Exploration and Inference in Learning from Reinforcement

Jeremy Wyatt

Ph.D.
University of Edinburgh
1997

For all my teachers

Abstract

Recently there has been a good deal of interest in using techniques developed for learning from reinforcement to guide learning in robots. Motivated by the desire to find better robot learning methods, this thesis presents a number of novel extensions to existing techniques for controlling exploration and inference in reinforcement learning.

First I distinguish between the well known exploration-exploitation trade-off and what I term exploration for future exploitation. It is argued that there are many tasks where it is more appropriate to maximise this latter measure. In particular it is appropriate when we want to employ learning algorithms as part of the process of designing a controller.

Informed by this insight I develop a number of novel measures of the agent's task knowledge. The first of these is a measure of the probability of a particular course of action being the optimal course of action. Estimators are developed for this measure for boolean and non-boolean processes. These are used in turn to develop probability matching techniques for guiding the exploration-exploitation trade-off. A proof is presented that one such method will converge in the limit to the optimal policy. Following this I develop an entropic measure of task-knowledge, based on the previous measure. An algorithm is then developed which guides exploration for future exploitation by selecting the action which is expected to lead to the greatest reduction in entropy. Empirical work is presented in which such methods outperform other exploration methods with respect to exploration for future exploitation on a variety of bandit tasks. It is also shown how the measures developed can be used in probabilistic verification of policies learned from reinforcement.

This thesis also explains how the exploration problem can be viewed as a form of the well-studied inference problem. As such the optimal solution can in principle be found using dynamic programming methods. Such methods for controlling exploration are *distal* methods, to be contrasted with existing *local* methods. In Chapter 6 some distal extensions of an existing exploration heuristic are presented. The chapter presents empirical results demonstrating the superior performance of such distal methods. This work highlights the problems involved in attempting to derive model-free distal exploration methods. It is concluded that in cases where exploration efficiency is of importance model-based methods are sometimes preferable to their model-free counterparts.

Acknowledgements

The single most significant acknowledgement must go to my excellent supervisors Gillian Hayes and John Hallam. They have given me an enormous amount of freedom to pursue the research goals I wanted, while never allowing me quite enough rope to actually hang myself. Gill in particular has devoted a good deal of time to helping me separate the promising ideas from the distinctly less than half-baked ones; as well as providing encouragement that has kept me going through the more arduous stages of the project. John has been an invaluable source of mathematical expertise and rigour, and his comments have prompted me to pursue avenues I would not otherwise have considered, and which have proven fruitful.

A research project is never carried out in complete isolation, and the mobile robotics lab in Edinburgh has proven a near-optimal setting in which to work. I have gained insight into the problems of robotics and reinforcement learning through conversations with a number of people. In particular I would like to single out Martin Westhead for patiently discussing the majority of my ideas with me at one time or another over the past three years. I have also benefited from discussions with Piak Chongstitvatana, Ashley Walker, Simon Perkins, John Demiris, Nuno Chagas, Sandra Gadhano, Edward Jones, Bill Chesters, Bridget Hallam, Janet Halperin, and Graham Deacon. Richard Reeve spent a number of hours assisting in the elucidation of obscure mathematical texts. I am grateful to John Demiris, Kal Perwaz and Ashley Walker for comments on thesis drafts and assistance with proof reading.

I informally attended a number of courses in the Department of Mathematics and Statistics, and would like to thank Colin Aitken and Ben Hambly for making them both enjoyable and informative. Thanks also go to Ben Hambly for discussions concerning work in Chapter 4. I would also like to thank Tim Colles for helping me sort out a large number of computer problems over the years, and for patiently answering some really dumb questions.

I enjoyed presenting parts of this thesis at a number of workshops and would like to acknowledge funding for some of these from the Department of Artificial Intelligence. This research was carried out while in receipt of an EPSRC research studentship.

Finally I would like to thank my family, who have been marvellously supportive throughout my studies. I am indebted to them.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Jeremy Wyatt
Edinburgh
April 7, 1997

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is concerned with improving existing methods for learning from reinforcement. In particular it will examine two problems that arise in systems which learn from such feedback. The first of these is the problem of how an agent should choose or influence the sequence of its experiences in order that the agent may converge rapidly and reliably to the correct behaviour. This is termed the *exploration* problem. The second problem is that of what inferences an agent may draw from a particular sequence of events (states, actions and rewards) and what methods are best used to draw such inferences. This is termed the *inference* problem. One conclusion of the work presented is that these are inextricably linked, and that the chosen solution to one dictates the form of the solution to the other.

It is important to state clearly the motivation for the work presented here. It is primarily a desire to find learning mechanisms which are appropriate for learning in embedded systems, and in particular for learning in mobile robots. Although the scope of the thesis does not include experiments on real robots, the problems of exploration and inference are considered interesting precisely because their solution is a prerequisite to the construction of practical robot learning systems. This introductory chapter therefore commences with an outline of the potential roles for robot learning (Sections 1.1 and 1.2). In this thesis I shall focus primarily on one of these: the use of learning mechanisms as part of the process of designing a robot controller. I therefore proceed to discuss the difficulties inherent in designing controllers for mobile robots (Section 1.3). It is argued that it is just these difficulties which make learning methods appealing. In particular I shall argue that adaptive controllers are attractive because they provide methods for designing controllers which are *verified in the limit*. All the methods examined in this thesis can be described as methods which learn from reinforcement. Hence in Sections 1.4—1.6 I give a brief overview of the state of research in this field; describe the relevant characteristics of learning from reinforcement; mention the known difficulties involved in their application to robotics; and demarcate more precisely the area for study. Finally (Section 1.8) I provide a road-map for the remainder of the thesis.

1.1 Mobile Robots

One of the most enduring images from Artificial Intelligence has been that of the autonomous mobile robot. There are no such robots yet. It is difficult, however to estimate how close we might be to such machines. Mobile robotics has undergone a sea-change during the past ten years. This has engendered a generation of robots which operate in changing, unstructured environments in real-time, rather than in the static, structured environments employed previously[1]. But at the time of writing even these robots are still autonomous in quite constrained ways for limited periods of time. Hardware is unreliable, extracting useful features from sensor information is non-trivial, and even given that these problems are soluble it is difficult to know how to tell a robot what to do so that it will behave reasonably across the range of situations it is likely to encounter.

In short it is hard to design good programs for mobile robots. There are many ways of expressing why this is the case. One way is to say that the problem is complex because when we design a mobile robot we are not merely designing an agent, but an interaction between an agent and an environment. The complexity of the interaction between even the simplest robot and the kind of physical environment it is likely to inhabit is great. Processes involving mobile robots are typically non-linear and stochastic. In the face of such systems the analytic approach of control theory breaks down. In consequence we currently have no coherent theory of mobile robot control.

[1] Compare the aims and approach of the Shakey project [58] with those of behaviour-based robotics [12].

Because of this, programming mobile robots is still something of a black art.

Learning is often cited as a panacea for these ills. It has yet to prove its worth. A learning robot is not better than a non-learning robot *per se*. Furthermore it is debatable as to whether any learning algorithms have yet found solutions to tasks too complex to hand code[2]. However, its ubiquity in the animal kingdom is indicative of its necessity for survival. It has been suggested by robot builders that learning confers two particular advantages in animals: adaptibility, and compactness of specification [13]. Do these advantages really carry over to robots? In order to answer this question we must address two points. First there are a number of important contextual differences in the use of learning in robots and animals. Second, there are many different applications of learning in robotics, for which different learning mechanisms may be appropriate. I now proceed to discuss the latter point.

1.2 Applications for Robot Learning

Learning is required by robots in a number of different circumstances. First, it is useful in situations where it is impossible to give the robot accurate information about its operating environment. This will be the case when robots are employed where it is impossible for people to go. The robot cannot possibly know enough about its environment prior to operation. Here the role of learning is in knowledge acquisition, learning a map or a model of parts of the environment. Second, it will be necessary when the task is of such a nature that the correct behaviour varies through time. This will be the case in robot tasks when the apparent properties of the environment change. This can happen in several ways: perhaps wear on moving parts such as gears alters the robot's dynamics; or a battery nearing the end of its life causes sensors to generate unusual readings. Some non-stationary problems are appropriately viewed as game-playing situations: when the policies[3] of other agents evolve, the consequences of a particular behaviour may change. Finally, learning techniques are frequently touted

[2] Interesting examples with respect to this point are Dean Pomerleau's work on ALVINN [61] and Sridhar Mahadevan and Jon Connell's work on Obelix [19].

[3] A policy is a mapping from the states of the world to the actions of the agent. Thus it may be viewed as the set of rules which determine an agent's immediate behaviour. I formally define the term in relation to Markov decision processes in Section 2.2.2.

along with evolutionary approaches as one way to simplify the task of designing robot controllers. By utilising learning techniques it is hoped that we can raise the level of abstraction of robot programming.

The first two circumstances in which learning might be used are similar in many respects to our common-sense notion of 'learning'. An agent operating entirely autonomously must learn quickly and (by definition) without human intervention. In such circumstances there is little room for error, and a trade-off exists between exploring[4] and exploiting[5] the environment. When using learning as part of the design process the criteria for success differ. The agent merely needs to maximise its *potential* performance on a task or its knowledge of an environment by the end of the learning period. We lose the need to exploit the environment because the agent's performance *during* the learning period does not matter. In addition the learning process need not be entirely autonomous. The designer may intervene in a number of ways. These could feasibly include driving the robot through some example solutions [17, 42]; decomposing the task into simpler tasks and then recomposing the solutions to these [49, 19]; or providing intermediate indicators of performance, i.e. telling the robot when it's doing well even if it has not yet achieved its goal [50]. By making assumptions about how the learning task is constrained we can modify general (weak) learning methods to be more specific (strong)[6].

