying that the Q values learnt by SARSA depend on the policy being followed. The concept of eligibility traces can be applied to SARSA resulting in the algorithm of SARSA(λ) [Sutton and Barto, 1998,

p.181]. Both SARSA and SARSA(λ) are on-policy and the Q values learnt by both will ultimately converge to identical values provided they are following the same policy. SARSA and SARSA(λ) also have the useful property[1] of being algorithmically equivalent when $\lambda = 0$.

Q-learning [Sutton and Barto, 1998, p.149] is a *off-policy* one-step TD algorithm. It is off-policy as the Q values learnt converge to the same values irrespective of the policy that the algorithm is currently following. Assuming the problem to be Markovian the Q values will converge to those that correspond to the optimal policy. Extending Q-learning using the concept of eligibility traces results in algorithms such as Watkins's Q(λ) [Sutton and Barto, 1998, p.184]. Like Q-learning, Watkins's Q(λ) is off-policy and the Q values it learns will ultimately be identical to the values converged on by Q-learning. However, unlike SARSA and SARSA(λ), the tabular form of Watkins's Q(λ) as presented in [Sutton and Barto, 1998] and reproduced in Algorithm 9 is not algorithmically equivalent to Q-learning for $\lambda = 0$. The algorithmic difference, which we detail in section F.2, can cause a noticeable difference in the average convergence of policies between Q-learning and Watkins's Q(λ) for $\lambda = 0$.

It is possible, through a simple modification of the Watkins's Q(λ) algorithm, to achieve an algorithm that is equivalent to Q-learning for $\lambda = 0$.

F.2 Watkins's Q(λ) and Q-learning

The algorithmic difference between Watkins's Q(λ) when $\lambda = 0$ and Q-learning occurs through the relative timing between the updating of Q values and the selection of the next action to execute. The timing of these two events is significant as the policy used to select the next action is normally based upon the current Q values, for example, through the use of ε-greedy [Sutton and Barto, 1998, p.28] or Gibbs/Boltzmann distribution (softmax) [Sutton and Barto, 1998, p,30].

[1] from the point of view of reducing the number of different algorithms that need to be implemented

In the tabular form of Watkins's Q(λ), Algorithm 9, a single loop over all state-action combinations is used to update both Q values and the eligibility trace. This loop is highlighted in bold in Algorithm 9. In order to ensure Watkins's Q(λ) backs-up only Q values corresponding to the optimum policy, the eligibility trace has to be truncated when the agent selects an action (a') that does not follow the current optimal policy (a^*). This requirement requires that updating of the eligibility trace can only occur once the next action which the algorithm will execute is chosen, that is, it is necessary to know if the next action will follow the current optimal policy or not. This requirement fixes the position of the loop that updates the eligibility trace. Minimising the number of loops in the algorithm by using the same loop to update Q values, therefore forces the Q value update to occur after the next action to be executed has been selected.

When $\lambda = 0$ the eligibility trace is no longer significant but the construction of the Watkins's Q(λ) algorithm forces the selection of the next action before the Q values have been updated. In contrast, in the Q-learning algorithm (Algorithm 5, in section B) an action is chosen, executed, and an update of Q values occurs before the algorithm loops round to choose the next action. Thus the next action to be executed is always chosen after the knowledge of the effect of the previous action has been used to update the Q values.

Based on this algorithmic difference one would expect that during learning Q-learning would generally select better greedy actions than Watkins's Q(λ) with $\lambda = 0$. Q-learning having taken into account more recent information when making its action selection than that used by Watkins's Q(λ).

F.3 Modified Form of Watkins's Q(λ)

The modification required to achieve algorithmic equivalent between the tabular form of Watkins's Q(λ) for $\lambda = 0$ and Q-learning is very straight forward. As described in section F.2 the difference arises due to the timing of Q value updates and the choice

Algorithm 9 Tabular version of Watkins's Q(λ) [Sutton and Barto, 1998, p.184] using replacing eligibility trace[Sutton and Barto, 1998, p.188]. The loop over state-action values that updates both Q values and eligibility trace is highlighted using bold

Initialise $Q(s,a) \leftarrow$ arbitrary values
Repeat (for each episode):
  Reset $e(s,a) = 0$ for all s, a
  Initialise s, a
  Repeat (for each step of episode):
    take action a, obtain r and observe s'
    choose a' from s' using policy derived from Q (for example, ε-greedy)
    $a^* \leftarrow \arg\max_b Q(s',b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
    $\delta = r + \gamma Q(s',a^*) - Q(s,a)$
    $e(s,a) \leftarrow 1$
    for all s, a:
      $\mathbf{Q(s,a)} \leftarrow \mathbf{Q(s,a)} + \alpha\delta\mathbf{e(s,a)}$
      if $\mathbf{a' = a^*}$**, then** $\mathbf{e(s,a)} \leftarrow \gamma\lambda\mathbf{e(s,a)}$
        else $\mathbf{e(s,a)} \leftarrow \mathbf{0}$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

of the next action. The Q value updates occur after the choice of the next action in Watkins's Q(λ) solely due to minimising the number of loops over the state-action space. If the function of updating Q values is separated from that of updating the eligibility trace, then there is no reason why the selection of the next action to execute cannot be delayed until after the Q values have been updated.

A modified form of Watkins's Q(λ) which implement these simple changes is presented in Algorithm 10. If you consider this modified algorithm when $\lambda = 0$ it is possible to see that it is now equivalent to Q-learning (Algorithm 5) with the update of Q values occurring before the next action is chosen. It is also possible to see that this modification does not effect the algorithm's aim which is to learn, irrespective of the policy being followed, the Q values corresponding to the optimal policy.

Algorithm 10 Modified form of Tabular Watkins's Q(λ) using replacing eligibility traces. Our modification of the algorithm which updates Q value before the next action is chosen and then updated eligibity traces is highlighted in bold

Initialise $Q(s,a) \leftarrow$ arbitrary values
Repeat (for each episode):
  Reset $e(s,a) = 0$ for all s, a
  Initialise s, a
  Repeat (for each step of episode):
    take action a, obtain r and observe s'
    $a^* \leftarrow \arg\max_b Q(s',b)$
    $\delta = r + \gamma Q(s',a^*) - Q(s,a)$
    $e(s,a) \leftarrow 1$
    for all s, a:
      $\mathbf{Q}(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s},\mathbf{a}) + \alpha\delta\mathbf{e}(\mathbf{s},\mathbf{a})$
    choose $\mathbf{a}'$ from $\mathbf{s}'$ using policy derived from updated Q (for example, ε-greedy)
    for all s, a:
      if $\mathbf{a}' = \mathbf{a}^*$, then $\mathbf{e}(\mathbf{s},\mathbf{a}) \leftarrow \gamma\lambda\mathbf{e}(\mathbf{s},\mathbf{a})$
      else $\mathbf{e}(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{0}$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

In a set of unpublished experiments we compared the performance of this modified algorithm with the original formulation of Watkins's Q(λ). Although the modification did result in improved mean path lengths and/or improved probability of convergence to satisficing policies on some POMDP tasks, the results were not uniform, with the unmodified version of the algorithm performing better on other tasks.

Appendix G

Supplemental Notes on U-Tree Implementation

This appendix is not intended as complete description of the U-Tree algorithm as details can be found in McCallum [1995b, pp87–94] and McCallum [1996] which are readily available from McCallum's web pages, see
`http://www.cs.umass.edu/~mccallum/publications-by-date.html`.

We intend this appendix as a supplement to the above two documents expanding points that we found to be unclear and detailing others which we considered under specified. In the course of implementing U-Tree we obtained code written by McCallum using Objective-C [McCallum, 2004]. This code is readily available from McCallum's web site, at `http://www.cs.umass.edu/~mccallum/code-data.html`, by follow the RLKIT link. Although we did not directly run this code, our examination of it guided our implementation. We also note in this appendix where our implementation significantly varies from this version of McCallum's code.

G.1 Algorithm Outline

U-Tree is an "instance-based" algorithm in that it maintains a complete history of instances that the agent has experienced. Each instance T_t stores a record of the previous instances that preceded it, T_{t-1}, the action taken a_{t-1}, and the resulting observation o_t and reward r_t obtained, that is, $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$. These instances are naturally stored as a time ordered chain, see figure G.1.

The U-Tree algorithm builds a second structure, a suffix tree (also know as a decison tree [Russell and Norvig, 2003]), which it uses to cluster the instances. Each instance is assigned to one of the leaves of this tree according to observation features and actions which preceded it, see figure G.1. The leaves of this tree form the internal states of the reinforcement learning agent and utility estimates (Q-values) are associated with each leaf. The agent determines its current "state" by starting at the root of this tree and falling down the tree based upon the history of instances it has observed until it reaches a leaf. It uses the utility estimates (Q-values) stored in the leaf to select its next action.

The U-Tree algorithm attempts to grows the tree in such a way as to divide the sensor space of the agent into task-relevant chunks. Deep parts of the tree correspond to finely distinguished parts of the space where many details and memories are important. Shallow parts of the tree correspond to broad regions of the input space where little detail is required in order to select the best action. Different regions of sensory space are divided up based on the utility that they afford in completing the current task.

The key feature of the U-Tree algorithm is that both feature selection, that is, selection of parts of the current sensory information, and short term memory selection are represented uniformly in the same tree structure.

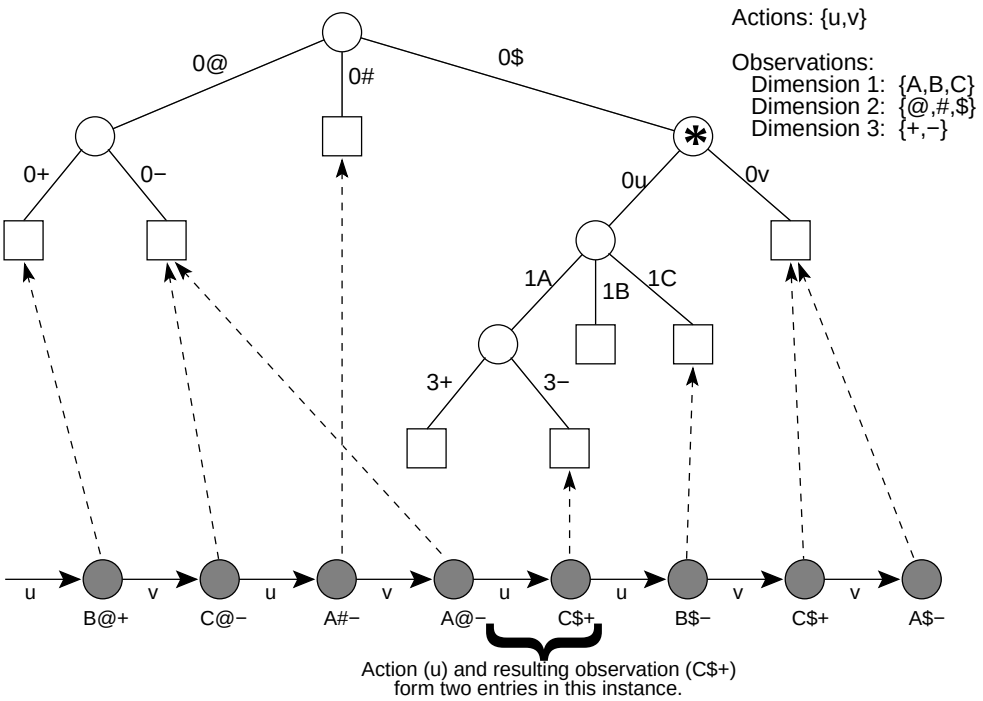


Figure G.1: Example of U-Tree instance-chain and tree. The instance chain is shown at the bottom of the figure, with each instance represented by a grey circle and the chain of events that preceded it. Every circle in the instance chain is labelled by the observation received at that time step, and every arrow is labelled with the action taken to make that transition. Each branch of the tree, top part of the figure, is labelled by a history index and perceptual dimension. History indexes enumerated backwards through the instance chain with 0 being the current instance, 1 being one instance ago, etc.. The perceptual dimension is either a sub-division of the sensory space or the action executed. Tree leaves, drawn as squares, are the agent's internal states. The dashed-arrows indicate the agent's "state" given that it just has experienced the instance where the dashed arrow originates. The dash-arrows do not indicate the leaf where the instance is stored, this is found by considering the previous instance. The diagram is modified from the one presented in McCallum [1995b] and McCallum [1996] to be consistent with our understanding of the algorithm. We believe the actions preceding each greyed circle forms part of that instance's definition, not the actions leaving each greyed circle, that is, the observation o_t is that received after taking action a_{t-1} and an instance is defined as $\langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$. We interpret this to mean that the actions preceding each greyed circle is the action at history 0, not history 1, thus we change the distinction test used at the inner tree node marked with an asterisk compared to the figure presented in McCallum [1995b] and McCallum [1996]. The test at this node is changed from testing the action at history 1 to test the action at history 0

G.2 Points of Clarification

G.2.1 Storing instances and determining internal state

McCallum [1995b, pp87–94] and McCallum [1996] appear to provide contradictory details about where instances are stored in the tree. Concentrating on McCallum [1996] the details in section 2, which introduces U-Tree, in conjunction with figure 1 (similar to the modified figure presented here – figure G.1), suggests that that the leaf which represents the agent's internal state after observing instance, T, is also the leaf where instance T is stored. In contradiction to this section 3, which details the algorithm, states that "An instance, T, is deposited in the leaf node whose conjunction, s, is satisfied by the actions and observation of the transition instances that *precede* T in time" (emphasis added by author).

Our opinion is that the latter statement from section 3 of McCallum [1996] is correct. This is supported by comments in McCallum's code that "the transitions in a tree node are the possible *outgoing* transitions. From the node's point of view, for a [transition instance] in the node, [the transition instance's perception] is the perception at time (t+1), that is after [the transition instance's action] has been executed" [McCallum, 2004, file UTNodePA.]. That is to say that the instance $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$ is stored in the leaf found by starting at the root of the tree and falling down the branches based on the observations and actions contain in T_{t-1}. This makes sense as the leaf found usingT_{t-1} represents the state from which action a_{t-1} was executed, and following which, observation o_t and reward r_t were obtained. Starting from the root of the tree and falling down the branches based on T_t, that is, using a_{t-1}, o_t, r_t, takes you to the leaf that represents the agent's new state having executed a_{t-1} and observed o_t.

Storing instances in this fashion makes it relatively easy to calculate the utility of each action for a given state, s, of the agent. Both the estimated immediate reward for executing an action, a, from state s and the estimated probability that the agent arrives in a state s' given that it executed action a from s can be calculated from the instances

stored in leaf node s. This point is stated to be the case in section 3 of McCallum [1996].