Robot learning is helpful not only to robot designers. It is also of interest to workers in machine learning. This is because the latest wave of work in machine learning has been partly inspired by behaviours and mechanisms observed in animals [30, 27, 68, 80]. It is therefore only natural that we should seek to evaluate these algorithms in artificial agents which also operate in the real world. By subjecting our learning algorithms to the problems of sensor noise and ineffective action we increase our confidence in the methods that are seen to work. In short, robots can be seen in some sense as a litmus test for our learning algorithms. In addition the latest wave of work on robot learning

[4] Acting to gain information about how to achieve a goal.

[5] Acting on the basis of information gained to achieve a goal.

[6] Here weak methods are thought of as being methods which can be applied to a large number of problems in principle, but which typically take an unreasonable amount of time in practice (i.e. on tasks of a reasonable size). Strong methods make assumptions about the domain of application in order to achieve a better level of performance, and are consequently less general.

has come about through the confluence of several distinct research traditions[7]. Work in this field is now being carried out broadly within what is referred to as the embedded systems approach to artificial intelligence. It is therefore worth understanding the assumptions of this approach. I outline these in the next section, arguing that they lead naturally to regarding learning as a potentially important component of a general method for robot controller design.

1.3 Deriving Controllers for Embedded Systems

A new approach to artificial intelligence has emerged in the past decade in response to the demands of building intelligent controllers for agents operating in unstructured stochastic environments, and to the opportunities for studying the complex behaviour of artificial and biological systems. While there are several partially distinct schools within this broad approach, each with its own set of techniques, formalisms and dogma, they are unified by the belief that in building intelligent agents we ought to focus on the *interaction* between the agent and its environment rather than on either of these objects separately. This approach seeks to carry out research,

> Using principled characterisations of interactions between agents and their environments to guide explanation and design.[1]

The systems under scrutiny have been variously described as embedded systems [36], situated automata [65], situated agents [1], and embodied agents [7]. These names have been lent to the methodologies for their study.

A subset of researchers, albeit a significant one, chooses to describe this interaction between agent and environment in terms of dynamical systems [7, 76, 77]. To be more precise the agent environment interaction is viewed as a coupled dynamical system in which the outputs of the agent are transformed into inputs to the environment and the outputs of the environment are transformed into inputs to the agent. The

[7] The past ten years have seen a resurgence in approaches to cognition inspired by neuroscience. This bottom-up approach attracted many roboticists for both its philosophical assumptions and its supposed technical advantages. At the same time robot learning has embraced work from fields as diverse as psychology, control engineering, and ethology.

agent's outputs are its intended actions, and these are mapped to the environment's inputs by the agent's effectors. The environment's outputs are transformed into the agent's inputs by the agent's sensors. It may be the case that the inputs to the agent fail to constitute a complete description of the state of the environment. It is also important to note that the environment is not static. It changes continuously through time, rather than waiting for a control action from the agent. Finally the agent in an embedded system influences the future sequence of its inputs by its outputs at each stage. Such a system may therefore be regarded as to some extent being autonomous from its designer[8].

From the perspective of a designer of intelligent agents this view has several implications. First, in consequence of its incomplete perception, an agent will require mechanisms to assess the accuracy of its perception of the environment and to modify its perceptual function through time. Second, because the environment is not passive but changes between the agent's decisions in a manner influenced but not entirely determined by the agent, the agent has an upper bound on the time it takes to decide what to do next. Embedded systems with a learning component face an additional difficulty. In non-embedded learning systems the sequence of inputs is not chosen by the agent. Thus no sub-sequence of these inputs can affect succeeding inputs — although it can affect their interpretation. If the embedded agent is in any sense adaptive[9] then the outputs it selects can affect not just the next input but the entire sequence of inputs, and thereby the space of possible future internal states of the agent. Thus the final state of the agent can be strongly influenced by the manner in which it selects its earlier outputs. In designing an adaptive agent's mapping from inputs to outputs — termed its *policy* — we must take this into account.

These problems have traditionally been laid to one side by AI, which regarded them as mere complications to the more fundamental problems of intelligent behaviour. AI methods such as planning and heuristic search typically attempt to construct

[8] Questions of autonomy are fraught with difficulty and this claim for the autonomy of embedded agents is not authoritive, merely reflecting our common-sense notions of autonomy, which derives from the Greek word for self-governance.

[9] Here this means self-modifying.

open-loop[10] control policies given a very general model of a deterministic environment[11]. They are typically NP-complete [14], and are thus unsuitable for anything other than *off-line* construction of policies. Such methods are very general in that they can find correct policies for complex, non-linear environments. The kind of open-loop policies constructed are, however, reliable for control only under a number of strong assumptions. The model must be accurate, perception complete, the system to be controlled deterministic, and there must be no unmodelled disturbances.

Control theorists have investigated a complementary set of problems. Most of control theory is also concerned with the analysis of environment models. But these models take into account environments which change continuously through time regardless of whether an action is supplied by the controller. Control theory gives us methods for constructing *closed-loop* control policies from such models. Closed-loop control policies can be regarded as universal plans: specifying the correct control action whatever the state of the system. Closed-loop policies are therefore suitable for controlling stochastic systems. Because incomplete perception and unreliable effectors can be modelled as stochastic transitions [37], closed loop control can in principle also deal with these phenomena. Techniques for constructing such control policies include analytic methods for simple (linear) systems, and numerical methods such as dynamic programming for more complex (non-linear and stochastic) systems. Once again these techniques are for constructing control policies off-line, given accurate environment models[12].

The challenge of embedded systems is to build controllers for large, complex, stochastic environments for which we do not possess models, and in which the agent may be hampered by incomplete perception and unreliable effectors. There are currently a mixture of methods for building such controllers [11, 18, 2, 64, 48, 25, 16, 35]. These methods vary in their degree of rigour. Early agents designed using such approaches employed controllers that were entirely reactive [2, 11] these being the only alternative

[10] Open-loop control is control in which the sequence of control actions is determined off-line and during the execution of which the sequence of control actions is not modified on the basis of feedback.

[11] A representation of the state of the world in terms of a set of predicates e.g. $on(\text{red block}, \text{table})$, and a set of rewrite rules e.g. $put(X,Y) : on(X,Z) \rightarrow on(X,Y)$; is considerably more general than a state-transition model for a specific problem. The state-transition model is not explicitly specified, and must be generated by search using the operators. This makes such AI methods very general.

[12] In fact estimated models (which may be inaccurate) can be used to construct control policies under the *certainty-equivalence* assumption. I shall return to the issues involved later in the main text.

at the time to the unwieldy inference methods from AI noted previously. While robust for the simple task-environment niches tackled, these controllers proved difficult to design, and impossible to verify as being correct[13]. These limitations in turn motivated a number of attempts to develop a coherent formal theory of embedded systems [66, 7, 4, 24, 41]. One such body of theory underlies the work in this thesis. This is the theory of the prediction and control of finite Markov processes, and in particular of the use of dynamic programming based methods for these tasks.

When we are trying to design controllers for embedded systems we typically have no model of the process. Thus if we design a controller blind we have no way of formally verifying whether or not it is optimal or even legal. We have three options in the face of this problem. First we can abandon formal methods, relying on our *ad hoc* solutions on the argument that the problem is too hard to solve. Second we can design a controller blind and use it to derive a process model on-line, employing this model to verify whether the controller is optimal. This method depends on the accuracy of the model. The accuracy of the model depends in turn on the sequence of control actions taken. Thus the problem of designing the initial controller becomes one of designing a controller which builds an accurate model with maximal efficiency.

The third option is to use an adaptive controller, i.e. a system which uses its experience to modify the process model and the controller on-line. There have been important recent advances which extend the theory of dynamic programming to on-line methods for adaptive control [4]. These methods, which I outline in Chapter 2, use the controllers they construct to bias the data used to adjust the model. By this means effort is concentrated on parts of the state space more relevant to learning control.

However, interleaving control with the design of the controller raises a number of difficult issues. First there is the trade-off between *identification* and *control*[14]. This thesis will argue that there is in fact a third point between the two, which I shall term *identification for control*. In the first case we seek to maximise the accuracy of our model with maximal efficiency. In the second case we seek to use our model to

[13] The term correct has two possible meanings. We may follow [37] in using it to refer to optimal behaviour. Alternatively we may use it to refer to the set of legal behaviours. Here on I shall use only the terms legal and optimal to avoid confusion.

[14] see [4] p. 30

control the process with maximal efficacy. In the third category we seek to maximise our model's accuracy in the parts most relevant to distinguishing the optimal control policy. Doing this with maximal efficiency is different from optimising the trade-off between identification and control. Identification for control thus means identifying the optimal control policy as efficiently as possible.

There are both *model-based* [54] and *model-free* [89, 79] on-line methods which are guaranteed to converge to the optimal controller in the limit, under certain constraints. Hence they may be regarded as methods which are verified in the limit. Although convergence is therefore not as strong a property as verification at each stage, it is a stronger statement than we are typically able to make about any hand designed system. Thus adaptive controllers can also be seen as component techniques in formal methods for the design of controllers for embedded systems.

If the agent modifies its policy on the basis of its experience then it may, on our previous definition, be regarded as a learning system. Learning systems may be viewed as a subclass of embedded systems. A learning agent may receive or generate some form of feedback other than the current state of the environment. What forms may such feedback take?

Feedback may come in the form of a signal indicating correct behaviour, either explicitly, or in the form of an error signal. Such a learning system is a supervised learning system. Alternatively the agent may receive or generate a signal which is an evaluation of the agent's policy according to some Index of Performance (IoP). In this latter case it is not indicated what optimal behaviour is, merely how good or bad the agent's behaviour is according to some objective measure. The second approach is useful when the optimal behaviour is unknown in some or all circumstances, but when it *is* possible to specify the agent's goals and their relative importance. Such a feedback signal conveys less information. To learn the optimal behaviour the agent must try all actions and choose that which maximises performance according to the chosen metric. The advantage of the first approach is that the learning problem is simpler. The advantage of the second approach is that it is applicable to a broader range of problems.

1.4 Reinforcement Learning

Reinforcement learning tasks can be solved by learning systems of the second kind. The reinforcement learning problem is thus to be defined as a class of learning problems characterised by the type of feedback signal received (generated) rather than as a collection of learning algorithms. A reinforcement learning algorithm is any algorithm which learns from an Index of Performance.

Reinforcement learning is a trial and error approach to learning in which an agent operating in an environment learns how to achieve a task in that environment. The agent learns by adjusting its *policy*[15] on the basis of positive (or negative) feedback — termed reinforcement. This feedback takes the form of a scalar value generated by the agent each time step, high and low values corresponding to rewards and punishments respectively. The mapping from environment states and agent actions to reinforcement values is termed the *reinforcement function*. The agent converges to the behaviour maximising reinforcement (the *optimal policy*). In theory an appropriate reinforcement function[16] exists for all tasks, although finding such a function is typically hard [3].

It is worth noting that unlike supervised learning procedures the error signal does not indicate which behaviour is correct, merely how good or bad the current behaviour is relative to others. This means that in order to find the best action in each state the agent must try it at least once and thus in order to *guarantee* converging to the optimal policy the agent must try all actions in all states at least once[17]. In addition the feedback received is usually delayed, and hence most work on RL has focussed on solving the problem of assigning credit (or blame) to individual actions within a sequence leading to the receipt of reinforcement. This is known as the *temporal credit assignment* problem (TCA). It is to be distinguished from the better known *structural credit assignment* problem (SCA), which is concerned with assigning credit to features of a state in order to generalise across states.

[15] The mapping from environment states to agent actions.

[16] That is, one in which the policy maximising reinforcement corresponds to the behaviour which the *designer* considers optimal.

[17] An infinite number of times if the environment is stochastic.

1.5 Problems putting RL onto robots

Recently a lot of work has been done trying to put RL algorithms onto real robots and there have been a number of successful implementations to date [26, 37, 49, 19, 50, 57]. There are, however, a number of difficulties associated with RL methods *per se* and these are especially pertinent to the problem of using them with real robots. In short RL makes assumptions which do not apply to real world tasks [50].

First RL assumes that the environment as perceived by the agent is a Markov Decision Process (MDP). Informally this means that the agent only need know the current state of the process in order to predict its future behaviour[18]. If the agent does not have sufficient information to predict the future process behaviour[19] then what Whitehead terms *perceptual aliasing* [92] occurs. This is when an agent cannot distinguish between two states which are significantly different with respect to their behaviour under the same policy. New algorithms have been designed to cope with this phenomenon [45, 92] but are not guaranteed to converge to the optimal policy under such conditions.

The second major problem is that of slow convergence. Although in general deterministic MDPs can be solved efficiently [46] in the size of the state and action spaces; this has not yet been shown to extend to MDPs with stochastic transition functions [46]. Furthermore, because the number of possible states rises exponentially with the number of features in the environment, the time taken to solve MDPs rises rapidly as the complexity of the environment increases. Consequently, in stochastic environments with large feature (and hence state) spaces (e.g. a typical robot task) times to convergence are prohibitively long. There are necessarily two types of solution: either make your temporal credit assignment mechanism faster or make your temporal credit assignment problem simpler. The first approach has manifested itself in two ways: work on more effective trace mechanisms[20] [15, 59, 60, 81], and work on the use of generalisation methods [34, 51]. The second approach includes methods such as task

[18] In principle any process can be represented as an MDP because an abitrary amount of information about the history of the process can be included in the description of the current state, e.g. we need to know the velocity and acceleration of a ball in order to be able to calculate its trajectory.

[19] In the case of RL this means it does not have sufficient information to predict average return accurately.

[20] I discuss trace mechanisms in detail in Chapter 5.

decomposition [21, 38, 44, 19, 73, 33] and the construction of better reinforcement functions [50][21].

When applying RL to robot problems it will not suffice to employ trivial extensions of existing RL algorithms. *Tabula rasa* RL will not work in real robot problems. There are four separate goals which if attained would bring RL nearer to being usefully employable on robots. First it is necessary to find a principled manner in which the agent-guided exploration typical of RL can be seeded with knowledge the designer already possesses concerning the structure of the task and environment. It is interesting to think of this difficulty as being similar to the kind of problems expert-systems builders faced in the 1970s. People can be viewed as expert systems for moving around and manipulating objects in stochastic physical environments. It is difficult to express such procedural knowledge formally. The problem of programming robots might be viewed as hard because this difficulty constitutes a knowledge acquisition bottleneck.

Second, it would be useful to be able to reverse this process: to understand the output of our learning systems, i.e. to extract rules comprehensible by humans, so that future robots can be programmed more easily without the use of learning techniques. Third, we need to build a theory to explain the trade-offs between model-based and model-free methods[22]. Model-free methods use less computation but require more experience than model-based methods. The best method to use depends therefore on the relative costs of computation and experience. Finally, it is of practical importance to understand more about the process of decomposing a complex task into sub-tasks which can be more easily learned. This is a more general problem to which there are numerous solutions. Two classes of solution are decomposition by sensory and motor functions (classical) and decomposition by task structure (behavioural). What are the computational overheads and savings of each approach? There has been work on automating decomposition by task structure[21, 38, 74]. What are the properties of learners which employ such automated decomposition?

---

[21] Even if a reinforcement function is *appropriate* in the sense outlined in Section 1.4 it may still fail to give the agent sufficient feedback to ensure speedy learning, and so may not be a *good* reinforcement function.

[22] A model is some representation of the causal relationships in the interactions between agent and environment. Some methods for learning from reinforcement require a model, some do not.

1.6 Exploration in learning from the reinforcement

One of simplest systems in which the agent learns from reinforcement is a bandit problem. Suppose that there are a limited number of trials and the agent must learn to maximise performance over the period of these trials. At each step the agent has a trade-off. It can choose to act to gain information (explore), thus sacrificing short term reward for a greater certainty about which action is best later on; or it can choose to act to gain reward (exploit) and risk permanently following a policy which is suboptimal. This problem has occupied a large amount of time and effort over the past fifty years. There are methods for determining the exploration policy for a k-armed bandit which optimise expected total reward over n trials. The problem of finding the optimal exploration policy for multi-state problems is still open. A robot which is learning while in operation ideally needs to follow such an optimal policy. There has been a large quantity of work recently investigating heuristic methods for approximating this optimal trade-off between exploration and exploitation[84, 71, 83].

A robot which is using learning as part of its design process is not required to maximise total reward generated over the n trials given to learning a task. The objective for such a robot is to maximise its potential performance over these n trials. By potential performance I mean that the policy which the robot believes to be optimal should be as close to the optimal policy as possible. In other words the agent must maximise its knowledge about the task, particularly it must maximise its knowledge about how to maximise reinforcement. Exploration methods use heuristic estimators of how much they know about different parts of the environment. It would be a sensible next step to find principled estimators.

1.7 Assumptions and Methodology

The research presented in this thesis is predicated on the assumption that the problem of using RL techniques in the design of robot controllers is an interesting one and that the general technique is promising. It has been mentioned that the trade-off between model-based and model-free methods depends on the relative costs of computation and experience. The real cost of robot experience varies from domain to domain. Experience for learning low-level sensory-motor skills (e.g. hand-eye coordination) may be relatively cheap, whereas gaining data for safety-critical tasks (e.g. running a nuclear power-plant) may be very expensive. Thus there is no one learning technique appropriate for all domains. What sort of learning methods will be appropriate for the kinds of domains in which we are currently using our robots will transpire through experience.

Most of the algorithms presented in this thesis are verified experimentally in simulations of finite Markov processes. Issues that arise out of continuous state or action spaces; the use of generalisation mechanisms; non-Markovian environments, etc. fall beyond the scope of this thesis. It should be born in mind that the methods introduced are not necessarily extendible to any of the above cases.