In implementing U-Tree we therefore require two separate functions that answer two distinctly different questions: (i) Given instance T_t in which leaf do I store it? (ii) Given instance T_t what do I believe my current state to be? The answer to (i) is given by falling down the branches of the tree based on T_{t-1} and ignoring a_{t-1}, o_t, r_t. The answer to question (ii) is given by falling down the branches of the tree based on T_t and utilising the extra information a_{t-1}, o_t, r_t.

G.2.2 Extending the tree

After every k steps the agent tests whether newly added instances or dynamic programming updates of the leaves' Q-values has changed the utility statistics sufficiently to warrant adding new distinctions to the agent's internal state space. For each leaf in the existing tree the algorithms creates a fringe of possible extensions. Then uses the Kolmogorov-Smirnov test to compare distributions of utility values associated with the same action from different nodes of this fringe. If the test indicates that two distributions have a statically significant difference, then promoting the relevant fringe nodes to proper leaves of the tree will improve the distinction made between states, and thus help the agent predict reward.

G.2.2.1 Testing fringe leaves against parent leaf

As per the suggestion in McCallum [1995b, pp87–94] and McCallum [1996], rather than compare the distribution of all the leaves of a possible fringe expansion against each other, a process requiring n^2 comparisons (where n is the number of fringe leaves), our code compares the distribution of instances in each fringe leaf against the proper leaf node which the fringe would replace. We note however that the version of McCallum's code that we examined [McCallum, 2004] appears to compare distributions between each fringe nodes, though partial code exists for comparison against the parent leaf node.

G.2.2.2 Selection between alternative fringes

For an individual leaf there will be many possible fringe expansions and there exist the possibility that several alternative fringes will have leaves that pass the Kolmogorov-Smirnov test. That is to say there may be more than one way of dividing up the instances such that a statistically significant distribution in their utility values is detected. McCallum [1995b, pp87–94] and McCallum [1996] doesn't specify what should happen in such a circumstance. We adopt a policy of preferring fringes with the smallest depth, where depth is the number of layers of decisions nodes from the leaf that the fringe replaces to the proposed fringe's leaves. This is akin to Occam's razor in preferring the smallest addition to the existing tree which leads to a better division of the data. As a secondary selection criteria, if we find multiple fringes for an existing leaf, each of which contain a fringe leaf that passes the Kolmogorov-Smirnov test, and both these fringes share the same smallest depth, then we prefer the fringe which contains the fringe leaf which produced the highest Kolmogorov-Smirnov D statistic. The D statistic reflecting the distance between the sample of instances at the parent leaf and the sample of instances at the fringe leaf. This criteria for selection we believe matches that used in McCallum's code [McCallum, 2004].

Note that no specific ordering is specified for the order in which the fringe test is applied to the leaves of the tree. Nor does our code try extend all leaves before adding fringes to the tree. Leaves are tested in the order that naturally comes to hand, this can be visualised a working from left to right over the leaves of the tree in figure G.1. All fringe expansion are tested using the above criteria for one leaf at a time. If a distinguishing fringe expansion is found for a given leaf then that fringe is added, and then the splitting process restarts at the left most leaf of the newly extended tree. Because of this the fringe test may be applied to some parts of the tree more than others, thus biasing the trees that are learnt. It may be better to try all fringe expansion over all existing leaves and then select the best extension found across the whole tree. Neither our code, nor, as far as we can tell, McCallum's code [McCallum, 2004] does this.

G.2.2.3 Testing policy actions

Both McCallum [1995b, pp87–94] and McCallum [1996] contain the rather cryptic statement that "The agent only tests the distributions of the [parent] leaf node's policy action and the fringe node's policy action." Through examination of McCallum's code [McCallum, 2004] we reached the following interpretation:

- In carrying out Kolmogorov-Smirnov test to compare the instances in held in the parent leaf and the fringe leaf we consider only those instances that have executed a particular action. That action being the action that is the policy action in the parent leaf. Where we define the policy action of a leaf, L, as being the greedy action, $\arg\max_a Q(L,a)$.
- In the case where the policy action of the fringe leaf does not match the policy action of the parent leaf then a second Kolmogorov-Smirnov test is carried out for the fringe node's policy action. The policy action of a fringe leaf, L_f, being defined as the greedy action, $\arg\max_a Q(L_f,a)$.
- In the case where multiple actions share he same Q-value for either the parent leaf, the fringe leaf, or both, then the distribution of instances for all such actions are tested in turn.

This fringe will be considered, along with others, for adding to the tree in the event that any one of these Kolmogorov-Smirnov tests indicate it would be useful to do so.

G.2.2.4 Minimum number of instances to test

The Kolmogorov-Smirnov test is only applied when a minmum number of instances (all sharing the same action) are stored in the fringe leaf. By implication at least this number of instances (sharing this action) will be stored in the parent leaf of the fringe. McCallum's code [McCallum, 2004] only applies the Kolmogorov-Smirnov if a minimum of 10 instances are stored in the fringe leaf. We use a minimum requirement of 20 instances.

G.2.2.5 Restarting split test if fringe added

If a fringe is added to the tree then "the agent starts the split test again to look for further possible splits that the new distinction may bring to light" McCallum [1995b, 1996]. No specification is given about the management of leaf Q-values when a fringe becomes part of the tree proper. Ensuring that the Q-values remain approximately valid is especially important because of the fact that testing is restarted to look for further distinctions to add.

We take the approach of calculating the utility value of an instance, T_i, as the reward it received, r_i, plus the utility of the leaf node that the instance transients to, that is, $Q(T_i) = r_i + \gamma U(L(T_{i+1}))$ where $L(T_{i+1})$ is the leaf which T_i transients to, and where the subsequent instance T_{i+1} is deposited, and U is $U = \max_a Q(L(T_{i+1}), a)$. Before carrying out a fringe expansion on a leaf we calculate the utility value of every instance in that leaf using the *current* tree. If a fringe then replaces the leaf we calculate the utility value of each new leaf using the Q-values of the instances stored in that leaf. When the split test is restarted the instance utility values for each leaf are then recalculated using the new tree. We believe that using this approach the Q-values of leafs and instances should remain reasonable accurate provided the number of fringes added to the tree each time is small and no circular references of one new fringe to another new fringe occur. Based on observing the typical performance of the U-Tree algorithm we feel these are reasonable assumptions.

An alternative, though more computationally expensive approach is that adopted by McCallum [2004]. Following the addition of a fringe, this code runs several repetitions of Dynamic Programming until some Bellman residual tolerance is met.

G.2.2.6 Initiating Q-values

All state-action values, $Q(L, a)$ are, in our implementaion, initiated to zero.

G.2.3 History index exceeds number of instances

McCallum [1995b] and McCallum [1996] do not specify what to do if the history index of a decision node on the tree is greater than the number of instances in the instance-chain. That is to say, when falling down the branches of the tree, a decision point is reached that requests testing an instance M steps ago but no such instance exists as the history is less than M steps long. McCallum's code [McCallum, 2004] avoids this problem by not including the first M instances of the instance-chain in the tree. Where M is equal to the maximum history depth; that is the maximum depth that the U-Tree algorithm looks back when forming possible fringes. Thus when classify instances that are included in the tree, thus ensuring it is not possible to index to an instance beyond the start of the instance-chain.

Our implementation takes an alternative approach. We including all instances in the tree, then should the history index request the parameter value of an instance which does not exist, we return a default value of 'Unknown' for the requested parameter. There is probably little to choose between the two approaches although ours avoids wasting the information contained in the first M instances, where M can be up to 40 steps into the past.

G.2.4 Episode boundaries

How to handle episode boundaries is not an issue specific to U-Tree but occurs for all instance based methods. McCallum presents U-Tree only in the context of a continuous task which has no episode breaks, McCallum [1995b] and McCallum [1996], so no advice is offered in these two papers. Two issue arise: (i) Should the instance-chain be reset for each episode? (ii) If the instance-chain is not reset should episode boundaries be marked?

The tasks considered in this thesis are mainly goal seeking tasks where the agent is immediate relocated randomly in the world upon reaching the goal state. The U-Tree

algorithm tests every k steps to see if expanding the tree will provide better discrimination of the task. As the values of k is typically between 100 and 1,000 steps, which could easily exceed the length of a single episode, we retain all instances in an single instance-chain and do not disregard them at the end of each episode.

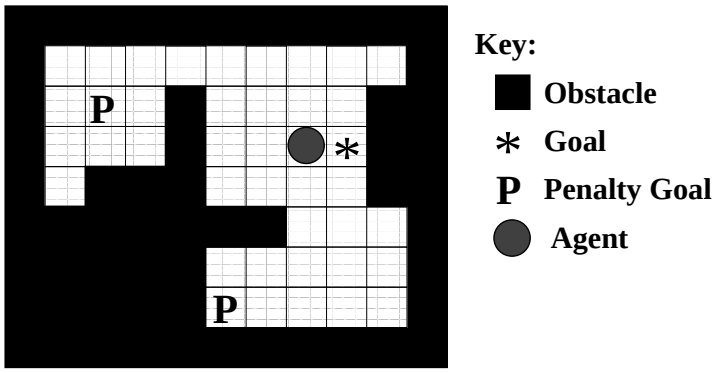


Figure G.2: Random grid world and agent

Having chosen to retain all instances we introduce an artificial instance to mark episode boundaries. The aim of this is (i) to ensure that the U-Tree learning algorithm obtains consistent observations for the goal state, (ii) it receives information which could allow it to detect episode restarts. An example best illustrates this approach. Consider the agent and world shown in figure G.2. The agent, shown as a greyed circle, is adjacent to the goal state marked with an asterisk. Selecting the action 'Move East' moves it to the goal at which point it receives a positive reward, say +10. The agent is then immediately relocated to a random location anywhere on the grid. If we do not introduce an artificial instance to mark the episode end, the agent will obtain the following instance $T_t = \langle T_{t-1}, MoveEast, +10, o_{randomlocation} \rangle$ where $o_{randomlocation}$ is the observation obtained from a random new start location. If, as we have done, an artificial instance is generated when the agent is relocated, then the instances obtained by the U-Tree learning algorithm are $\langle T_{t-1}, MoveEast, +10, o_{goal} \rangle$ followed by $\langle T_t, WorldReset, 0, o_{randomlocation} \rangle$. As can be observed in the latter case the agent obtains a consistent observation on reaching the goal state, o_{goal}. The start of the next episode is then flagged by an artificial instance containing the artificial action 'World Reset', for which the agent receives no reward and observes the new random starting location.

Appendix H

Implementation of Lion Algorithm

The Lion algorithm used in this thesis is that given by Whitehead and Ballard [1991] and shown below in algorithm 11. An outline of how the algorithm operates can be found in section 3.1.1 or in more detail in either Whitehead and Ballard [1991] or Whitehead [1992]. We make one change to the algorithm as originally presented. This is the introduction of a "suppression value" Q_{sup},which is used to suppress Q-values when they are believed to be inconsistent. This is in place of simply using a default value of zero. This change allows the algorithm to learn with negative reinforcement rewards as described in section 8.1.1.2. The modified lines of the algorithm are shown in bold font.

Algorithm 11 'Lion' Algorithm

initiate $\mathbf{Q_{sup}} = \mathbf{0}$.

Overt Cycle

1. Execute Perceptual Cycle and generate $\mathbf{S_t}$, a set of internal representations for the current world state.
2. Estimate the utility of the current world state, s_t: $V_E(s_t) \leftarrow \max_{s \in \mathbf{S_t}}[V_I(s)]$.
   Where V_E is the estimated state value of the world and V_I is the internal estimated state value.
3. Execute Update-Overt-Q-Estimates based on $V_E(s_t)$, r_t, $oact_{t-1}$ and $lion_{t-1}$; where r_t is the reward received at time t, $oact_{t-1}$ is the last overt action executed, and $lion_{t-1}$ is the internal state selected to represent the previous world state (see below).
4. Choose the next overt action to execute:
   With probability p follow policy:
   $oact \leftarrow a_\pi$ such that $\exists_{s_\pi \in \mathbf{S_t}}[Q_I(s_\pi, a_\pi) = \max_{(s,a) \in \mathbf{S_t} \times \mathbf{A_O}}[Q_I(s,a)]]$
   Otherwise choose randomly: $oact \leftarrow R(\mathbf{A_O})$, where Q_I is the internal estimated state-action value, $\mathbf{A_O}$ is the set of agent's overt actions, and R indicates a random choice.
5. Select the *Lion* : $Lion \leftarrow (s_L, oact)$ such that $Q_I(s_L, oact) = \max_{s \in \mathbf{S_t}}[Q_I(s, oact)]$.
6. Execute $oact$ to obtain a new world state and go to 1.

Update-Overt-Q-Estimates

1. Estimate the error in the lion's action-value: $error \leftarrow (r_t + \gamma V_E(s_t)) - Q_I(Lion_{t-1})$, where γ is the discount rate of future reward.
2. Update the action-value of the *Lion*:
   If $(\mathbf{error} < \mathbf{0})$ the lion is suspected of being inconsistent, so suppress it: $\mathbf{Q_I(Lion_{t-1})} \leftarrow \mathbf{Q_{sup}}$
   Else update it using the standard 1-step Q-learning rule:
   $Q_I(Lion_{t-1}) \leftarrow Q_I(Lion_{t-1}) + \alpha\, error$, where α is the learning rate.
   If $\mathbf{Q_I(Lion_{t-1})} < \mathbf{Q_{sup}}$ then $\mathbf{Q_{sup}} = \mathbf{Q_I(Lion_{t-1})}$.
3. Update non-lion internal states:
   For each $s \in \mathbf{S_{t-1}}$ and $s \neq state(Lion_{t-1})$ do:
   $Q_I(s, oact_{t-1}) \leftarrow Q_I(s, oact_{t-1}) + \alpha'\, error$, where α' is a learning rate smaller than α.
   If $\mathbf{Q_I(s, oact_{t-1})} < \mathbf{Q_{sup}}$ then $\mathbf{Q_{sup}} = \mathbf{Q_I(s, oact_{t-1})}$.

Perceptual Cycle

1. Initialise $\mathbf{S_t}$: $\mathbf{S_t} \leftarrow \{s_c\}$, where s_c is the current internal state.
2. Do n times (where n is a number fixed by hand):
   (a) select the next perceptual action ($pact$):
   with probability p' : $pact \leftarrow a$ such that $Q_I(s_c, a) = \max_{b \in \mathbf{A_p}}[Q_I(s_c, b)]$,
   otherwise: $pact \leftarrow R(\mathbf{A_p})$,
   where Q_I is the internal estimate value of the state-action pair, $\mathbf{A_p}$ is the set of possible perceptual actions, and R indicates a random choice.
   (b) execute $pact$ to obtain new internal state s'.
   (c) update the action-value estimate for the decision $(s_c, pact)$:
   $Q_I(s_c, pact) \leftarrow Q_I(s_c, pact) + \alpha(V_I(s') - Q_I(s_c, pact))$.
   (d) add s' to $\mathbf{S_t}$: $\mathbf{S_t} = \mathbf{S_t} \cup \{s'\}$
   (e) update s_c : $s_c \leftarrow s'$.
3. Return $\mathbf{S_t}$.