1.8 Organisation

Chapter 2 of this thesis outlines the mathematical foundations for learning from reinforcement. This will include descriptions of finite Markov chains, and Markov decision processes; as well as a review of model-based and model-free techniques for their prediction and control. Chapter 3 continues with a review of the different forms of the exploration problem outlined briefly in Section 1.3. Various solutions to the exploration-exploitation problem are surveyed, including work from the literature on bandit problems (single state tasks); and that from artificial intelligence on heuristic methods for multi-state tasks. The exploration problem is explained within a coherent framework based on a categorisation of exploration techniques according to four criteria. Most importantly it is argued that the exploration problem is appropriately posed as an inference problem of a fundamentally similar form to the general problem of learning from reinforcement.

In Chapter 4 the focus is on controlling exploration in single state problems. First, a measure of the probability of each possible action being an optimal action is proposed. Following this estimators for such a measure are developed for boolean and non-boolean tasks. These are in turn used to develop simple algorithms for controlling exploration. A proof of convergence is presented for one of these algorithms. Second, an entropic measure of an agent's knowledge about its task is derived. This can be used to control exploration by selecting the action at each stage which is believed most likely to lead to a large reduction in entropy. A heuristic method based on the theory of hypothesis testing is also presented. All of these algorithms are evaluated empirically on a comprehensive range of tasks and their performance compared to other leading methods. In the non-boolean case, the methods introduced in this thesis are shown to outperform existing methods with regard to maximising convergence speed. This advantage becomes more pronounced as task complexity rises. Finally it is argued that the measures developed also have potential as model-free verification methods for controllers learned from reinforcement.

Chapter 5 compares the behaviour of model-based and model-free methods. First some simple variations of a well known model-free learning method called Q(λ)-learning are investigated. In particular the performance of different forms of this algorithm are examined under various exploration schemes. It is shown that the algorithm's estimates of long term reward quickly become corrupted when the agent deviates from the policy that it believes to be best. A known [89], but previously unimplemented modification to the algorithm is shown to perform much better in this respect. The performance of Q(λ)-learning is then compared empirically to that of a recent model-based algorithm, called prioritised sweeping. This comparative work informs the work of Chapter 6 on comparing the performance of model-based and model-free inference methods in guiding exploration.

Chapter 6 first compares model-free and model-based local methods for guiding exploration. It then investigates a general method for extending local exploration methods. Measures of immediate exploratory worth can be transformed into distal measures using dynamic programming based techniques. A model-free and a model-based distal extension are given for a counter-based exploration measure. Empirical work shows that while the model-based distal method is an improvement on its local counterpart; the model-free distal method fails to improve substantially on an equivalent model-free local rule. The chapter closes with a discussion of other possible distal extensions of existing exploration measures.

Finally Chapter 7 summarises the work carried out; and the conclusions to be drawn from the thesis work as a whole. Directions for future research are suggested.

Chapter 2

Foundations: Inference

We are concerned with finding solutions to problems involving an *agent* acting in an *environment*. There is a comprehensive body of mathematics for modelling agent-environment interactions. While this has been set out previously elsewhere (see [89, 37, 56, 10]) I summarise it using my own notation. The environment-agent interaction is modelled as an extension of a Markov decision process (MDP) in which the agent observes and controls the process. I shall then explain some of the computational problems involved in the prediction of Markov processes (MP) and the control of Markov decision processes. The prediction problem is the problem of inferring the long term behaviour of the process in terms of reward, and the control problem is that in which we must infer a mapping from states to actions which will maximise the agent's performance in terms of reward. I shall deal only with finite processes.

2.1 Markov Processes

In a discrete stochastic process we take the random variable X_t to denote the outcome at the t^{th} stage or time step. The stochastic process is defined by the set of random variables $\{X_t, t \in T\}$, where $T = \{0, 1, 2 \ldots\}$ is the set of possible times. The domain of X_t is the set of possible outcomes denoted $\mathcal{S} = \{s_1, s_2, \ldots s_{|\mathcal{S}|}\}$. In the general case the outcome at time t is dependent on the prior sequence of outcomes $x_0, x_1, \ldots x_{t-1}$:

$$\Pr(X_t = s_j | x_{t-1} \wedge x_{t-2} \wedge \ldots \wedge x_0) \tag{2.1}$$

We define q_{t-1} as any statement[1] defined on the domain of all possible sequences of outcomes prior to the time t. A process can be said to be an *independent process* if the outcome at each time t is independent of the outcomes at all prior stages, and thus of any statement q_{t-1}:

$$\Pr(X_t = s_j | q_{t-1}) = \Pr(X_t = s_j) \tag{2.2}$$

A Markov process weakens this independence assumption minimally by requiring that the outcome at time t is independent of all previous outcomes bar that at $t-1$.

$$\Pr(X_t = s_j | (X_{t-1} = s_i) \wedge q_{t-1}) = \Pr(X_t = s_j | X_{t-1} = s_i) \tag{2.3}$$

Equation 2.3 is known as the *Markov property*[2]. The probability $\Pr(X_t = s_j | X_{t-1} = s_i)$ can be regarded as a transition probability from the outcome s_i at $t-1$ to the outcome s_j at time t. I denote this transition by $s_i \rightsquigarrow s_j$. If the transition probabilities are independent of time then the process is a Markov Chain[3]. In this case the outcomes are referred to as the *states* of the process. I use the following shorthand to denote the probability of the transition from state $s_i \rightsquigarrow s_j$:

$$p_{ij} = \Pr(X_t = s_j | X_{t-1} = s_i) \tag{2.4}$$

Given the current state of a Markov chain and its transition probabilities we can predict its behaviour any number of steps into the future. The transition probabilities are represented in the form of a transition matrix, $\mathbf{W}$, the i,j^{th} element of which is p_{ij}. We also define a probability distribution across the starting states (i.e. when $t = 0$), denoted by the row vector $\mathbf{x}_0 = [\Pr(X_0 = s_1), \Pr(X_0 = s_2), \ldots \Pr(X_0 = s_n)]$, where n is the number of states. I denote the probability distribution across $\mathcal{S}$ at any time t by $\mathbf{x}_t$. Given $\mathbf{x}_0$ and $\mathbf{W}$, $\mathbf{x}_t$ can be expressed elegantly as the product:

---

[1] A statement is a function the value of which is either true or false.

[2] It is also necessary to assume that $(X_{t-1} = s_i) \wedge q_{t-1}$ is consistent.

[3] Here on we will consider Markov Chains as they constitute the majority of the applications and expression of the theory of learning from reinforcement to date.

$$\mathbf{x}_t = \mathbf{x}_0.(\mathbf{W})^t \tag{2.5}$$

The significance of this is that the study of the state of the process n steps into the future is the study of the n^{th} power of the transition matrix. It is worth noting for practical purposes that the notion of the future behaviour of the process being dependent solely on the current state of the process is a representational device. Processes whose future behaviour relies on knowing some or all of the process history can be made to satisfy the Markov property by including sufficient record of that history in the description of the current state. This may be expressed in the following manner. If the description of the state at time t is denoted by the column vector κ_t then we can denote the supplemented description of the current state by the concatenation of two vectors

$$\kappa'_t = [\kappa_t^T,\, f(\kappa_{t-1} \ldots \kappa_{t-k})^T]^T \tag{2.6}$$

where $f(.)$ is a function summarising the process history in the form of a new vector from states as far back in time as necessary, here k steps. In many cases the additional information may not add excessively to the length of the state description. If for example, we wish to predict the trajectory of a ball thrown through the air, then we use first and second order derivatives of position to summarise the history of the process necessary for the prediction of the future. If we use this information to control a process then we say that the controller has state. One of the primary problems with optimization methods relying on the Markov assumption is that we do not always know how much information it is necessary to supplement the description of the current state with. This is referred to in robotics as the question of how much state to include in the controller.

It can, however, be seen that this ability in principle to represent *any* stochastic process as a Markov process is a potentially powerful one. The inferential power gained is achieved by the way the Markov property separates the past and the future. The *necessary* history of the process is encapsulated in the description of the current state and this state completely determines future behaviour. I will now discuss Markov decision processes.

2.1.1 Markov Decision Processes

The finite state, discrete-time Markov chain model is extended by making the transition matrix at time t depend on an *action* a_t chosen at that time. The set of possible actions may vary from state to state so we write,

- $\mathcal{A}$ for the set of possible actions across all states
- $\mathcal{A}_x \subseteq \mathcal{A}$ for the set of possible actions in state x.

The transition probabilities that depend on the action chosen are denoted $p_{ij}(a)$, and there are now m transition matrices (where the size of the set $\mathcal{A}$ is m), one for each action: $\mathbf{W}_a = [p_{ij}(a)]$. If an action a is not possible in a particular state s_i, then $p_{ij}(a) = 0$. We may regard $\mathbf{W}$ as a function specified by these m transition matrices, mapping from all possible pairs of states and actions into a probability distribution across the set of states. I denote the transition from s_i to s_j following the selection of action a in state s_i by $s_i \overset{a}{\leadsto} s_j$. Finally we define a *reinforcement function* R which is defined as a mapping from the product of the state and action spaces into the set $\mathcal{R} \subseteq \Re$,

$$R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$$

At each observation, a *reward* is generated (reinforcement, index of performance)[4] dependent upon the state and the action performed. The random variable denoting the reward at time t is $R_t = R(x_t, a_t)$. The actual reward generated during the transition $x_t \overset{a_t}{\leadsto} x_{t+1}$ is r_t.

2.2 Modelling an Agent-Environment interaction

Following Kaelbling [37] the agent-environment interaction is modelled as an extension of a Markov decision process. We extend the model by allowing for incomplete percep-

[4] I use the terms reward and reinforcement interchangeably.

tion of the state of the process, and for the action at time t to depend on the sequence of previous states, actions and rewards. This additional complexity is represented as being entirely contained within the agent.

2.2.1 The Environment

The environment[5] is defined as the portion of a Markov decision process described by the triple $(\mathcal{S}, \mathcal{A}, \mathbf{W})$. At each step t the transition function $\mathbf{W}$ maps from its inputs: the state x_t and the action a_t, to an output: the state x_{t+1}. This state x_{t+1} becomes the input state at the next time step. The structure of the environment is shown in Figure 2.1.

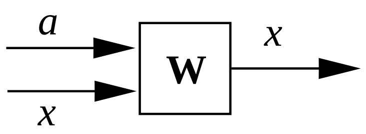


Figure 2.1: The environment.

2.2.2 The Agent

The agent is modelled by the tuple $(\mathcal{C}, P, R, B)$. $\mathcal{C} = \{c_1, c_2, \ldots c_{|\mathcal{C}|}\}$ is the set of possible perceived states[6]. The mapping from states to perceived states is determined by the agent's perception function, $P : \mathcal{S} \to \mathcal{C}$. If the agent's perception is complete then this mapping is one-to-one. If the agent's perception is incomplete then the mapping will be many-to-one. Incomplete perception can in principle simply be modelled by use of a stochastic transition function [37]. In practice however it can cause severe problems for an agent if two states are significantly different with respect to the future behaviour of the process. This problem is sometimes referred to as *perceptual aliasing* in the reinforcement learning literature and has been studied quite extensively [40, 45, 47, 71, 93, 91]. The process being controlled is then referred to as a Partially Observable

[5] In the RL literature it is common to include the reward function as part of of the environment. In the embedded systems literature, however, the reward function is seen as part of the agent. This thesis follows the latter convention.

[6] The elements of both $\mathcal{S}$ and $\mathcal{C}$ are numbered separately. The element c_i is the i^{th} element of $\mathcal{C}$, not the element of $\mathcal{C}$ corresponding to the i^{th} element of $\mathcal{S}$.

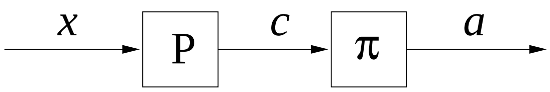


Figure 2.2: Structure of a non-learning agent.

Markov decision process. The set of applicable actions and the reward function are defined on the domain $\mathcal{C}$. So that $\mathcal{A}_c \subseteq \mathcal{A}$, and $R : \mathcal{C} \times \mathcal{A} \rightarrow \mathcal{R}$.

The structure of B, the behaviour, is more complex. There is a policy π which is a mapping from $\mathcal{C}$ into the set of probability distributions across $\mathcal{A}$. π is specified by a stochastic matrix, where the i, j^{th} element of π is,

$$\pi_{ij} = \Pr(a_j | c_i)$$

π says what to do in every possible perceived state. If the agent is non-learning and its policy is unchanging then π completely specifies B. The structure of a non-learning agent is shown in Figure 2.2. The agents we are interested in modify π as a function of their experience. The behaviour B of a *learning* agent is modelled by the tuple $(\psi, \eta, \sigma, \pi)$. In a learning agent the policy π_t depends on the entire previous history of perceived states, actions and rewards.

$$\pi_t = f([P(x_0), P(x_1), \ldots, P(x_{t-1})], [a_0, a_1, \ldots, a_{t-1}], [r_0, r_1, \ldots, r_{t-1}])$$

The experiences of the agent until time $t-1$ are summarised by the state of the agent σ_t. The state of the agent is itself updated each step, on the basis of the most recent experience and the previous state of the agent.

$$\sigma_{t+1} = \psi(P(x_t), a_t, r_t, \sigma_t)$$

At each stage the policy is a function of the state of the agent,

$$\pi_t = \eta(\sigma_t)$$

The structure of a generic learning agent as defined by these equations is shown in Figure 2.3. One consequence of this scheme is that it leads naturally to definitions of different kinds of learning agent. We may define a *policy-modifying* agent as any agent whose policy π varies as a function of its previous experiences. This should be distinguished from an agent which acquires knowledge by modifying σ, but does not alter π. I refer to such an agent as a *knowledge-modifying* agent[7]. This division is similar to the distinction between skill-acquisition and knowledge-acquistion in early work on machine learning [52]. I now discuss the nature of the agent's reward function.

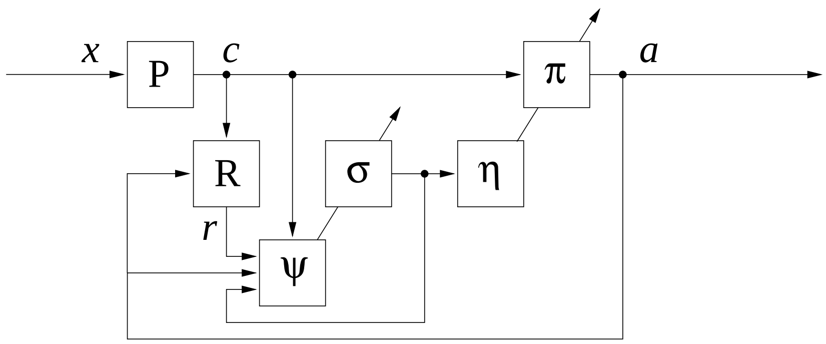


Figure 2.3: Structure of a learning agent.

Reward functions

The simplest possible reward function is when the set of possible rewards is Boolean, $\mathcal{R} = \{0, 1\}$. In this case the reward model is termed a P-model. Any problem with well-defined criteria for success and failure can be represented as a P-model. If for example the aim of a process is to track a set-point ω, and a certain magnitude of error ε is acceptable then taking 1 to be success[8] the reward at time t is

$$r_t = \begin{cases} 1 & \text{if } |\hat{\omega}_t - \omega| \leq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{\omega}_t$ is the system's approximation to the set point at time t. The minimal extension of this model is to allow any finite number of reward values in the interval $[0, 1]$. Such a model is termed a Q-model. Problems with real-valued rewards can be expressed in this form by means of normalisation and quantisation. The most general case is when the reward can take any real value in the interval $[0, 1]$. Such a reward

[7] Presumably this includes agents which forget as well as remember.

[8] Usually in Learning Automata Theory 0 is taken to be success and 1 to be failure. Since the convention in RL is to maximise reward I reverse this for convenience.

model is termed an S-model. By normalisation any problem with bounded reward can be expressed as an S-model.

The reward function merely specifies the reward generated at a particular instant. We can also construct a measure of reward over several time steps. Following Barto et al. [6], I refer to a measure of long-term reward as a measure of *return.* $\mathbf{R}_t$ is the random variable denoting return at time t. There are several measures of return investigated in the literature. All can be expressed in terms of a discount vector $G = \{\gamma_0, \gamma_1, \gamma_2, \ldots\}$, where the return at time t is,

$$\mathbf{R}_t = \sum_{k=0}^{\infty} \gamma_k r_{t+k} \tag{2.7}$$

The vector G may in principle be arbitrary, but in practice one of three *discount* schemes is used. The first is the *finite horizon* model of return, where the horizon is a finite number h of steps into the future,

$$\gamma_k = \begin{cases} 1 & \text{if } k \leq h \\ 0 & \text{if } k > h \end{cases}$$

This model has been studied extensively in the bandit literature, which will be reviewed in Section 3.3. Alternatively we may use the *average-reward* model [72, 75].

$$\gamma_k = \begin{cases} 1/h, & \text{if } k \leq h \\ 0 & \text{if } k > h \end{cases}$$

By far the most widely studied measure of return however, particularly within recent work on learning from delayed reinforcement, is an infinite horizon model termed the *geometric discount* model of return.

$$\gamma_k = \gamma^k, \text{ where } 0 \leq \gamma < 1$$

The value of γ chosen determines the relative weighting of short and long term rewards. As $\gamma \to 0$ short-term rewards become more important. When $\gamma = 0$ the only reward that matters is the immediate reward. As well as being attractive for its elegance this method has been shown to make certain problems in learning from reinforcement more tractable, e.g. bandit tasks. Here on I consider only the geometric discount model of return.

2.3 Optimal Policies

For control purposes it is obviously necessary to be able to order policies according to some index of performance. The reward models I have discussed can be used to derive just such an ordering on the set of possible policies. A policy π_1 is said to be at least as good as a policy π_2 if it has an expected return which is greater than or equal to that of policy π_2 in each of the possible initial states of the process:

$$E[\mathbf{R}(c_i)|\pi_1] \geq E[\mathbf{R}(c_i)|\pi_2],\, \forall c_i \in \mathcal{C}_0 \Leftrightarrow \pi_1 \succeq \pi_2 \tag{2.8}$$

where $a \succeq b$ is a preference operator, indicating that a is at least as good as b; and $\mathcal{C}_0 \in \mathcal{C}$ is the set of possible initial states of the process. An *optimal* policy is defined as any policy π^* which is as least as good as any other policy,

$$E[\mathbf{R}(c_i)|\pi^*] \geq E[\mathbf{R}(c_i)|\pi_j],\, \forall c_i \in \mathcal{C}_0,\, \pi_j \in \Pi \Leftrightarrow \pi^* \succeq \pi_j,\, \forall \pi_j \in \Pi \tag{2.9}$$

where Π is the set of possible policies. From the above equations it can be seen that the agent's optimal policy is defined with respect to the set of perceived states rather than the set of actual states. It is an important result that for both the finite horizon and geometrically discounted models of return there is at least one policy that is optimal for any Markov decision process. There may be more than one optimal policy, and so I will denote the set of optimal policies Π^*. It is also the case that any optimal policy is a stationary policy, i.e. it does not vary through time. The aim of any policy-modifying agent is to converge to such a policy. Our aim is to design learning agents which converge to an optimal policy quickly and reliably. The first step in designing such agents is to be able to estimate the expected return for a given policy in order that we may compare policies by Equations 2.8 and 2.9.