Appendix I

Examples of Non-Convergence to a Static Policy for $\varepsilon > 0$

This analysis is based on that presented by Perkins [2001] in which they demonstrate that SARSA(λ) using ε-greedy exploration cannot be guaranteed to converge to a single policy for any λ and for all $0 < \varepsilon < \varepsilon_0$ for some $\varepsilon_0 > 0$. We extend this analysis to the algorithms CEQ(λ), CEQ(λ)$^+$ and Monte-Carlo-with-exploring-starts. CEQ(λ) and CEQ(λ)$^+$ are new algorithms which we develop and introduce in this thesis; algorithm 4 on page 235 and algorithm 12 on page 386.

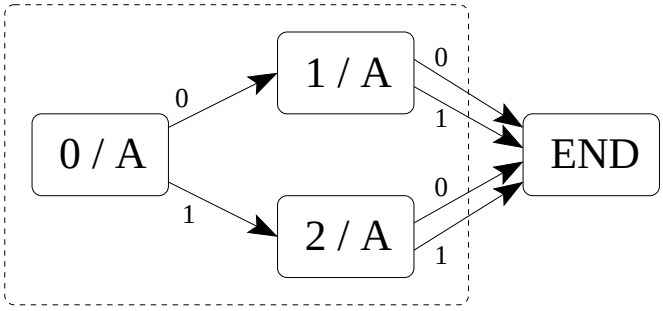


Figure I.1: Counter example to convergence of CEQ(λ) for $\varepsilon > 0$

Figure I.1 depicts the POMDP with rounded boxes indicating states and arrows indicating actions. There are three non-terminal states labelled 0, 1 and 2 all of which

give observation A, and a single terminal state labelled End. From the three non-terminal states two actions are allowed, action 0 and action 1. The arrows indicated the transitions that each action affords. All transitions are deterministic. Reward is only received when entering the terminal state and the value of the reward depends on the action sequence taken to get there. The reward structure is set out in table I.1. We let these rewards have positive values such that $r_0 < r_3 < r_1 = r_2$. Additionally there is no discounting of future rewards, that is, $\gamma = 1$.

Action sequence	Terminal reward
0 0	r_0
0 1	r_1
1 0	r_2
1 1	r_3

Table I.1: Reward on reaching End

The proof can be summarised as follows. A typical reinforcement learning (RL) algorithm attempts to estimate Q values through sampling of the observation-action space. Given that all three states yield the same observation there are only two observation-action values Q_0 and Q_1, which corresponding to selecting action 0 and action 1 respectively. Further, as states 1, 2 and 3 return the same observation there are only two possible *deterministic reactive* policies, either always take action 0 or always take action 1.

Now suppose that a RL algorithm using ε-greedy action selection could converge to the policy of always taking action 0 for some small ε. Then the action sequence 0 0 would occur most frequently, sequences 0 1 and 1 0 would occur less frequently and 1 1 would be the lease frequently occurring sequence. Given that the sequence 0 0 will be the most dominant sequence where samples relating to Q_0 are obtained, then the RL algorithm would arrive at a value of Q_0 which is close to r_0. For Q_1 samples obtained from sequence 1 1 are very rare, instead the majority of samples relating to the value of Q_1 would be obtained from the sequences 0 1 and 1 0. Thus the value learnt for Q_1 tends towards $r_1 = r_2$. Given that $r_1 = r_2 > r_0$ then Q_1 has a greater

value than Q_0 contradicting our assumption of a convergence to the policy of always selecting action 0. Similarly an assumption that the algorithm converging to the policy of always selection action 1 results in a contradiction where $Q_0 > Q_1$. Thus the RL algorithm must indefinitely switch between the two policies.

Perkins [2001] presented a formalised proof of the above for ε-greedy SARSA(λ) when using either replacing or accumulating eligibility traces. We present below a similar formal proof for ε-greedy CEQ(λ) using replacing traces.

I.1 CEQ(λ)

Assume ε-greedy CEQ(λ) has converged to the policy of always taking action 0 for some small ε. Table I.2 below sets out for each action sequence the probability of that action sequence occurring and the updates that would occur during that episode[1].

Collecting the updates for Q_0 we find the fixed point is at:

$$(1-\varepsilon)^2(r_0 - Q_0) + (1-\varepsilon)\varepsilon(r_2 - Q_0) = 0$$

which rearranges to:

$$Q_0 = (1-\varepsilon)r_0 + \varepsilon r_2$$

Similarly for Q_1 we have:

$$(1-\varepsilon)\varepsilon(r_1 - Q_1) + \varepsilon^2(r_3 - Q_1) = 0$$

rearranging to give:

$$Q_1 = (1-\varepsilon)r_1 + \varepsilon r_3$$

[1]Note that the CEQ(λ) as presented in algorithm 4 on page 235 selects the next action to execute using the current policy $\pi = \arg_a \max Q(o,a)$ and not the modified form of the current policy $\pi \leftarrow (o^e, a^e)$. Selecting the exploratory action 1 once therefore does not increase the likelihood of selecting it again. Hence the probabilities shown in table I.2 for CEQ(λ) are identical to those that would be expected for SARSA(λ).

Action sequence	Probability	Updates
0 0	$(1-\varepsilon)^2$	$Q_0 \leftarrow Q_0 + \alpha(Q_0 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha(r_0 - Q_0)$
0 1	$(1-\varepsilon)\varepsilon$	$Q_0 \leftarrow Q_0 + \alpha(Q_0 - Q_0)$, $Q_1 \leftarrow Q_1 + \alpha(r_1 - Q_1)$
1 0	$(1-\varepsilon)\varepsilon$	$Q_1 \leftarrow Q_1 + \alpha(Q_1 - Q_1)$, $Q_0 \leftarrow Q_0 + \alpha(r_2 - Q_0)$
1 1	ε^2	$Q_1 \leftarrow Q_1 + \alpha(Q_1 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha(r_3 - Q_1)$

Table I.2: Probability of action sequences and the updates which will occur using ε-greedy CEQ(λ) assuming a policy of always selecting action 0. For action sequences 0 1 and 1 0 the eligibility trace is truncated when the exploratory action changes

Action sequence	Probability	Updates
0 0	ε^2	$Q_0 \leftarrow Q_0 + \alpha(Q_0 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha(r_0 - Q_0)$
0 1	$(1-\varepsilon)\varepsilon$	$Q_0 \leftarrow Q_0 + \alpha(Q_0 - Q_0)$, $Q_1 \leftarrow Q_1 + \alpha(r_1 - Q_1)$
1 0	$(1-\varepsilon)\varepsilon$	$Q_1 \leftarrow Q_1 + \alpha(Q_1 - Q_1)$, $Q_0 \leftarrow Q_0 + \alpha(r_2 - Q_0)$
1 1	$(1-\varepsilon)^2$	$Q_1 \leftarrow Q_1 + \alpha(Q_1 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha(r_3 - Q_1)$

Table I.3: Probability of action sequences and the updates which will occur using ε-greedy CEQ(λ) assuming a policy of always selecting action 1. For action sequences 0 1 and 1 0 the eligibility trace is truncated when the exploratory action changes

Now as $\varepsilon \to 0$, $Q_0 \to r_0$ and $Q_1 \to r_1$. Given that $r_1 > r_0$ then for some ε near zero $Q_1 > Q_0$. Thus if we assume CEQ(λ) does ε-greedy exploration and ε is small, and that the algorithm has converged to the policy of always taking action 0, then it should learn a value of Q_1 greater than Q_0. Thus action 0 is not greedy which contradicts convergence to that policy.

Now if we assume ε-greedy CEQ(λ) has converged to the policy of always taking action 1 for some small ε then we arrive at table I.3.

Which in turn gives:

$$Q_0 = \varepsilon r_0 + (1-\varepsilon) r_2$$

$$Q_1 = \varepsilon r_1 + (1-\varepsilon) r_3$$

As $\varepsilon \to 0$ then $Q_0 \to r_2$ and $Q_1 \to r_3$. Given $r_2 > r_3$ then for some ε near zero $Q_0 > Q_1$. This contradicts the assumption that the algorithm has converged to the greedy policy of always taking action 1.

Having shown that convergence to greedily following either deterministic policy is not possible the algorithm must therefore flip between these two policies as they are the only ones which it can represent.

I.2 CEQ(λ)$^+$

We had intended that the selection of an exploratory action in CEQ(λ) should promote the chances of that action be selected on subsequent occasions thus obtaining more samples of its value. Having recognised through the above analysis that this does not occur we propose CEQ(λ)$^+$ as presented in algorithm 12 on page 386. This algorithm is identical to CEQ(λ) with the exception that it selects the next action to execute using the modified form of the current policy $\pi \leftarrow (o^e, a^e)$. We analysis CEQ(λ)$^+$ below to see how this change affects the stability of policies on the above example.

Table I.4 sets out the revised probabilities for each action sequence using ε-greedy CEQ(λ)$^+$ assuming that the algorithm has converged to always selection action 0. The updates are identical to those for CEQ(λ) the only change is in the frequency with which each sequence occurs. Sequence 1 1 is now as likely to occur as sequence 0 1. Sequence 1 0 is the least likely to occur as this requires two modifications of the policy.

This results in the following fixed points for Q_0 and Q_1:

$$(1-\varepsilon)^2(r_0 - Q_0) + \varepsilon^2(r_2 - Q_0) = 0$$

$$Q_0 = \frac{(1-\varepsilon)^2 r_0 + \varepsilon^2 r_2}{(1-\varepsilon)^2 + \varepsilon^2}$$

$$(1-\varepsilon)\varepsilon(r_1 - Q_1) + (1-\varepsilon)\varepsilon(r_3 - Q_1) = 0$$

$$Q_1 = \frac{r_1 + r_3}{2}$$

As $\varepsilon \to 0$ then $Q_0 \to r_0$, while $Q_1 = \frac{r_1+r_3}{2}$. Given $r_0 < r_3 < r_1$ then $r_0 < \frac{r_1+r_3}{2}$ thus for some small value of ε, $Q_0 < Q_1$. Which contradicts the assumption that the algorithm has converged to the *greedy* policy of always selection action 0.

Now if we assume instead that ε-greedy CEQ(λ)$^+$ has converged to always selection action 1 we get table I.5.

From this we see that the fixed points for Q_0 and Q_1 are:

$$(1-\varepsilon)\varepsilon(r_0 - Q_0) + (1-\varepsilon)\varepsilon(r_2 - Q_0) = 0$$

$$Q_0 = \frac{r_0 + r_2}{2}$$

$$\varepsilon^2(r_1 - Q_1) + (1-\varepsilon)^2(r_3 - Q_1) = 0$$

$$Q_1 = \frac{\varepsilon^2 r_1 + (1-\varepsilon)^2 r_3}{\varepsilon^2 + (1-\varepsilon)^2}$$

Action sequence	Probability	Updates
0 0	$(1-\varepsilon)^2$	$Q_0 \leftarrow Q_0+\alpha(Q_0-Q_0)$, $Q_0 \leftarrow Q_0+\alpha(r_0-Q_0)$
0 1	$(1-\varepsilon)\varepsilon$	$Q_0 \leftarrow Q_0+\alpha(Q_0-Q_0)$, $Q_1 \leftarrow Q_1+\alpha(r_1-Q_1)$
1 0	ε^2	$Q_1 \leftarrow Q_1+\alpha(Q_1-Q_1)$, $Q_0 \leftarrow Q_0+\alpha(r_2-Q_0)$
1 1	$(1-\varepsilon)\varepsilon$	$Q_1 \leftarrow Q_1+\alpha(Q_1-Q_1)$, $Q_1 \leftarrow Q_1+\alpha(r_3-Q_1)$

Table I.4: Probability of action sequences and the updates which will occur using ε-greedy CEQ(λ)$^+$ assuming a policy of always selecting action 0. For action sequences 0 1 and 1 0 the eligibility trace is truncated when the exploratory action changes

Action sequence	Probability	Updates
0 0	$(1-\varepsilon)\varepsilon$	$Q_0 \leftarrow Q_0+\alpha(Q_0-Q_0)$, $Q_0 \leftarrow Q_0+\alpha(r_0-Q_0)$
0 1	ε^2	$Q_0 \leftarrow Q_0+\alpha(Q_0-Q_0)$, $Q_1 \leftarrow Q_1+\alpha(r_1-Q_1)$
1 0	$(1-\varepsilon)\varepsilon$	$Q_1 \leftarrow Q_1+\alpha(Q_1-Q_1)$, $Q_0 \leftarrow Q_0+\alpha(r_2-Q_0)$
1 1	$(1-\varepsilon)^2$	$Q_1 \leftarrow Q_1+\alpha(Q_1-Q_1)$, $Q_1 \leftarrow Q_1+\alpha(r_3-Q_1)$

Table I.5: Probability of action sequences and the updates which will occur using ε-greedy CEQ(λ)$^+$ assuming a policy of always selecting action 1. For action sequences 0 1 and 1 0 the eligibility trace is truncated when the exploratory action changes

As $\varepsilon \to 0$ then $Q_1 \to r_3$, while $Q_0 = \frac{r_0+r_2}{2}$. We assumed in setting up the problem that $r_0 < r_3 < r_1 = r_2$. If we introduce the additional constraint that $r_1 = r_2 > 2r_3$ then we can say that $\frac{r_0+r_2}{2} > r_3$. Thus for some small value of ε, $Q_0 > Q_1$ which contradicts the assumption that the algorithm has converged to the *greedy* policy of always selection action 1.

Thus we show that ε-greedy CEQ$(\lambda)^+$ cannot be guaranteed to converge to a single deterministic policy for $\varepsilon > 0$.

I.3 Monte Carlo with Exploring-Starts

Given Pendrith and McGarity [1998] prove of the existence of stationary points in the observation-action value space for undiscounted first-visit Monte Carlo methods, it is useful to show that the theoretical existence of such points does not result in static policies.

Tables I.6 and I.7 set out the probabilities of action sequences and the resulting updates for a first-visit Monte Carlo algorithm using exploring-starts. We assume the exploring-starts has a uniform probability of selecting between the two possible starting actions. After the initial exploratory action the algorithm then follows policy. We assume in table I.6 that the Monte Carlo algorithm has converged to always selection action 0. Similarly table I.7 sets out the probability of action sequences and the updates assuming that the Monte Carlo algorithm has converged to always selection action 1.