2.4 Prediction

Given a policy π, a transition function $\mathbf{W}$, and a reward function R for a Markov decision process we can calculate the expected return[9]. Before discussing this we need some additional notation. The random variable $R^\pi(s_i, n)$ denotes the reward received on the n-th step after starting in s_i and following policy π for n steps. The value $V^\pi(s_i)$ of policy π in state s_i is the expected value of the return under that policy. Hence $V^\pi(s_i)$ can be written,

$$V^\pi(s_i) = E[R^\pi(s_i, 0) + \gamma R^\pi(s_i, 1) + \gamma^2 R^\pi(s_i, 2) + \ldots + \gamma^n R^\pi(s_i, n) + \ldots]$$

This can be expressed recursively for all $s_i \in \mathcal{S}$,

$$V^\pi(s_i) = E[R^\pi(s_i, 0)] + \gamma \sum_j p_{ij}(\pi(s_i)) V^\pi(s_j), \; \forall s_i \in \mathcal{S} \tag{2.10}$$

The $V^\pi(s_i)$ for all s_i define the *value function* under the policy π. If we know $E[R^\pi(s_i, 0)]$ as well as $\mathbf{W}$ then the value function can be calculated off-line by solving this set of linear equations. Methods employing known $\mathbf{W}$ to derive the value function are termed *model based* methods for predicting the value of a policy. If the agent does not possess an analytic model it may employ a model of the state-transition function estimated from its own experience as it proceeds through the environment. Methods for estimating the value function using models of either kind are also often referred to as *indirect methods*. At each stage the model can be used to calculate the value of policies using exactly the same methods as for prefabricated models under the *certainty equivalence* assumption[10]. If the estimated state-transition model $\hat{\mathbf{W}}$ is inaccurate, however, then the accuracy of the estimated value function cannot be guaranteed. In future when I refer to a model I shall mean a model of the state transition

[9] Here on I assume that the agent's perception of the environment is complete. P is the identity function, and I denote the set of perceived states by $\mathcal{S}$. This assumption is made on the basis that the problems of perception fall beyond the scope of the remainder of the thesis. The additional notation in Section 2.2.2 may thus be considered a detour, and has been included merely to make it clear how this work as a whole fits into the embedded systems view which provides the motivation for the project, as described in Chapter 1.

[10] This is the assumption that the estimated transition probabilities are the same as the actual transition probabilities.

probabilities.

If the agent has neither $E[R^\pi(s_i, 0)]$ nor $\mathbf{W}$, then we can use a *direct* or *model-free* method for predicting the value of the policy on-line. Model-free methods build an estimate of the value function directly from their experience, i.e. from the sequence of perceived states and rewards generated. Thus they forego the information to be gained from estimating a state-transition model from experience[11]. A well-known and elegant model-free method for estimating expected return for Markov chains is Sutton's *temporal difference* method [79]. This works in roughly the following manner. Given that the transition $x_t \rightsquigarrow x_{t+1}$ occurs, the reward r_t is received during the course of this transition. At time t we have estimates of the value of each state, $\hat{V}_t(x_t)$ and $\hat{V}_t(x_{t+1})$. It turns out that a better estimate of $V(x_t)$ than $\hat{V}_t(x_t)$ can be provided by using

$$r_t + \gamma \hat{V}_t(x_{t+1})$$

The temporal difference is the difference between these two estimates,

$$r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$$

The basic temporal difference equation uses this to update the estimate of $V(x_t)$ each time the transition is made:

$$\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \alpha[r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)] \tag{2.11}$$

$0 < \alpha \leq 1$, is the learning rate. As $\alpha \to 1$ so $\hat{V}_{t+1}(x_t)$ depends more on $r_t + \gamma \hat{V}_t(x_{t+1})$ and less on $\hat{V}_t(x_t)$. α acts as a filter damping the variance in $r_t + \gamma \hat{V}_t(x_{t+1})$. As $t \to \infty$ the estimates $\hat{V}_t(x_t)$ are guaranteed to converge to $V_t(x_t)$ if α is declined toward 0 slowly enough [79]. Equation 2.11 forms the basis of the TD(0) algorithm which is specified in Figure 2.5.

Updating $\hat{V}(x)$ only on making a transition from x is comparatively inefficient. Because $V(x)$ depends to some extent on $V(y)$ for all y which can be reached eventually from

[11] The argument in favour of such methods is classically that models are expensive to maintain and use, and that model-free methods are therefore better when little computation is available per real world observation. I shall return to the issue of the trade-offs between model-based and model-free inference methods in Chapter 5.

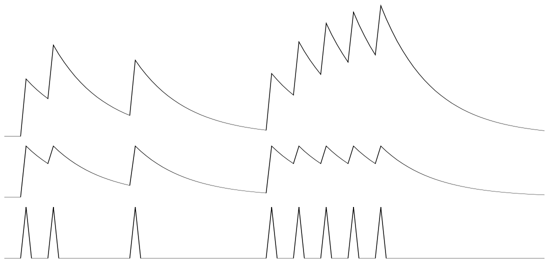


Figure 2.4: Behaviour of accumulating and replacing traces.

x, $\hat{V}(x)$ may not only be updated when it occurs, but also on the basis of the temporal difference for *any* subsequent transition. Using this insight Sutton generalised TD(0) by defining a class of prediction algorithms called TD(λ) (see Figure 2.5). In TD(λ) the extent to which a change in $\hat{V}_t(x_{t+1})$ is mirrored in other states is determined by the value of a function $\bar{e}$ called an eligibility trace, defined on the domain $\mathcal{S}$. It is included in the temporal difference update equation as follows,

$$\hat{V}_{t+1}(s) = \hat{V}_t(s) + \alpha[r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)]\bar{e}_t(s),\ \forall s \in \mathcal{S} \tag{2.12}$$

The value of $\bar{e}_t(s)$ is updated each transition for all $s \in \mathcal{S}$. There are two forms of the update equation. An *accumulating* trace updates the eligibility of a state using,

$$\bar{e}_t(s) = \begin{cases} \gamma\lambda\bar{e}_{t-1}(s) + 1 & \text{if } s = x_t \\ \gamma\lambda\bar{e}_{t-1}(s) & \text{otherwise} \end{cases} \tag{2.13}$$

A *replacing* trace update is defined by,

$$\bar{e}_t(s) = \begin{cases} 1 & \text{if } s = x_t \\ \gamma\lambda\bar{e}_{t-1}(s) & \text{otherwise} \end{cases} \tag{2.14}$$

Under both mechanisms the eligibility of a state decays away exponentially when the state is unvisited. Under an accumulating trace the eligibility is increased by a constant every time the state is visited, and under a replacing trace the eligibility is reset to a constant on each visit. The effect of each update rule on the eligibility of a state according to the frequency of visits is illustrated qualitatively in Figure 2.4.

An eligibility trace can be thought of as a short-term memory process initiated the first time a state is visited by an agent. The degree of activation depends on the

Algorithm 1 TD(λ) $0 < \alpha$

$t := 0$
$\hat{V}_t(x) := 0,\ \bar{e}_0(x) := 0,\ \forall x \in \mathcal{S}$
repeat
  observe the transition $x_t \rightsquigarrow x_{t+1}$
  update $\bar{e}(x)$ for all $x \in \mathcal{S}$ according to Eq. 2.13 or 2.14
  update $\hat{V}(x)$ for all $x \in \mathcal{S}$ according to
$$\hat{V}_{t+1}(x) := \hat{V}_t(x) + \alpha[r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)]\bar{e}_t(x)$$
  $t := t + 1$

Figure 2.5: The TD(λ) algorithm.

recency of the most recent visit and on the frequency of visits. Thus eligibility traces implement two heuristics, a *recency heuristic* and a *frequency heuristic*. These can stated informally as saying that reinforcement received now is probably caused to a greater degree by more recently and frequently occurring states than by less recently and frequently occurring states. Accumulating traces implement both these heuristics, while replacing traces implement just the recency heuristic [78]. The rate of decay of the trace is determined by $0 \leq \lambda \leq 1$. Hence the class of algorithms defined is referred to as TD(λ). If $\lambda = 0$ then Equation 2.12 simplifies to 2.11. If $\lambda = 1$ then the estimate of $V(x_t)$ ignores the estimated values $\hat{V}(x_{t+k})$ of any subsequent states and is based entirely on the actual rewards received at each step. If a perceptron is being used for structural credit assignment then using TD(1) makes its updates equivalent to those of the Widrow-Hoff rule. In general high values of λ give fast initial convergence to $V(x)$, and low λ values give low standard error. Better performance than by using fixed λ is therefore obtained by declining the value of λ from 1 to 0. Recently procedures for declining λ systematically have been investigated [81]. These work well for acyclic processes, but do not appear likely to extend to the more general cyclic case.

It has recently been shown that replacing and accumulating traces converge to different estimates of expected return, and that the replacing trace estimate is closely related to the certainty equivalent estimate of the value function [82]. In general the certainty equivalent estimate is the best approximation to the value function we can expect to obtain with an estimated model. Some early empirical results suggest that replacing traces used with TD(λ) outperform accumulating traces [82]. Although the TD(λ) algorithm applies strictly to estimating the value function for a Markov chain, any Markov decision process under a given policy can be modelled as a Markov chain.

In summary I have reviewed a model-based and a model-free method for the estimation of the value function for any given policy. It can be seen that in principle such techniques could be used to identify the optimal policy by estimating the value function for all possible policies. Such an approach would be grossly inefficient however, and there are much faster methods which I now discuss.

2.5 Control

As for the problem of predicting the value function there are both model-based and model-free methods for computing the optimal policy for a Markov decision process. If the agent has a model then we can employ one of a number of *dynamic programming* techniques to achieve this. I shall initially discuss two of these, *policy iteration* and *value iteration*. These algorithms are suitable for derivation of optimal controllers off-line given accurate process models. They can however, be extended to work on-line employing an adaptive process model.

Policy iteration commences with an arbitrary policy f and calculates the value function V^f by solving the set of linear equations given by Equation 2.10. It then improves this policy and recalculates the value function, repeating this process until the optimal policy is reached. The key lies in the way f is improved each iteration. Consider a particular state s_i. Suppose the agent takes any action a in state s_i and follows the policy f elsewhere. The value of this modification to policy f is termed the *action-value* of the action a in state s_i with respect to policy f [89], denoted $Q^f(s_i, a)$,

$$Q^f(s_i, a) = E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a) V^f(s_j) \tag{2.15}$$

We choose g as the policy which selects action a in state s_i and follows policy f in all

Algorithm 2 (POLICY ITERATION)

f := arbitrary policy
repeat

solve	$V^f(s_i) = E[R^f(s_i,0)] + \gamma \sum_j p_{ij}(f(s_i))V^f(s_j)$	$\forall s_i \in \mathcal{S}$
choose	$g(s_i) := \arg\max_a [Q^f(s_i,a)]$	$\forall s_i \in \mathcal{S}$
where	$Q^f(s_i,a) = E[R^a(s_i,0)] + \gamma \sum_j p_{ij}(a)V^f(s_j)]$	
let	$f(s_i) := g(s_i)$	$\forall s_i \in \mathcal{S}$

until there is no change in $f(s_i), \forall s_i \in \mathcal{S}$

Figure 2.6: The policy iteration algorithm.

other states, where:

$$a = \arg\max_a \{Q^f(s_i,a)\}$$

Because the set $\mathcal{A}_{s_i}$ includes the action chosen by policy f we know that g must be at least as good as f. We can carry out this improvement process in all states simultaneously, so that

$$V^g(x) \geq V^f(x),\ \forall x \in \mathcal{S}$$

Because a change in the policy at any single state can impact on the value function across the entire state space the entire value-function must then be recalculated for the new policy. This constitutes a single iteration of the policy iteration algorithm (see Figure 2.6). The algorithm iterates in this manner until no further improvements can be made to the policy in any state. Any such policy must be an optimal policy.

Policy iteration can be fast if the action space is quite small, as in such a case it will usually converge in few iterations. Even so recalculating the entire value function from scratch each iteration is expensive. Value iteration avoids this problem by solving the optimality equation for a series of geometrically discounted finite horizon problems defined on the same Markov decision process. Thus it only calculates the value function for the infinite horizon problem once. The value function for the optimal policy π^* is written V^*. Its solution for a problem with horizon $h = 0$ is given by,

$$V_0^*(s_i) = \max_a \{E[R^a(s_i, 0)]\}, \; \forall s_i \in \mathcal{S} \tag{2.16}$$

Using Bellman's optimality equation the solution for a finite horizon problem with $h = 1$ is given in terms of V_0^*. In general V_n^* is written in terms of V_{n-1}^*.

$$V_{n+1}^*(s_i) = \max_a \{E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a) V_n^*(s_j)\}, \; \forall s_i \in \mathcal{S} \tag{2.17}$$

As $n \to \infty$ so $V_n^* \to V^*$. The difficulty with value iteration is defining a stopping criterion for n. Recently this problem has been partly solved. A upper bound on the error of the final estimate can be expressed given the maximum error between the last two approximations to the value function [94]. Worst case performance can thus be improved to a measureable point arbitrarily close to the optimum [12]. The value iteration algorithm is given in Figure 2.7. There is some debate as to whether policy or value iteration converges faster for large problems.

There are a number of variations to value iteration as presented above, which is strictly termed *synchronous value iteration*. Synchronous refers simply to the fact that each iteration all the $V_n^*(x)$ are updated using the estimates $V_{n-1}^*(x)$. Gauss-Seidel iteration changes this by updating the states in a particular order. States updated later in the sequence use the already updated values for other states. Asynchronous value iteration relaxes the rules concerning the order in which the states are updated still further by allowing that an arbitrary subset of states is backed up synchronously each iteration. The set of states may vary in each iteration. In the completely asynchronous case this means that one state is updated each iteration. Both synchronous value-iteration and Gauss-Seidel iteration are guaranteed to converge to the optimal policy; asynchronous value-iteration is only guaranteed to converge if each state has a finite probability of being updated each iteration.

[12] In this thesis all approximations to the value function were calculated until the upper bound on the error was less than .001% of the minimum of the estimated value function.

Algorithm 3 (VALUE ITERATION) Choose ϵ so that $\frac{2\epsilon\gamma}{(1-\gamma)}$ is sufficiently small, where $\max_{\mathcal{S}}(|\hat{V}^*(s_i) - V^*(s_i)|) \leq \frac{2\epsilon\gamma}{(1-\gamma)}$.

$V_0 :=$ arbitrary bounded function
$n := 1$
repeat
  let $V_n^*(s_i) := max_a\{E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a) V_{n-1}^*(s_j)\}$ $\forall s_i \in \mathcal{S}$
until $\max_{\mathcal{S}}(|V_n^*(s_i) - V_{n-1}^*(s_i)|) < \epsilon$
let $\hat{V}^*(s_i) := V_n^*(s_i) \quad \forall s_i \in \mathcal{S}$

Figure 2.7: The value iteration algorithm.

The theory of asynchronous dynamic programming was originally developed in order to facilitate multi-processor implementation. More recently this theory has been used to show that dynamic programming algorithms can work on-line, interleaving estimation of the model, policy modification, and control [4]. A class of value-iteration algorithms, termed adaptive real-time dynamic programming (ARTDP) algorithms, can be defined (see Figure 2.8). The set $\mathcal{S}_t$ is the set of states whose values are to be updated at time t. It should be chosen so that the algorithm satisfies the conditions for the convergence of asynchronous value-iteration, and also that there is sufficient computation available to carry out all the updates. In addition because the agent is estimating the model on-line it must choose a suitable sequence of control actions so that it experiences each transition a sufficient number of times for the estimates $\hat{p}_{ij}$ to be good.

In a similar manner to Gauss-Seidel iteration, the choice of $\mathcal{S}_t$ at each stage can speed convergence to the optimal value function. There are a number of possible schemes. One of the simplest is to choose k additional members of $\mathcal{S}_t$ at random[13]. A good choice of $\mathcal{S}_t$ can speed convergence considerably. Moore and Atkeson have published a technique which can be seen as a slight variant of ARTDP. They call it Prioritised Sweeping. Its central feature is that it chooses sequences of updates that maximise convergence speed. Rather than carry out just one set of synchronous updates for each

[13] See [39] for a description of such an algorithm. They refer to it as a Dyna method, although it is rather different from Sutton's original method of the same name [67].

Algorithm 4 (ADAPTIVE REAL-TIME VALUE ITERATION)
$\hat{Q}_t(x,a) = \bar{r}(x,a) + \gamma \sum_{y \in \mathcal{S}} \hat{p}_{xy}(a)\hat{V}_t^*(y)$. explore is a function mapping from estimated Q-values to a probability distribution across actions. $\bar{r}(i,a)$ is an estimate of $E[R^a(i,0)]$.