Q_0 and Q_1 are updated as averages of the two sets of return values, that is, $Q_0 \leftarrow average(Returns(A,0))$, $Q_1 \leftarrow average(Returns(A,1))$. Now if we assumed that the Monte Carlo algorithm has converged to the greedy policy of always selecting action 0, then it can be seen from table I.6 that when using exploring-starts the value of $average(Returns(A,0))$ will converge towards $\frac{r_0+r_2}{2}$. It can also be seen from table I.6 that $average(Returns(A,1))$ will converge towards r_2. Given $r_0 < r_2$ then $\frac{r_0+r_2}{2} < r_2$ and hence given sufficient episodes $average(Returns(A,0)) < average(Returns(A,1))$, which in turn implies $Q_0 < Q_1$. This indicates that action 1 eventually becomes more

Action sequence	Probability	Updates
0 0	0.5	$Returns(A,0) \leftarrow Returns(A,0) \cup r_0$
0 1	0	
1 0	0.5	$Returns(A,1) \leftarrow Returns(A,1) \cup r_2$, $Returns(A,0) \leftarrow Returns(A,0) \cup r_2$
1 1	0	

Table I.6: Probability of action sequences and the updates which will occur using first-visit Monte Carlo algorithm using exploring-starts assuming a policy of always selecting action 0

Action sequence	Probability	Updates
0 0	0	
0 1	0.5	$Returns(A,0) \leftarrow Returns(A,0) \cup r_1$, $Returns(A,1) \leftarrow Returns(A,1) \cup r_1$
1 0	0	
1 1	0.5	$Returns(A,1) \leftarrow Returns(A,1) \cup r_3$

Table I.7: Probability of action sequences and the updates which will occur using first-visit Monte Carlo algorithm using exploring-starts assuming a policy of always selecting action 1

valued than action 0 which is in direct contradiction to our assumption of convergence to the greedy policy of always selecting action 0. We note that this will occur regardless of the initial values contained in the two sets of returns. Updates will continually added new values to both sets until these new values dominate any initial values.

Now assume that the exploring-starts first-visit Monte Carlo algorithm has converged to always selecting action 1. From table I.7 we can see that $average(Returns(A,0))$ will converge towards r_1 and $average(Returns(A,1))$ will converge towards $\frac{r_1+r_3}{2}$. Given $r_3 < r_1$ then $\frac{r_1+r_3}{2} < r_1$ and hence given sufficient episode $average(Returns(A,1)) < average(Returns(A,0))$ which implies $Q_1 ¡ Q_0$. This indicates that action 0 eventually becomes regarded as more valuable than action 1 which is in contradiction to our assumption of convergence to a greedy policy.

Thus we demonstrate that first-visit Monte Carlo using exploring-starts cannot converge to a static policy on the given POMDP example. This clearly indicates that the guarantee of convergence to a stationary point in the observation-action *value* space [Pendrith and McGarity, 1998] does not imply convergence to a stationary policy.

Appendix J

CEQ(λ)$^+$

A version of CEQ(λ) which ensures that the probabilities used in selecting the next action take into account the current modified policy $\pi \leftarrow (o^e, a^e)$, where $\pi = \arg_a \max Q(o,a)$ and a^e is the exploratory action being executed when observation o^e occurs.

As described in algorithm 4, page 235, CEQ(λ) selects the next action to execute in line with the current policy $\pi = \arg_a \max Q(o,a)$ with probability $1-\varepsilon$ and varies from policy π with probability ε. Thus the probability of selecting the next action does not take into account the modified policy $\pi \leftarrow (o^e, a^e)$ which *is* used in calculating updates and truncate the eligibility traces. The proposed algorithm CEQ(λ)$^+$ fixes this apparent inconsistency.

Algorithm 12 CEQ(λ)$^+$, a version of CEQ(λ) which ensures that the probabilities used in selecting the next action take into account the current modified policy $\pi \leftarrow (o^e, a^e)$. The change to the algorithm is highlighted in bold font. Algorithm shown using replacing eligibility traces.

Initialise $Q(o,a) \leftarrow$ arbitrary values
Repeat (for each episode):
  Initialise $e(o,a) = 0$ for all o, a
  Initialise start state
  get observation o and select a using policy derived from Q (for example, ε-greedy)
  if $a \neq \arg\max_b Q(o,b)$, then $a^e = a$; $o^e = o$
    else $a^e = o^e =$ none
  Repeat (for each step of episode):
    take action a, obtain r and observe o'
    if $o' = o^e$, then $a^* \leftarrow a^e$
      else $a^* \leftarrow \arg\max_b Q(o',b)$
    $\delta = r + \gamma Q(o', a^*) - Q(o,a)$
    $e(o,a) \leftarrow 1$
    for all o, a:
      $Q(o,a) \leftarrow Q(o,a) + \alpha\delta e(o,a)$
    choose a' for o' using a search strategy (for example ε-greedy) which is applied to policy $\pi \leftarrow (o^e, a^e)$ where π is $\arg_a \max Q(o,a)$
    for all o, a:
      if $a' = a^*$, then $e(o,a) \leftarrow \gamma\lambda e(o,a)$
        else $e(o,a) \leftarrow 0$
    if $a' \neq a^*$ then $a^e = a'$, $o^e = o'$
    $o \leftarrow o'$; $a \leftarrow a'$
  until terminal state is reached

Appendix K

CE-SARSA(λ)

The proposed Consistent Exploration form of SARSA(λ), CE-SARSA(λ), is shown below in algorithm 13. It is based on the conventional form of tabular SARSA(λ) as presented in appendix D but with the modification that the most recently selected exploratory action is recorded (a^e) and will be carried out each time the observation where it was first executed is re-observed (o^e). Both a^e and o^e are updated when the next exploratory action is selected.

CE-SARSA(λ) differs from CEQ(λ) and CEQ(λ)$^+$ in that the eligibility trace is not truncated when an exploratory action is taken. Like CEQ(λ)$^+$ the probability of the next action to be selected is influenced by the modified policy $\pi \leftarrow (o^e, a^e)$.

Algorithm 13 Proposed Consistent Exploration form of SARSA(λ), CE-SARSA(λ) using replacing eligibility traces.

Initialise $Q(o,a) \leftarrow$ arbitrary values
Repeat (for each episode):
  Initialise $e(o,a) = 0$ for all o, a
  Initialise start state
  get observation o and select a using policy derived from Q (for example, ε-greedy)
  if $a \neq \arg\max_b Q(o,b)$, then $a^e = a$; $o^e = o$
    else $a^e = o^e =$ none
  Repeat (for each step of episode):
    take action a, obtain r and observe o'
    choose a' for o' using policy derived from Q (for example, ε-greedy)
    if $a' \neq \arg\max_b Q(o',b)$, then $a^e = a'$; $o^e = o'$
      else if $o' = o^e$, then $a' \leftarrow a^e$
    $\delta = r + \gamma Q(o',a') - Q(o,a)$
    $e(o,a) \leftarrow 1$
    for all o, a:
      $Q(o,a) \leftarrow Q(o,a) + \alpha\delta e(o,a)$
      $e(o,a) \leftarrow \gamma\lambda e(o,a)$
    $o \leftarrow o'$; $a \leftarrow a'$
  until terminal state is reached

Appendix L

Random Grid World Generation Algorithm

The algorithm used to generate the ten random grid worlds used in chapter 8 is set out below.

Algorithm 14 Random grid world generator

Start with an empty, bounded 9×7 grid world

N = 1 + | random value from Gaussian distribution mean 0 and standard deviation of 1 |

place N *positive-goals* in empty squares of the grid world

M = | random value from Gaussian distribution mean 0 and standard deviation of 1.47 |

place M *penalty-goals* in empty squares of the grid world

B_5 = random value from Gaussian distribution mean 0.75 and standard deviation of 0.5

if $B_5 > 0$ then place B_5 blocks, which occupy either 5×1, or 1×5 grid squares, randomly in the grid world such that they do not overlap any goals or previously placed blocks

B_4 = random value from Gaussian distribution mean 1.5 and standard deviation of 1.21

if $B_4 > 0$ then place B_4 blocks, which occupy 4×1, or 1×4 grid squares, randomly in the grid world such that they do not overlap any goals or previously placed blocks

B_3 = random value from Gaussian distribution mean 3 and standard deviation of 1

if $B_3 > 0$ then place B_3 blocks, which occupy 3×1, or 1×3 grid squares, randomly in the grid world such that they do not overlap any goals or previously placed blocks

B_2 = random value from Gaussian distribution mean 1.67 and standard deviation of 1

if $B_2 > 0$ then place B_2 blocks, which occupy 2×1, or 1×2 grid squares, randomly in the grid world such that they do not overlap any goals or previously placed blocks

B_1 = random value from Gaussian distribution mean 3 and standard deviation of 2

if $B_1 > 0$ then place B_1 blocks, which occupy 1 grid square, randomly in the grid world such that they do not overlap any goals or previously placed blocks

starting from positive-goals flood fill grid with a marker. Stop flood fill at squares filled with blocks or penalty goals.

fill in any part of the grid that does not contain the marker with blocks.

remove markers.

remove any penalty goals that are stranded; that is, surrounded by blocks in the four principal compass directions. Replace with blocks

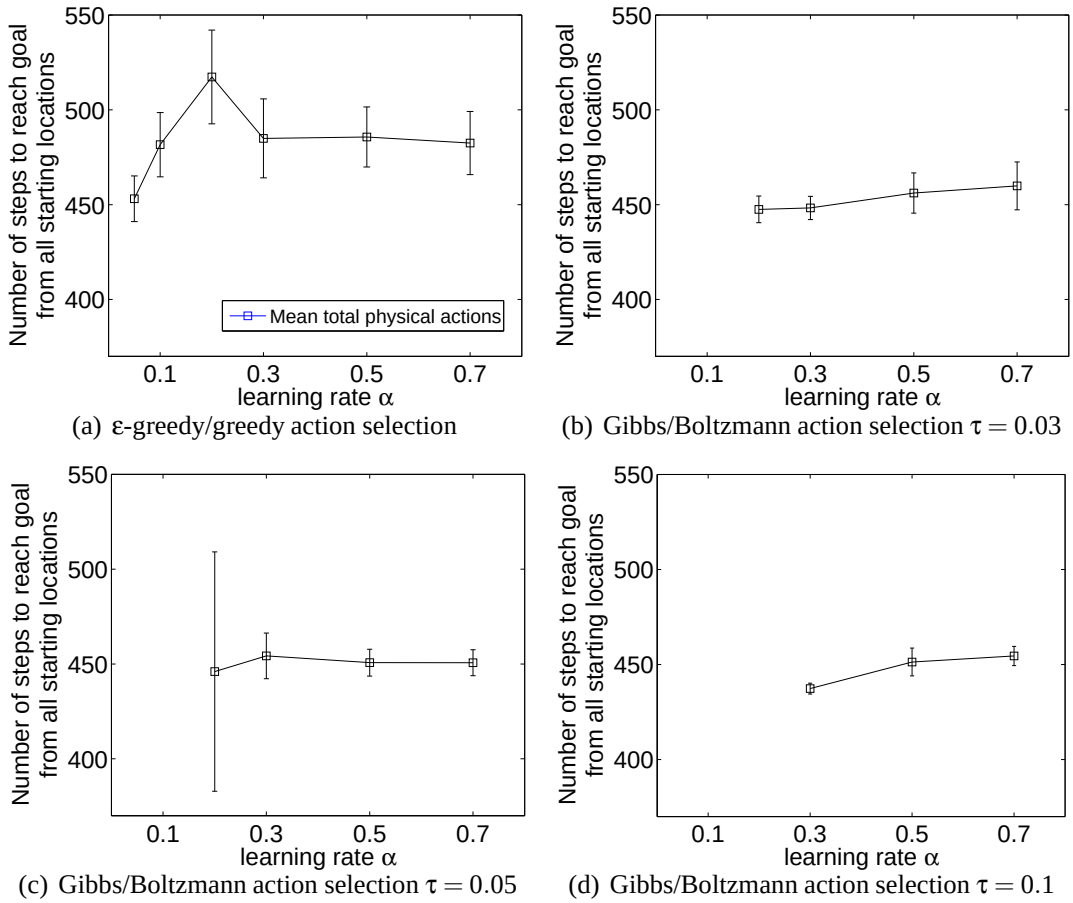
check if goal states are stranded. If all are stranded then redo from start.

Appendix M

Breakdown of figures 5.8(a) and 5.8(b)

Breakdown of plots shown in figure 5.8 on page 174 in section 5.2.2. These show the change in quality of the policies with variation of learning rate α. Results are shown for each combination of learning algorithm and action selection parameters. We measure the quality of a policy in terms of the total steps taken to reach the goal location from all start states. This averaged over all satisficing policies learnt after one million action-learning steps. These results demonstrate that there is no significant difference in policy degradation between SARSA and Q-learning or between the action selection methods.

Note only ε-greedy/greedy and τ values of 0.03, 0.05 and 0.1 are plotted. We deemed that there were insufficient satisficing policies for the remaining two values of τ to make the plots meaningful.



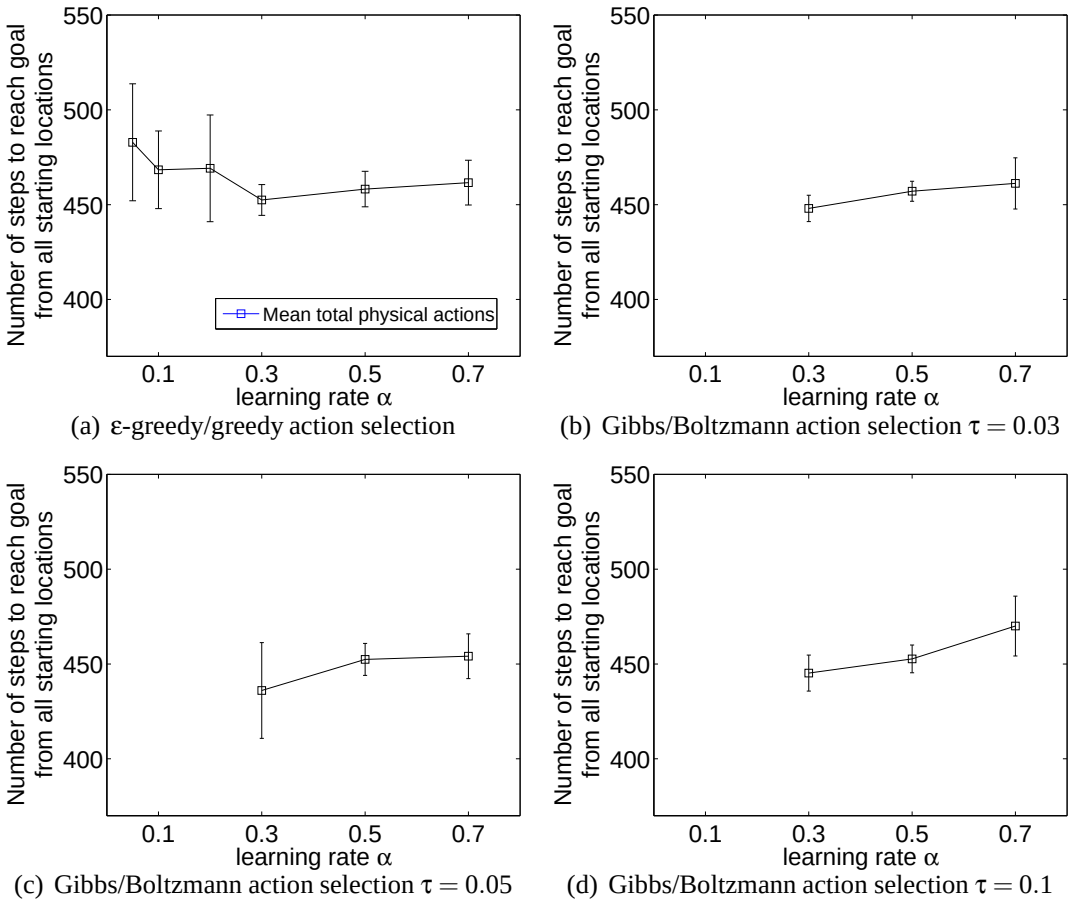
(a) ε-greedy/greedy action selection

(b) Gibbs/Boltzmann action selection $\tau = 0.03$

(c) Gibbs/Boltzmann action selection $\tau = 0.05$

(d) Gibbs/Boltzmann action selection $\tau = 0.1$

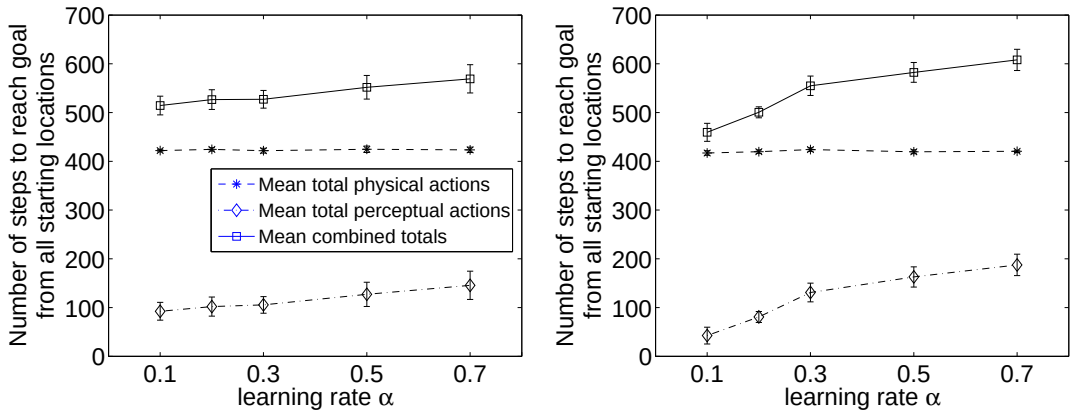
Figure M.1: Q-learning, Eight Adjacent Squares Agent. Plot of mean total physical action steps for satisficing policies against variation in learning rate α. Error bars indicate 95% confidence intervals



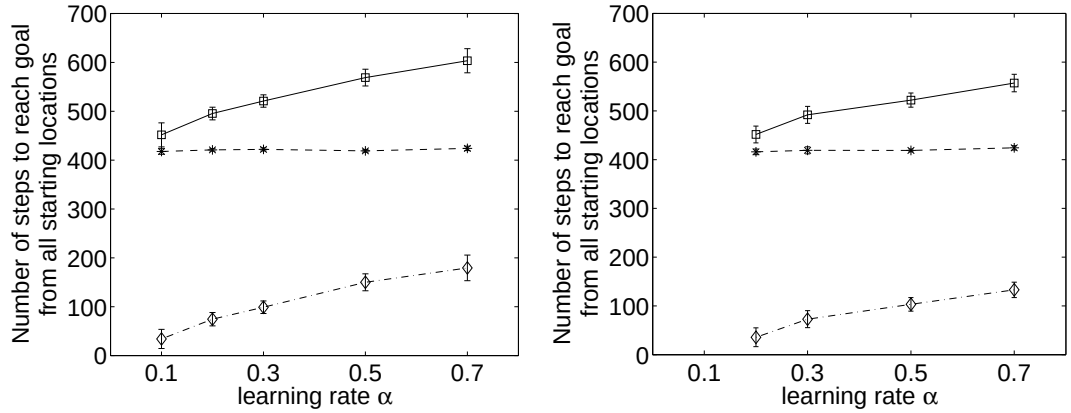
(a) ε-greedy/greedy action selection (b) Gibbs/Boltzmann action selection $\tau = 0.03$

(c) Gibbs/Boltzmann action selection $\tau = 0.05$ (d) Gibbs/Boltzmann action selection $\tau = 0.1$

Figure M.2: SARSA, Eight Adjacent Squares Agent. Plot of mean total physical action steps for satisficing policies against variation in learning rate α. Error bars indicate 95% confidence intervals

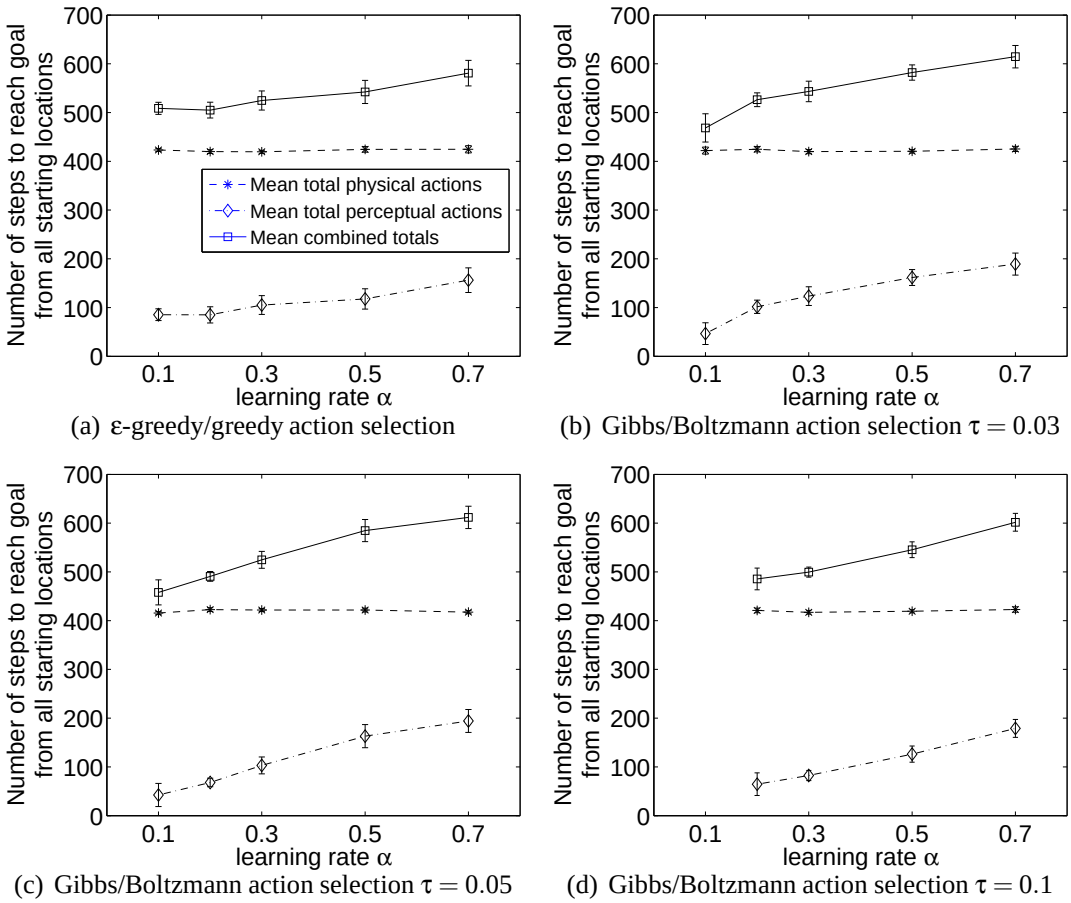


(a) ε-greedy/greedy action selection (b) Gibbs/Boltzmann action selection $\tau = 0.03$



(c) Gibbs/Boltzmann action selection $\tau = 0.05$ (d) Gibbs/Boltzmann action selection $\tau = 0.1$

Figure M.3: Q-learning, Active Perception Agent. Plot of mean total physical, perceptual and combined action steps for satisficing policies against variation in learning rate α. Error bars indicate 95% confidence intervals



(a) ε-greedy/greedy action selection

(b) Gibbs/Boltzmann action selection $\tau = 0.03$

(c) Gibbs/Boltzmann action selection $\tau = 0.05$

(d) Gibbs/Boltzmann action selection $\tau = 0.1$

Figure M.4: SARSA, Active Perception Agent. Plot of mean total physical, perceptual and combined action steps for satisficing policies, against variation in learning rate α. Error bars indicate 95% confidence intervals

Appendix N

Detailed Comparison of Mean Total Steps for Section 7.1.3

A set of experiments comparing the performance of SARSA(λ) and CEQ(λ) over three test problems are presented in section 7.1.3. The tables included in this appendix provide a detailed comparison of the quality of the satisficing policies learnt by the two learning algorithms broken down by all parameter combinations tested. Quality is measured as the total steps taken by a satisficing policy to reach the goal state from all possible starting states. For each combination of parameter values one hundred trials were run allowing the mean total path length and standard deviation to be calculated[1]. The results are presented in tables N.1 to N.6. For each parameter combination in the tables two figures are presented, the top left bold value indicating the values for CEQ(λ), bottom right the result for SARSA(λ).

N.1 First Test Problem

The first test problem consists of Sutton's Grid World and the Fixed Eight Adjacent Squares Agent. The values presented in tables N.1 and N.2 underpin the results pre-

[1] NB not all one hundred trials will be satisficing, the number of trials used to calculate each mean and standard deviation figure can be found by consulting tables 7.1 to 7.3

sented in figure 7.4 and towards the end of section 7.1.4.1.

N.2 Second Test Problem

The second test problem consists of Sutton's Grid World and the Active Perception Agent. The values presented in tables N.3 and N.4 underpin the results presented in figures 7.5(a) and 7.5(b), and also the overall averages presented at the end of section 7.1.4.2.

N.3 Third Test Problem

The third test problem consists of Wilson's Woods 7 grid world and a modified fixed perception agent as described in section 7.1.3.4. The values presented in tables N.5 and N.6 underpin the results presented in figure 7.6 and towards the end of section 7.1.4.3.

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**433.7±30.7** / 437.0±22.3	**422.1±8.7** / 426.6±17.1	**423.0±9.8** / 425.5±10.7	**434.1±5.9** / 433.4±4.0	**435.4±4.4** / 438.5±15.4
0.9	**436.1±28.5** / 431.6±23.3	**422.3±11.3** / 426.1±14.7	**419.8±7.6** / 419.6±6.6	**420.0±7.6** / 417.7±5.1	**431.8±7.9** / 440.0±21.0
0.99	**438.0±27.3** / 443.0±25.2	**424.2±11.3** / 428.1±14.2	**419.8±6.9** / 419.0±5.7	**416.9±3.7** / 416.6±2.5	**422.1±8.7** / 444.4±30.5
1.0	**439.3±28.8** / 450.3±36.4	**422.1±9.5** / 427.8±16.3	**419.1±6.2** / 419.0±7.0	**416.6±3.0** / 417.7±6.9	**420.3±7.5** / 451.1±33.8

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**425.2±16.9** / 420.8±11.1	**420.1±7.6** / 419.2±6.0	**421.7±8.7** / 419.8±7.1	**430.2±8.4** / 426.8±8.5	**434.6±3.7** / 439.4±16.4
0.9	**424.5±18.0** / 427.3±17.8	**419.9±7.4** / 419.5±6.3	**418.3±5.8** / 417.0±3.7	**418.8±6.2** / 417.5±4.4	**430.0±7.6** / 440.7±27.0
0.99	**422.3±9.1** / 437.4±24.8	**419.8±7.1** / 426.1±17.2	**418.6±5.5** / 417.1±3.2	**417.4±3.4** / 422.0±15.1	**422.3±13.8** / 475.7±52.1
1.0	**425.6±18.0** / 444.2±33.7	**419.1±7.0** / 426.2±19.3	**417.6±3.9** / 417.1±3.6	**416.5±0.8** / 426.4±18.8	**421.9±18.3** / 479.9±48.5

Table N.1: Average total steps taken by satisficing policies on 1^{st} test problem. Results for $\gamma = 0.8$ and $\gamma = 0.9$. Top left bold figures CEQ(λ), bottom right SARSA(λ). Figures presented are mean total steps $\pm$ standard deviation

(a) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**419.6±6.8** / 418.2±7.0	**420.5±7.8** / 418.6±5.9	**420.6±8.0** / 417.9±5.5	**426.9±9.5** / 420.6±7.8	**435.3±9.7** / 444.6±26.0
0.9	**424.0±17.1** / 424.0±16.4	**420.1±12.9** / 420.1±9.3	**418.6±6.3** / 416.9±3.7	**419.5±6.9** / 425.4±18.0	**428.2±10.6** / 473.3±43.6
0.99	**424.0±18.7** / 450.7±33.2	**420.1±6.5** / 438.2±26.0	**418.2±4.2** / 449.8±36.8	**417.9±3.0** / 490.1±49.1	**433.8±31.4** / 529.6±77.3
1.0	**424.6±15.3** / 460.6±35.8	**419.2±5.6** / 442.1±28.5	**418.1±3.8** / 480.2±53.1	**418.9±5.0** / 513.0±62.9	**433.6±29.4** / 536.9±62.3

(b) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**420.3±10.1** / 420.5±16.9	**420.8±8.0** / 418.0±5.1	**422.4±9.0** / 417.4±4.5	**425.4±9.4** / 422.2±7.9	**435.0±5.0** / 447.4±19.6
0.9	**424.4±18.3** / 421.8±11.8	**418.9±5.9** / 420.3±12.1	**420.2±7.1** / 418.0±7.0	**419.4±6.5** / 427.1±21.1	**428.7±13.5** / 481.5±51.3
0.99	**422.9±14.4** / 458.2±37.9	**419.5±5.2** / 439.2±19.4	**418.8±4.1** / 480.8±58.8	**418.4±4.0** / 512.0±61.5	**438.8±32.5** / 556.4±71.2
1.0	**423.8±15.7** / 502.9±53.8	**419.9±5.7** / 509.7±61.6	**418.3±3.5** / 535.6±61.7	**418.3±4.1** / 460.0±27.8	**444.9±35.9** / —

Table N.2: Average total steps taken by satisficing policies on 1^{st} test problem. Results for $\gamma = 0.99$ and $\gamma = 0.1$. Top left bold figures CEQ(λ), bottom right SARSA(λ). Figures presented are mean total steps $\pm$ standard deviation

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**478.0±39.1** 510.8±41.1	**538.6±61.3** 492.0±53.4	**432.0±7.8** 435.3±11.6	**433.2±14.3** 442.8±20.7	**479.9±38.0** 523.5±61.3
0.9	**483.6±46.8** 526.3±51.0	**518.7±54.6** 466.0±42.7	**430.7±12.9** 430.7±8.1	**430.3±12.9** 435.5±16.0	**474.9±39.0** 537.9±58.0
0.99	**501.0±37.6** 550.9±62.0	**510.4±56.9** 475.1±44.0	**425.8±8.4** 432.0±11.9	**425.5±12.3** 444.0±19.3	**477.7±35.0** 570.0±63.1
1.0	**496.3±46.2** 556.3±72.7	**506.8±49.2** 482.8±48.4	**425.7±8.5** 435.5±15.0	**425.5±11.5** 446.2±23.8	**491.1±41.1** 579.9±63.4

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**506.7±46.7** 536.1±51.8	**526.0±58.8** 456.2±39.1	**429.8±7.9** 428.9±6.2	**433.1±13.9** 447.0±28.8	**493.8±42.9** 550.5±56.9
0.9	**518.1±52.2** 537.5±59.1	**508.6±58.1** 443.8±24.0	**425.7±10.3** 428.1±10.2	**431.4±19.5** 457.5±26.8	**529.5±53.7** 574.9±73.6
0.99	**533.9±48.8** 547.1±64.7	**511.2±65.8** 463.9±27.6	**424.6±11.9** 438.4±14.8	**441.4±25.7** 479.3±41.2	**546.0±60.3** 653.8±82.8
1.0	**534.6±52.9** 552.6±66.5	**497.4±60.9** 470.9±33.4	**425.8±10.9** 440.7±21.7	**440.0±22.0** 491.5±42.8	**552.4±61.8** 639.0±79.5

Table N.3: Average total steps taken by satisficing policies on 2^{nd} test problem. Results for $\gamma = 0.8$ and $\gamma = 0.9$. Top left bold figures CEQ(λ), bottom right SARSA(λ). Figures presented are mean total steps $\pm$ standard deviation

(a) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**571.1±61.7** / 533.0±55.3	**524.9±61.9** / 447.8±34.1	**430.1±8.4** / 436.9±20.3	**434.9±14.1** / 475.9±37.1	**527.7±50.3** / 589.3±69.9
0.9	**575.3±72.0** / 537.8±59.2	**508.3±64.4** / 445.2±20.7	**430.8±14.4** / 448.1±25.0	**451.3±28.8** / 487.4±43.8	**557.8±55.4** / 649.3±74.2
0.99	**575.4±65.5** / 588.5±65.3	**504.8±61.4** / 513.3±47.8	**437.4±17.7** / 503.5±36.6	**480.6±42.0** / 591.8±77.1	**600.8±70.3** / 773.0±118.7
1.0	**576.3±54.1** / 619.9±81.5	**506.1±64.5** / 541.6±62.5	**439.2±17.3** / 527.1±58.7	**486.8±37.1** / 608.1±72.6	**605.5±70.9** / 844.1±136.2

(b) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**576.4±57.3** / 535.8±50.4	**513.0±59.6** / 449.2±36.4	**430.2±8.1** / 435.0±15.4	**438.2±19.3** / 474.4±41.3	**519.7±49.6** / 592.3±66.0
0.9	**577.1±59.2** / 532.3±60.8	**510.0±74.6** / 444.6±24.1	**433.8±15.7** / 452.8±27.2	**448.2±27.9** / 503.9±46.3	**565.8±65.6** / 656.0±75.8
0.99	**583.6±61.1** / 612.3±79.3	**498.6±67.9** / 538.8±50.8	**443.6±24.2** / 530.2±47.7	**486.6±37.8** / 632.2±81.5	**621.1±75.6** / 836.0±133.2
1.0	**601.8±64.9** / 832.1±126.1	**503.1±61.5** / 806.7±142.1	**443.0±22.0** / 741.1±171.6	**495.3±45.6** / 793.3±169.1	**612.1±74.8** / 936.6±152.4

Table N.4: Average total steps taken by satisficing policies on 2^{nd} test problem. Results for $\gamma = 0.99$ and $\gamma = 1$. Top left bold figures CEQ(λ), bottom right SARSA(λ). Figures presented are mean total steps ± standard deviation

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**4347±211** 4575±622	**4118±466** 4302±459	**3507±225** 3693±379	**3200±148** 3299±200	**3116±172** 3426±327
0.9	**4473±523** 4589±479	**4147±388** 4302±488	**3520±305** 3614±347	**3219±143** 3310±238	**3113±211** 3499±317
0.99	**4601±487** 4646±453	**4204±382** 4322±449	**3547±264** 3617±316	**3265±227** 3323±239	**3175±187** 3482±313
1.0	**4349±144** 4856±667	**4189±433** 4234±432	**3585±268** 3640±314	**3240±193** 3366±237	**3099±143** 3441±293

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**5479±1049** 5568±698	**4798±677** 5077±892	**3979±460** 4313±564	**3474±313** 3669±355	**3206±225** 3841±374
0.9	**5649±727** 5663±936	**4890±612** 5091±749	**4158±488** 4188±611	**3468±349** 3637±360	**3217±256** 3803±469
0.99	**5545±623** 5775±1175	**4842±588** 5070±725	**4156±580** 4130±585	**3604±356** 3789±456	**3286±265** 3818±343
1.0	**5725±1055** 5539±708	**4809±611** 5075±778	**4117±560** 4120±518	**3579±408** 3766±429	**3360±332** 3751±376

Table N.5: Average total steps taken by satisficing policies on 3^{rd} test problem. Results for $\gamma = 0.8$ and $\gamma = 0.9$. Top left bold figures CEQ(λ), bottom right SARSA(λ). Figures presented are mean total steps ± standard deviation

(a) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**9541±3088** / 11923±4923	**8270±2064** / 10100±2255	**6690±1302** / 7877±1552	**6314±819** / 7111±1297	**6162±1032** / 7273±756
0.9	**11166±5712** / 9158±2313	**8652±2460** / 8544±2007	**6862±1356** / 7717±1808	**6232±1046** / 6995±1454	**6157±771** / 6999±705
0.99	**9106±2425** / 8456±1896	**8485±2029** / 8507±2342	**6872±1573** / 6512±1295	**6145±926** / 5748±1048	**6386±988** / 6501±1068
1.0	**9468±3357** / 9677±4310	**8488±2177** / 8385±1970	**6631±1180** / 6377±1430	**6253±908** / 5843±1176	**6564±1214** / 6451±1260

(b) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	— / —	— / —	— / —	— / —	**8530** / 20509
0.9	— / —	— / —	— / —	**14740** / —	**11156±6104** / —
0.99	— / —	— / —	— / —	**12153** / —	**16973±3594** / —
1.0	— / —	— / —	— / —	**8471** / —	**11235±4406** / —

Table N.6: Average total steps taken by satisficing policies on 3^{rd} test problem. Results for $\gamma = 0.99$ and $\gamma = 1$. Top left bold figures CEQ(λ), bottom right SARSA(λ). Figures presented are mean total steps ± standard deviation

Appendix O

Breakdown of Mean Number of Satisficing Policies for Section 8.1

Experiments perform in section 8.1 compare the performance of CEQ(λ), SARSA(λ), Lion algorithm and U-Tree on a set of ten randomly generated grid worlds. The results presented in this appendix are the mean number of satisficing policies learnt, averaged across the ten random grid worlds, for each combination of parameter values tried. These tables allow examination of the sensitivity of each of the leaning algorithms to variation in their learning parameters. It also allows for the identification of parameter value combinations where each learning algorithm performs best, and the number of satisficing policies learnt for such combinations.

O.1 CEQ(λ) and SARSA(λ)

Tables O.1 – O.6 show results for SARSA(λ) and CEQ(λ). We take advantage of the fact that SARSA(λ) and CEQ(λ) were tested over the same parameter space by presenting the results for both algorithms on the same tables. The values in bold correspond to results for CEQ(λ) and those in normal type are the results for SARSA(λ). All results are out of 20 trials. Tables O.1 and O.2 are for the Fixed Eight Adjacent

Squares Agent, tables O.3 and O.4 are for the Active Perception Agent and tables O.5 and O.6 are for the 1-bit memory agent.

O.2 Lion Algorithm and U-Tree

We similarly present results for the Lion algorithm and U-Tree in tables O.7 and O.8 respectively. As for SARSA(λ) and CEQ(λ), the values presented are mean number of satisficing policies learnt across the ten random grid worlds, and are presented for each of the parameter combinations tested. Note that the results for U-Tree are out of 10 trials only, while those for SARSA(λ), CEQ(λ) and the Lion algorithm are out of 20.

key:
CEQ(λ)
SARSA(λ)

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**17.6±6.2** 15.9±8.4	**18.0±6.3** 15.8±8.4	**18.0±6.3** 15.8±8.4	**17.5±6.3** 15.9±8.4	**16.1±8.2** 13.6±9.4
0.9	**17.8±6.3** 15.0±8.1	**17.6±6.3** 15.7±8.3	**17.9±6.3** 15.7±8.3	**18.0±6.3** 15.5±8.3	**16.0±8.4** 13.2±9.3
0.99	**17.7±6.3** 14.7±8.1	**17.9±6.3** 15.6±8.3	**17.7±6.3** 15.4±8.3	**17.5±6.3** 14.7±8.7	**16.0±8.4** 13.8±9.5
1.0	**17.4±6.2** 15.6±8.3	**17.9±6.3** 15.6±8.3	**17.9±6.3** 15.8±8.4	**17.6±6.3** 15.1±8.4	**16.1±8.0** 13.8±9.5

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**18.0±6.3** 17.0±6.5	**18.0±6.3** 17.1±6.6	**18.0±6.3** 15.9±8.4	**18.0±6.3** 15.9±8.4	**17.1±6.6** 14.1±8.3
0.9	**17.8±6.3** 16.4±7.2	**18.0±6.3** 16.1±8.0	**18.0±6.3** 16.0±8.4	**18.0±6.3** 16.0±8.4	**17.0±6.7** 13.9±9.6
0.99	**17.9±6.3** 16.4±6.8	**18.0±6.3** 16.0±8.4	**18.0±6.3** 16.0±8.4	**18.0±6.3** 15.6±8.3	**16.6±7.1** 12.7±9.1
1.0	**18.0±6.3** 15.8±7.5	**18.0±6.3** 16.2±7.8	**18.0±6.3** 15.8±8.4	**18.0±6.3** 15.4±8.3	**16.7±6.8** 13.5±9.4

Table O.1: Number satisficing policies learnt for Eight Adjacent Squares agent out of 20 trials. Tables presented for $\gamma = 0.8$ and $\gamma = 0.9$. Values averaged across all test worlds. Top left bold figures CEQ(λ), bottom right SARSA(λ). Values shown are mean ± standard deviation

key:
CEQ(λ)
SARSA(λ)

(a) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**17.7±6.3** 17.3±6.2	**18.0±6.3** 17.7±6.3	**18.1±6.0** 18.0±6.3	**18.1±6.0** 17.9±6.3	**18.5±4.7** 17.5±6.2
0.9	**17.4±5.9** 17.2±6.4	**18.1±5.7** 17.8±6.3	**18.0±6.3** 18.0±6.3	**18.2±5.7** 18.0±6.3	**19.3±2.2** 17.0±6.1
0.99	**17.7±5.3** 17.4±6.3	**18.0±5.7** 17.8±6.3	**18.0±6.3** 17.4±6.2	**18.1±6.0** 17.6±6.3	**19.9±0.3** 16.7±5.9
1.0	**18.0±5.3** 17.0±6.3	**18.1±6.0** 17.9±6.0	**18.2±5.7** 18.0±6.0	**18.2±5.7** 17.6±4.9	**19.9±0.3** 15.1±5.5

(b) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**17.3±6.2** 16.4±6.4	**18.2±5.3** 17.2±6.3	**18.3±5.4** 18.0±6.3	**18.4±5.1** 18.0±6.3	**19.6±1.3** 18.0±5.0
0.9	**17.6±6.2** 17.5±6.3	**18.0±5.6** 17.5±6.2	**17.9±6.3** 17.9±6.3	**18.7±4.1** 18.0±6.3	**19.9±0.3** 18.3±4.0
0.99	**17.6±6.0** 17.1±6.2	**17.9±5.7** 17.8±6.3	**18.4±5.1** 18.1±6.0	**19.1±2.8** 18.3±4.0	**19.7±0.9** 17.9±3.8
1.0	**17.7±5.6** 12.6±8.1	**18.3±5.4** 11.9±8.2	**18.4±5.1** 10.6±8.5	**19.2±2.5** 9.3±8.2	**20.0±0.0** 5.0±6.1

Table O.2: Number satisficing policies learnt for Eight Adjacent Squares agent out of 20 trials. Tables presented for $\gamma = 0.99$ and $\gamma = 1.0$. Values averaged across all test worlds. Top left bold figures CEQ(λ), bottom right SARSA(λ). Values shown are mean ± standard deviation

key:
CEQ(λ)
SARSA(λ)

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**17.7±4.0** 18.4±4.0	**19.8±0.6** 17.2±6.5	**19.9±0.3** 16.0±8.4	**19.8±0.4** 15.9±8.4	**19.0±2.8** 16.0±8.4
0.9	**18.1±3.3** 17.2±6.0	**19.9±0.3** 16.8±7.0	**20.0±0.0** 16.2±8.0	**20.0±0.0** 15.9±8.4	**19.4±1.6** 15.9±8.4
0.99	**17.4±3.4** 17.0±6.5	**19.8±0.6** 16.6±7.3	**20.0±0.0** 16.0±8.4	**20.0±0.0** 16.0±8.4	**19.3±2.2** 15.6±8.2
1.0	**17.8±3.3** 16.5±7.5	**19.9±0.3** 16.5±7.5	**19.9±0.3** 16.1±8.2	**20.0±0.0** 16.0±8.4	**18.8±3.8** 15.9±8.4

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**18.9±2.3** 19.9±0.3	**20.0±0.0** 20.0±0.0	**20.0±0.0** 19.3±1.9	**20.0±0.0** 16.8±6.6	**19.7±0.5** 15.5±8.2
0.9	**19.5±0.8** 19.7±0.7	**20.0±0.0** 19.6±1.3	**20.0±0.0** 18.6±3.3	**20.0±0.0** 16.6±7.2	**19.7±0.7** 15.7±8.3
0.99	**19.3±1.3** 18.6±3.5	**19.9±0.3** 18.1±4.8	**20.0±0.0** 17.4±5.8	**20.0±0.0** 16.1±8.2	**19.9±0.3** 15.8±8.3
1.0	**19.3±1.3** 18.0±5.1	**19.9±0.3** 18.5±3.6	**19.9±0.3** 17.6±5.3	**19.9±0.3** 16.3±7.8	**20.0±0.0** 15.3±8.1

Table O.3: Number satisficing policies learnt for Active Perception agent out of 20 trials. Tables presented for $\gamma = 0.8$ and $\gamma = 0.9$. Values averaged across all test worlds. Top left bold figures CEQ(λ), bottom right SARSA(λ). Values shown are mean ± standard deviation

key:
CEQ(λ)
SARSA(λ)

(a) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**20.0±0.0** / 19.9±0.3	**20.0±0.0** / 19.8±0.4	**20.0±0.0** / 20.0±0.0	**19.8±0.6** / 20.0±0.0	**19.6±1.3** / 18.8±2.5
0.9	**19.8±0.6** / 19.7±0.7	**20.0±0.0** / 19.9±0.3	**19.7±0.9** / 19.6±1.0	**20.0±0.0** / 19.2±1.9	**19.6±1.3** / 18.0±3.4
0.99	**19.8±0.4** / 19.4±1.6	**19.9±0.3** / 19.5±1.6	**20.0±0.0** / 19.7±0.9	**20.0±0.0** / 19.1±1.7	**19.8±0.6** / 17.0±4.9
1.0	**19.7±0.5** / 19.2±2.5	**20.0±0.0** / 19.1±2.5	**20.0±0.0** / 18.7±3.8	**19.9±0.3** / 19.0±2.8	**20.0±0.0** / 17.4±4.2

(b) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**19.9±0.3** / 19.7±0.5	**20.0±0.0** / 19.5±1.0	**19.9±0.3** / 20.0±0.0	**20.0±0.0** / 20.0±0.0	**18.9±2.4** / 17.5±4.6
0.9	**19.8±0.4** / 17.9±6.0	**19.9±0.3** / 17.8±6.3	**20.0±0.0** / 18.1±5.7	**20.0±0.0** / 18.7±3.8	**19.7±0.7** / 17.2±5.3
0.99	**20.0±0.0** / 15.9±7.3	**20.0±0.0** / 16.3±6.7	**20.0±0.0** / 16.4±6.9	**20.0±0.0** / 16.0±7.9	**19.8±0.4** / 18.2±2.9
1.0	**19.9±0.3** / 14.2±8.8	**19.9±0.3** / 13.5±8.6	**19.9±0.3** / 13.7±8.6	**20.0±0.0** / 12.5±8.8	**20.0±0.0** / 12.4±8.0

Table O.4: Number satisficing policies learnt for Active Perception agent out of 20 trials. Tables presented for $\gamma = 0.99$ and $\gamma = 1.0$. Values averaged across all test worlds. Top left bold figures CEQ(λ), bottom right SARSA(λ). Values shown are mean ± standard deviation

key:
CEQ(λ)
SARSA(λ)

(a) $\gamma = 0.8$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**19.8±0.4** 16.8±7.0	**19.7±0.9** 16.5±7.4	**20.0±0.0** 16.0±8.4	**20.0±0.0** 16.0±8.4	**20.0±0.0** 15.7±8.3
0.9	**19.7±0.7** 16.4±7.6	**19.8±0.4** 16.1±8.2	**19.9±0.3** 16.0±8.4	**20.0±0.0** 16.0±8.4	**20.0±0.0** 14.9±8.4
0.99	**19.8±0.4** 16.2±7.8	**20.0±0.0** 16.1±8.2	**19.9±0.3** 16.1±8.2	**20.0±0.0** 16.0±8.4	**20.0±0.0** 14.4±9.1
1.0	**19.6±0.8** 16.2±7.8	**19.9±0.3** 16.1±8.2	**20.0±0.0** 16.0±8.4	**20.0±0.0** 16.0±8.4	**20.0±0.0** 15.7±8.3

(b) $\gamma = 0.9$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**20.0±0.0** 19.9±0.3	**19.9±0.3** 20.0±0.0	**20.0±0.0** 18.2±5.0	**20.0±0.0** 17.6±6.3	**20.0±0.0** 16.0±8.4
0.9	**19.9±0.3** 19.7±0.9	**20.0±0.0** 19.3±1.9	**19.9±0.3** 17.5±6.3	**20.0±0.0** 17.6±6.3	**20.0±0.0** 16.0±8.2
0.99	**20.0±0.0** 19.3±1.9	**20.0±0.0** 18.2±5.0	**20.0±0.0** 17.7±6.3	**20.0±0.0** 17.4±6.4	**20.0±0.0** 14.5±8.6
1.0	**19.9±0.3** 18.5±4.4	**20.0±0.0** 18.0±5.1	**20.0±0.0** 17.5±6.3	**20.0±0.0** 17.8±6.3	**20.0±0.0** 16.0±7.9

Table O.5: Number satisficing policies learnt for 1-bit memory agent agent out of 20 trials. Tables presented for $\gamma = 0.8$ and $\gamma = 0.9$. Values averaged across all test worlds. Top left bold figures CEQ(λ), bottom right SARSA(λ). Values shown are mean ± standard deviation

key:
CEQ(λ)
SARSA(λ)

(a) $\gamma = 0.99$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**20.0±0.0** 19.6±0.8	**19.9±0.3** 19.6±1.0	**20.0±0.0** 20.0±0.0	**20.0±0.0** 19.9±0.3	**20.0±0.0** 19.6±0.7
0.9	**20.0±0.0** 19.9±0.3	**20.0±0.0** 19.9±0.3	**20.0±0.0** 20.0±0.0	**20.0±0.0** 19.8±0.6	**20.0±0.0** 19.2±1.9
0.99	**20.0±0.0** 20.0±0.0	**20.0±0.0** 20.0±0.0	**20.0±0.0** 20.0±0.0	**20.0±0.0** 19.9±0.3	**20.0±0.0** 18.8±2.1
1.0	**20.0±0.0** 19.9±0.3	**20.0±0.0** 20.0±0.0	**20.0±0.0** 19.8±0.4	**20.0±0.0** 19.8±0.4	**20.0±0.0** 17.5±3.0

(b) $\gamma = 1.0$

λ \ α	0.01	0.02	0.05	0.1	0.3
0.8	**19.4±1.3** 19.6±1.0	**19.9±0.3** 19.7±0.9	**19.7±0.9** 19.9±0.3	**19.8±0.4** 19.8±0.4	**20.0±0.0** 18.2±3.8
0.9	**19.6±1.3** 19.8±0.4	**19.8±0.4** 19.7±0.5	**19.9±0.3** 19.9±0.3	**19.8±0.4** 19.6±1.0	**20.0±0.0** 18.3±3.7
0.99	**19.6±0.7** 19.9±0.3	**19.8±0.4** 20.0±0.0	**19.8±0.4** 20.0±0.0	**19.9±0.3** 19.2±1.9	**20.0±0.0** 18.1±2.4
1.0	**19.7±0.7** 17.7±5.9	**20.0±0.0** 17.4±5.8	**19.9±0.3** 16.6±6.2	**20.0±0.0** 16.2±5.3	**20.0±0.0** 12.2±8.2

Table O.6: Number satisficing policies learnt for 1-bit memory agent agent out of 20 trials. Tables presented for $\gamma = 0.99$ and $\gamma = 1.0$. Values averaged across all test worlds. Top left bold figures CEQ(λ), bottom right SARSA(λ). Values shown are mean ± standard deviation

(a) $\gamma = 0.8$

α' \ α	0.01	0.02	0.05	0.1	0.3
$\alpha/2$	2.0±6.3	2.0±6.3	2.2±6.3	2.9±6.2	3.7±6.4
$\alpha/10$	2.0±6.3	2.0±6.3	2.2±6.3	2.4±6.2	4.6±6.9

(b) $\gamma = 0.9$

α' \ α	0.01	0.02	0.05	0.1	0.3
$\alpha/2$	2.0±6.3	2.1±6.3	2.3±6.3	3.0±6.2	3.3±6.4
$\alpha/10$	2.1±6.3	2.4±6.3	3.2±6.2	2.8±6.3	5.0±7.2

(c) $\gamma = 0.99$

α' \ α	0.01	0.02	0.05	0.1	0.3
$\alpha/2$	10.9±9.0	6.2±8.0	3.7±6.3	3.8±6.8	3.6±6.4
$\alpha/10$	18.0±6.3	16.3±7.8	15.2±8.2	9.6±9.5	6.5±8.9

(d) $\gamma = 1.0$

α' \ α	0.01	0.02	0.05	0.1	0.3
$\alpha/2$	17.6±6.3	17.5±6.3	16.2±8.0	15.2±8.3	10.5±9.5
$\alpha/10$	17.7±6.3	17.4±6.4	17.4±6.4	17.8±5.7	16.0±8.4

Table O.7: Number satisficing policies learnt for Lion algorithm out of 20 trials. Averaged across all test worlds. Values shown are mean ± standard deviation

γ \ p	0.1	0.01	0.001
0.8	4.4±4.4	4.9±4.8	5.0±4.7
0.9	4.7±4.6	4.7±4.6	5.0±4.9
0.99	4.8±4.4	5.2±4.9	5.3±5.0
1.0	4.3±4.5	4.9±4.8	4.7±4.7

Table O.8: Number satisficing policies learnt for U-Tree out of 10 trials. Averaged across all test worlds. Values shown are mean ± standard deviation

Appendix P

Profile of Sensor Models for Simulated Robot Experiment

Figures P.1 and P.2 shows profiles of sonar and infra-red (IR) models used for the simulated robot in experiments in chapter 8.2. The Webots simulator provides a simple distance sensor model which allows the mean value returned by the sensor plus noise to be specified for various distances. The noise is specified as a percentage of the sensor's reading and provides an upper and lower bound on the values which can be are returned. Linear interpolation is used to determine the values at intermediate distances.

The values specified for the sonar sensor model are set out in table P.1. They are then interpreted by the simulator to give the profile shown in figure P.1. The profile is modelled on readings obtained from the real B21 robot "Gillespie". The sonar readings are clipped off (returning a maximum value of 32767) for objects closer than 4.2cm and further away than 6m. The noise on sonar readings between these two distances is quite small.

The values for the IR sensor model are set out in table P.2 which result the profile shown in figure P.2. This profile is also modelled on readings obtained from the real B21 robot "Gillespie". There is considerable noise on the readings returned by the IR sensors due to background illumination, etc. In specifying the IR model in VRML we

do not use the Webots built in sensor `type` tag of ”infra-red” as we do not want the readings returned to be dependent on the colour of the corridor walls.

Distance from sensor to object (metres)	Sensor reading	Noise ±%
0	32767	0
0.0419	32767	0
0.042	331	0
0.153	402	0
0.312	562	0.2
4	4246	0.03
6	6299	0.03
6.0001	32767	0

Table P.1: Sonar sensor model

Distance from sensor to object (metres)	Sensor reading	Noise ±%
0	97	0
0.155	93	32
0.335	23	30
0.6	9	60
1.002	5	60
2.057	2.5	20

Table P.2: Infra-red sensor model

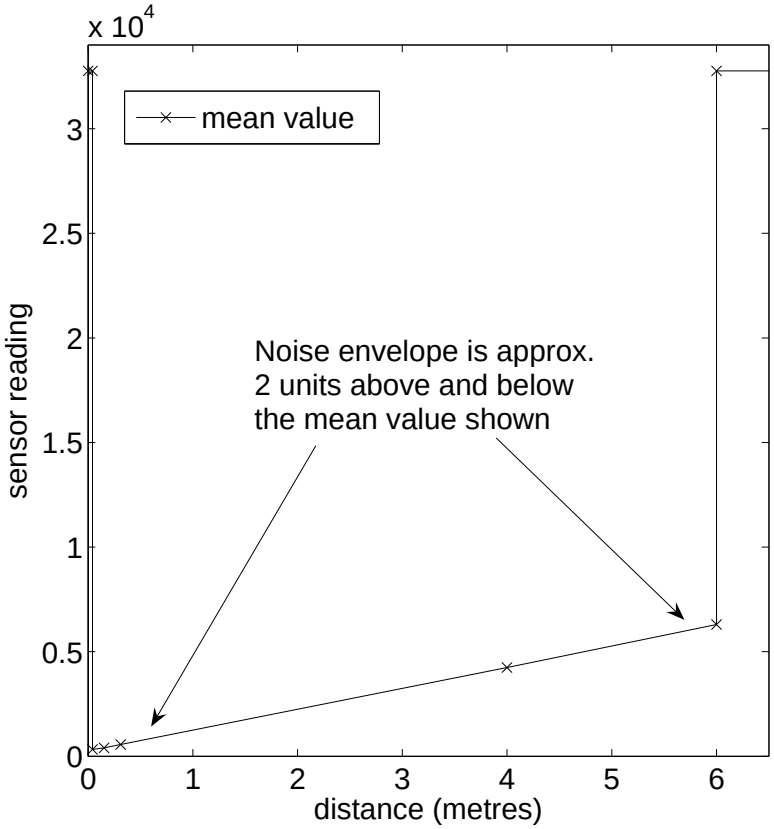


Figure P.1: Sonar sensor model profile

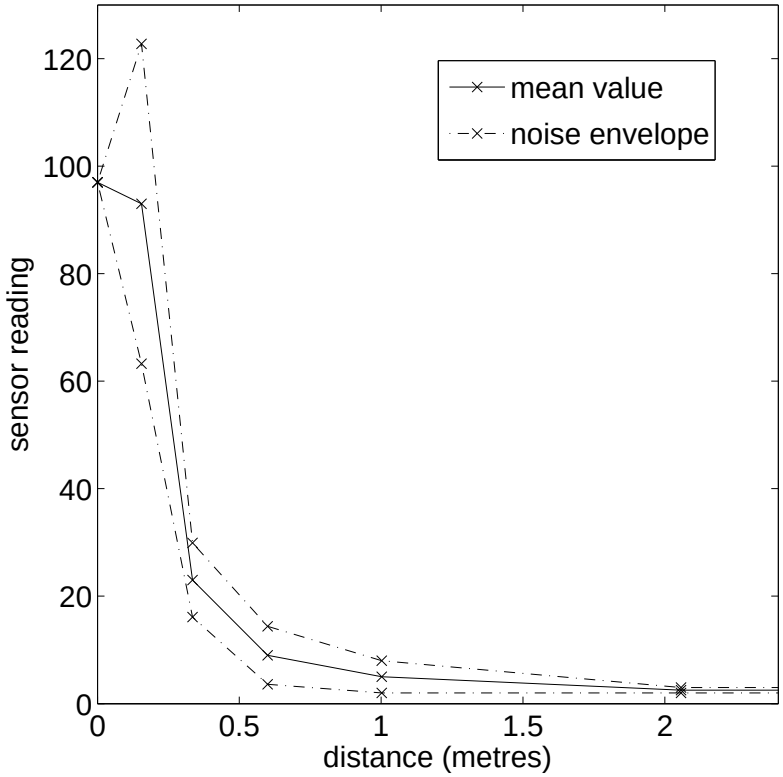


Figure P.2: Infra-red sensor model profile

Bibliography

Aberdeen, D. and Baxter, J. (2002). Scaling internal-state policy-gradient methods for POMDPs. In Sammut, C. and Hoffmann, A. G., editors, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 3–10. Morgan Kaufmann.

Aberdeen, D. A. (2003). *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, The Australian National University.

Allport, A. (1989). Visual attention. In Posner, M. I., editor, *Foundations of Cognitive Science*, pages 631–682. MIT Press.

Aloimonos, J., Bandyopadhyay, A., and Weiss, I. (1987). Active vision. In *First International Conference on Computer Vision*, pages 35–54. Also appears in International Journal of Computer Vision 1 (4), 1988, pages 333-356.

Aloimonos, Y. (1993). *Active Perception*. Lawrence Erlbaum Associates.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In Prieditis, A. and Russell, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pages 30–37. Morgan Kaufmann.

Baird, L. and Moore, A. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11 (NIPS'98)*, pages 968–974, Cambridge, MA. MIT Press.

Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, 76(8):996–1005.

Bakker, B., Linaker, F., and Schmidhuber, J. (2002). Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. In *Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, pages 938–943. IEEE.

Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 430–435. IEEE.

Balkenius, C. and Hulth, N. (1999). Attention as selection-for-action: A scheme for active perception. In *Third European Workshop on Advanced Mobile Robots, 1999 (EUROBOT '99)*, pages 113–119. IEEE.

Ballard, D. H. (1991). Animate vision. *Artificial Intellignce*, 48:57–86.

Basu, A. and Frazer, L. N. (1990). Rapid determination of the critical temperature in simulated annealing inversion. *Science*, 249(4975):1409–1412.

Baumgarter, P. and Stenz, G. (2004). IJCAR 2004 Tutorial T3, Instance based methods. http://www.mpi-inf.mpg.de/~baumgart/ijcar-tutorial/index.html. (contributions by H. Ganzinger, C. Tinelli and K. Korovin).

Blake, A. and Yuille, A., editors (1992). *Active Vision*. The MIT Press.

Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7 (NIPS'94)*, pages 369–376, Cambridge, MA. MIT Press.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23. also MIT AI Memo 864, September 1985.

Brooks, R. A. (1991). Intelligence without reason. In *Computers and Thought, Proceedings of the International Joint Conference on Artificial Intelligence*.

Chapman, D. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In Mylopoulos, J. and Reiter, R., editors, *Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, volume 2, pages 726–731. Morgan Kaufmann. Related document "Learning from delayed reinforcement in a complex domain", Technical Report TR-90-11, Teleos Research, December 1990.

Chrisman, L. (1992a). Planning for close-loop execution using partially observable Markovian decision processes. In *AAAI Spring Symposium Series: Control of Selective Perception*. Stanford University.

Chrisman, L. (1992b). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Tenth National Conference on Artificial Intelligence*, pages 183–188. AAAI/MIT Press.

Chrisman, L. and Simmons, R. G. (1991). Sensible planning: Focusing perceptual attention. In *Ninth National Conference on Artificial Intelligence*, pages 756–761. AAAI / The MIT Press.

Crook, P. A. and Hayes, G. (2003a). Active perception in navigation of partially observable grid worlds. In *Sixth European Workshop on Reinforcement Learning (EWRL-6)*. http://www.loria.fr/conferences/EWRL6/EWRL6.program.html.

Crook, P. A. and Hayes, G. (2003b). Could active perception aid navigation of partially observable grid worlds? In *Proceedings of the Fourteenth European Conference on Machine Learning (ECML 2003)*, volume 2837 of *Lecture Notes in Artificial Intelligence*, pages 72–83. Springer-Verlag.

Crook, P. A. and Hayes, G. (2003c). Learning in a state of confusion: Perceptual aliasing in grid world navigation. In *Towards Intelligent Mobile Robots 2003 (TIMR 2003), 4th British Conference on (Mobile) Robotics*, UWE, Bristol.

Crook, P. A., Marsland, S., Hayes, G., and Nehmzow, U. (2002). A tale of two filters – on-line novelty detection. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA 2002)*, pages 3894–3899. IEEE.

Cyberbotics (2005). Webots. http://www.cyberbotics.com/.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, 8(3/4):341–362.

Elman, J. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225.

Findlay, J. M. and Gilchrist, I. D. (2003). *Active Vision:The Psychology of Looking and Seeing*. Oxford University Press.

Floreano, D., Kato, T., Marocco, D., and Sauser, E. (2004). Coevolution of active vision and feature selection. *Biological Cybernetics*, 90(3):218–228.

Free Software Foundation (1991). GNU general public license, version 2. http://www.gnu.org/copyleft/gpl.html.

Goodrich, M. A., Stirling, W. C., and Frost, R. L. (1998). A theory of satisficing decisions and control. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 28(6):763–779.

Greiner, R. (1996). Palo: a probabilistic hill-climbing algorithm. *Artificial Intelligence*, 84(1-2):177–208.

Hartley, A. R. (2004). *Variable State Methods for Learning with Hidden State*. PhD thesis, School of Computer Science, The University of Birmingham.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7 (NIPS'94)*, pages 361–368, Denver, Colorado, USA. MIT Press.

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398.

James, M. R. and Singh, S. P. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. In Brodley, C. E., editor, *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*. ACM.

Kaelbling, L., Littmann, M., and Moore, A. (1996). Reinforcement learning: a survey. *Journal of AI Research*, pages 237–285.

Kaufman, B. E. (1990). A new theory of satisficing. *The Journal of Behavorial Economics*, 19(1):35–51.

Kirkpatrick, S., Jr., D. G., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Lanzi, P. L. (2000). Adaptive agents with reinforcement learning and internal memory. In Meyer, J.-A. et al., editors, *From Animals to Animats 6: Proceedings of the Sixth International Conference on the Simulation of Adaptive Behavior (SAB 2000)*, pages 333–342. The MIT Press, Cambridge, MA.

Lanzi, P. L. and Wilson, S. W. (2000). Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation*, 8(4):393–418.

Littman, M. L. (1992). An optimization-based categorization of reinforcement learning environments. In Meyer, J.-A., Roitblat, H. L., and Wilson, S. W., editors, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB'92)*, pages 262–270. The MIT Press/Bradford Books, Cambridge, MA.

Littman, M. L. (1994). Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB'94)*, pages 238–245. The MIT Press, Cambridge, MA.

Littman, M. L., Sutton, R. S., and Singh, S. P. (2001). Predictive representations of state. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 (NIPS'01)*, pages 1555–1561. MIT Press.

Loch, J. and Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, pages 323–331. Morgan Kaufmann, San Francisco, CA.

Marom, Y. (2003). *Influencing Robot Learning Through Design and Social Interactions — A Framework for Balancing Designer Effort with Active and Explicit Interactions*. PhD thesis, University of Edinburgh.

Marsland, S. (2001). *On-line Novelty Detection Through Self-Organisation, With Application to Inspection Robotics*. PhD thesis, Department of Computer Science, University of Manchester.

Marsland, S., Nehmzow, U., and Shapiro, J. (2000). Novelty detection on a mobile robot using habituation. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour (SAB 2000)*, pages 189 – 198. MIT Press.

Mataric, M. J. and Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83.

McCallum, A. K. (1993). Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning (ICML'93)*, pages 190–196. Morgan Kaufmann, San Francisco, CA.

McCallum, A. K. (1994). Instance-based state identification for reinforcement learning. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7 (NIPS'94)*, pages 377–384. MIT Press.

McCallum, A. K. (1995a). Instance-based utile distinctions for reinforcement learning with hidden state. In Prieditis, A. and Russell, S. J., editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pages 387–395. Morgan Kaufmann.

McCallum, A. K. (1995b). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Department of Computer Science, Rochester, New York.

McCallum, A. K. (1996). Learning to use selective attention and short-term memory in sequential tasks. In Maes, P. et al., editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB'96)*, pages 315–324. The MIT Press, Cambridge, MA.

McCallum, A. K. (2004). rlkit0.3.3. Personnel correspondence. Available from: http://www.cs.umass.edu/~mccallum/code-data.html.

McCallum, R. A. (1997). Efficient exploration in reinforcement learning with hidden state. In *AAAI Fall Symposium on "Model-directed Autonomous Systems"*.

McFadden, F. E., Peng, Y., and Reggia, J. A. (1993). Local conditions for phase transitions in neural networks with variable connection strengths. *Neural Networks*, 6(5):667–676.

Metropolis, N., Rosenbluth, A., Rosenbluth., M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 6(21):1087–1092.

Papadimitriou, C. H. and Tsitsiklis, J. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12:441–450.

Pendrith, M. D. and McGarity, M. J. (1997). An analysis of non-Markov automata games: Implications for reinforcement learning. Technical Report UNSW-CSE-TR-9702, The University of New South Wales.

Pendrith, M. D. and McGarity, M. J. (1998). An analysis of direct reinforcement learning in non-Markovian domains. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, pages 421–420. Morgan Kaufmann, San Francisco, CA.

Perkins, T. J. (2001). Action value based reinforcement learning for POMDPs. Technical Report UM-CS-2001-20, Department of Computer Science, University of Massachusetts Amherst.

Perkins, T. J. (2002a). *Lyapunov Methods for Safe Intelligent Agent Design*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

Perkins, T. J. (2002b). Reinforcement learning for POMDPs based on action values and stochastic optimization. In *Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 199–204, Menlo Park, CA. AAAI Press/MIT Press.

Perkins, T. J. and Pendrith, M. D. (2002). On the existence of fixed points for Q-learning and SARSA in partially observable domains. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 490–497.

Peshkin, L., Meuleau, N., and Kaelbling, L. (1999). Learning policies with external memory. In *Proceedings of the Sixteenth International Conference in Machine Learning (ICML'99)*, pages 307–314. Morgan Kaufmann.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rao, R. P. and Ballard, D. H. (1995). An active vision architecture based on iconic representations. *Artificial Intelligence*, 78:461–505.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (second edition)*. Prentice Hall.

Santos, S. (1995). Phase transitions in sparsely connected boltzmann machines. Series of Publications C C-1994-15, Department of Computer Science, University of Helsinki.

Selman, B. (1995). Stochastic search and phase transitions: AI meets physics. In *Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 998–1002.

Shatkay, H. and Kaelbling, L. P. (1997). Learning topological maps with weak local odometric information. In *IJCAI (2)*, pages 920–929.

Simon, H. A. (1955). A behavioral model of rational choice. *The Quarterly Journal of Economics*, 69(1):99–118.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, 63:129–138.

Singh, S. P. (2004). Myth: Rl is model free. http://neuromancer.eecs.umich.edu/cgi-bin/twiki/view/Main/RLmodelfree.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pages 284–292. Morgan Kaufmann, San Francisco, CA.

Singh, S. P., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

Singh, S. P., Littman, M. L., Jong, N. K., Pardoe, D., and Stone, P. (2003). Learning predictive state representations. In Fawcett, T. and Mishra, N., editors, *Proceedings of the Twentieth International on Machine Learning Conference (ICML 2003)*, pages 712–719. AAAI Press.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

Smart, W. D. and Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 903–910. Morgan Kaufmann, San Francisco, CA.

Stirling, W., Goodrich, M., and Packard, D. (2002). Satisficing equilibria: A non-classical theory of games and decisions. In *Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, volume 5, pages 305–328.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML'90)*, pages 216–224. Morgan Kaufmann, San Francisco, CA.

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

Sutton, R. S. and Barto, A. G. (2006). Errata and notes for *Reinforcement Learning: An Introduction*. http://www.cs.ualberta.ca/~sutton/book/errata.html.

Suzuki, M. and Floreano, D. (2006). Evolutionary active vision toward three dimensional landmark-navigation. In *From Animals to Animats 9: The Ninth International Conference on the Simulation of Adaptive Behavior (SAB'06)*, Lecture Notes in Artificial Intelligence, pages 263–273.

Tan, M. (1991a). Cost-sensitive reinforcement learning for adaptive classification and control. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 774–780. AAAI Press / The MIT Press.

Tan, M. (1991b). *Cost-Sensitive Robot Learning*. PhD thesis, School of Computer Science, CMU.

Tan, M. (1991c). Learning a cost-sensitive internal representation for reinforcement learning. In *Proceedings of the Eighth International Workshop on Machine Learning (ML91)*, pages 358–362. Morgan Kaufmann.

Tan, M. (1993). Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13:7–33.

Tan, M. and Schlimmer, J. C. (1990). Two case studies in cost-sensitive concept acquisition. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 854–860. AAAI Press / The MIT Press.

Thrun, S. and Schwartz, A. (1993). Issues in Using Function Approximation for Reinforcement Learning. In Mozer, M., Smolensky, P., Touretzky, D., Elman, J., and Weigend, A., editors, *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale, NJ. Lawrence Erlbaum.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge.

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

Whitehead, S. D. (1992). *Reinforcement Learning for the Adaptive Control of Perception and Action*. PhD thesis, University of Rochester, Department of Computer Science, Rochester, New York.

Whitehead, S. D. and Ballard, D. H. (1990a). Active perception and reinforcement learning. In Porter, B. W. and Mooney, R. J., editors, *Proceedings of the Seventh International Conference on Machine Learning*, pages 179–189. Morgan Kaufmann.

Whitehead, S. D. and Ballard, D. H. (1990b). Active perception and reinforcement learning. *Neural Computation*, 2(4):409–419.

Whitehead, S. D. and Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83.

Whitehead, S. D. and Lin, L.-J. (1995). Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306.

Wiering, M. and Schmidhuber, J. (1997). HQ-learning. *Adaptive Behavior*, 6(2).

Wilson, S. W. (1985). Knowledge growth in an artificial animal. In Grefenstette, J., editor, *First International Conference on Genetic Algorithms and Their Applications*, pages 16–23. Lawrence Erlbaum Assoc., Hillsdale, NJ.