$t := 0$
$\hat{V}_t^* :=$ arbitrary bounded function
observe x_t
repeat
    choose a_t from explore$_a(\hat{Q}_t(x_t, a))$
    observe the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$
    update $\hat{p}_{x_t x_{t+1}}(a_t)$ and $\bar{r}(x_t, a_t)$
    for some $\mathcal{S}_t \subseteq \mathcal{S}$ such that $x_t \in \mathcal{S}_t$ update
        $\hat{V}_{t+1}^*(i) := \max_a\{\bar{r}(i,a) + \gamma \sum_{j \in \mathcal{S}} \hat{p}_{ij}(a)\hat{V}_t^*(j)\}$ $\forall i \in \mathcal{S}_t$
        $\hat{V}_{t+1}^*(i) := \hat{V}_t^*(i)$ $\forall i \notin \mathcal{S}_t$
    $t := t + 1$

Figure 2.8: The adaptive real-time value iteration algorithm.

real-time experience, their algorithm makes a sequence of updates like Gauss-Seidel iteration. Unlike Gauss-Seidel iteration the sequence varies each real-time experience. This can be seen in terms of asynchronous value-iteration, by thinking of each update as a separate iteration with $\mathcal{S}_t$ being a singleton set, there being up to k iterations for each real experience. The key is the sequence in which the states are updated. They are prioritised according to how much the value-function is changing in each part of the state-space. The method Moore and Atkeson use is to keep a queue of states indexed by their priorities. States with the highest priorities are processed first. When $\hat{V}_{t+1}^*(j)$ has been calculated using Equation 2.17 any state i that has immediately preceded j is added to the queue. Its priority is given by the change in the value function at j:

$$\Psi(i) = \max_a\{\hat{p}_{ij}(a)|\hat{V}_{t+1}^*(j) - \hat{V}_t^*(j)|\}$$

Initially the priority queue is empty. When a transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$ occurs the state x_t is added to the queue with a priority an arbitrary amount larger than the current largest priority. When the updating of states begins x_t is the first state to be updated. Hence

Algorithm 5 (PRIORITISED SWEEPING)

$\hat{Q}_t(x,a) = \bar{r}(x,a) + \gamma \sum_{y \in \mathcal{S}} \hat{p}_{xy}(a)\hat{V}_t^*(y)$. explore is a function mapping from estimated Q-values to a probability distribution across actions. $\mathcal{P}$ is the priority queue. $\Psi(i)$ is the priority of the state i. $0 < \varepsilon$. k is a positive integer. $\bar{r}(i,a)$ is an estimate of $E[R^a(i,0)]$.

$t := 0$, $\mathcal{P} := \emptyset$
$\hat{V}_t^* :=$ arbitrary bounded function
observe x_t
repeat
  choose a_t from explore$_a(\hat{Q}_t(x_t,a))$
  observe the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$ and reward r_t
  update $\hat{p}_{x_t x_{t+1}}(a_t)$ and $\bar{r}(x_t, a_t)$
  add x_t to $\mathcal{P}$, $\Psi(x_t) := \max_{s \in \mathcal{P}}\{\Psi(s)\} + \varepsilon$
  repeat k times or until $\mathcal{P} = \emptyset$
    for $i := \arg\max_{s \in \mathcal{P}}\{\Psi(s)\}$
    $\hat{V}_{t+1}^*(i) := \max_a\{\bar{r}(i,a) + \gamma \sum_{j \in \mathcal{S}} \hat{p}_{ij}(a)\hat{V}_t^*(j)\}$
    $\Delta := |\hat{V}_{t+1}^*(i) - \hat{V}_t^*(i)|$
    for each (i', a) such that $\hat{p}_{i'i}(a) > 0$
      if $\hat{p}_{i'i}(a)\Psi(i) > \varepsilon$
      add i' to $\mathcal{P}$, $\Psi(i') := \max\{\Psi(i'), \hat{p}_{i'i}(a)\Delta\}$
    remove i from $\mathcal{P}$
  $t := t + 1$

Figure 2.9: The prioritised sweeping algorithm.

its predecessors are added to the queue. If the change in the value function at x_t is large then its predecessors will have high priorities. They in turn will be updated and their predecessors added to the queue. Hence suprising events relevant to the task are propagated through the state space in an efficient manner. The Prioritised Sweeping algorithm for control is given in Figure 2.9[14].

There are a number of model-free policy-modification algorithms; the earliest of these [78, 32] used either the TD(λ) algorithm, or temporal difference methods similar to

[14] Moore and Atkeson give a specific solution to the exploration problem using an exploration bonus method. I specify the algorithm using an arbitrary exploration method.

Algorithm 6 Q(λ)-LEARNING

$\hat{V}^*(x) = \max_a \hat{Q}(x,a)$. $0 \leq \lambda \leq 1$. ε'_t and ε_t are error signals. explore is a function mapping from estimated Q-values to a probability distribution across actions.

$t := 0$
$\hat{Q}(x,a) := 0$ and $\bar{e}_t(x,a) := 0, \forall x, a$
observe x_t
repeat
  choose a_t from explore$_a(\hat{Q}_t(x_t,a))$
  observe the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$
  $\varepsilon'_t := r_t + \gamma \hat{V}^*_t(x_{t+1}) - \hat{Q}_t(x_t,a_t)$
  $\varepsilon_t := r_t + \gamma \hat{V}^*_t(x_{t+1}) - \hat{V}^*_t(x_t)$
  update $\bar{e}(x,a)$ for all $x \in \mathcal{S}, a \in \mathcal{A}$ according to Eq. 2.19 or 2.20
  update $\hat{Q}_{t+1}(x,a)$ for all $x \in \mathcal{S}, a \in \mathcal{A}$ using
    $\hat{Q}_{t+1}(x_t,a_t) := \hat{Q}_t(x_t,a_t) + \alpha\varepsilon'_t\bar{e}_t(x_t,a_t)$
    $\hat{Q}_{t+1}(x,a) := \hat{Q}_t(x,a) + \alpha\varepsilon_t\bar{e}_t(x,a)$ for all $\hat{Q}(x,a)$ except $\hat{Q}(x_t,a_t)$
  t:=t+1

Figure 2.10: The $Q(\lambda)$ algorithm.

it, to convert a delayed reward signal into a heuristic reward signal (the $\hat{V}^\pi(x_t)$ under the current policy π). The heuristic reward signal is then fed to any algorithm which modifies its policy on the basis of immediate reward, in place of the immediate reward signal r_t. Thus the class of algorithms compatible with this method is large. The policy of the agent is modified using the estimate $\hat{V}^{\pi_t}(x_t)$. In the next step the TD(λ) algorithm estimates V with respect to a different policy π_{t+1}. Thus the value function is changing as the policy changes. Examples of policy-modification algorithms based on the TD(λ) algorithm include [37, 43, 78]. To my knowledge there are currently no proofs for the convergence of any such policy-modification algorithms.

A model-free method which is guaranteed to converge is Q-learning [89]. This is a model-free approximation to adaptive real-time value iteration. The primary structural difference between Q-learning and methods based on the TD(λ) algorithm is that whereas the latter maintain estimates of the values of states under the current policy, Q-learning maintains estimates of action-values. It adjusts these estimates each step using a temporal difference mechanism,

$$\hat{Q}_{t+1}(x_t, a_t) = (1-\alpha)\hat{Q}_t(x_t, a_t) + \alpha[r_t + \gamma \max_a\{\hat{Q}_t(x_{t+1}, a)\}] \tag{2.18}$$

where $0 < \alpha < 1$ is the learning rate. The update equation is fundamentally of the same form as the value iteration update, replacing the estimates $\hat{p}_{ij}$ with α. Q-learning is guaranteed to converge asymptotically given that each state-action pair is tried infinitely often; that $\sum_{t=0}^{\infty} \alpha_t = \infty$; that $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$; and that $\alpha_t > 0$, $\forall t \in T$ [90]. The other nice property of Q-learning is that the estimates of the Q-values are independent of the policy followed by the agent, the consequence of this being that the agent may deviate from the optimal policy at any stage while still constructing unbiased estimates of the Q-values.

Q-learning may also be extended to take advantage of eligibility traces. Q(λ) [60] contains one-step Q-learning as a special case ($\lambda = 0$) and so I give the full algorithm for this generalised version (Figure 2.10). Convergence has only been proved for Q(λ) with $\lambda = 0$. The eligibility traces used in Q(λ) are necessarily defined over the domain formed by the Cartesian product $\mathcal{S} \times \mathcal{A}$. The update equations are, however, fundamentally the same:

$$\bar{e}_t(x,a) = \begin{cases} \gamma\lambda\bar{e}_{t-1}(x,a) + 1 & \text{if } x = x_t \text{ and } a = a_t \\ \gamma\lambda\bar{e}_{t-1}(x,a) & \text{otherwise} \end{cases} \tag{2.19}$$

$$\bar{e}_t(x,a) = \begin{cases} 1 & \text{if } x = x_t \text{ and } a = a_t \\ \gamma\lambda\bar{e}_{t-1}(x,a) & \text{otherwise} \end{cases} \tag{2.20}$$

2.6 Summary

In this chapter I have introduced the notation employed in this thesis for finite Markov chains and finite Markov decision processes. I have also reviewed a number of model-free and model-based techniques for their prediction and control. We will return to the subject of inference proper in Chapter 5, where I compare the performance of a model-based and a model-free inference method. We now turn to the problem of exploration. In the next chapter it will be explained why this problem is important and I shall discuss its composition. I will then review techniques for controlling exploration developed within the artificial intelligence and statistics communities, presenting these techniques within the mathematical framework introduced above.

Chapter 3

Foundations: Exploration

3.1 Introduction

The exploration problem arises when an embedded adaptive agent chooses or influences its sequence of future experiences. Because the agent is adaptive these choices in turn influence the sequence of its future internal states. A good sequence of experiences can cause an adaptive agent to converge to a stable and useful policy. A bad sequence may cause an agent to converge to an unstable or a poor policy. Designing mechanisms that ensure the sequence of experiences is a good one is an important problem in building robust adaptive agents. Although this task arises in any embedded learning system, it is particularly well-studied in the case of systems which learn from reinforcement.

In this chapter I argue that the exploration problem is in fact three separate but related problems (Section 3.2). This analysis precedes a review of the current techniques for solving one of these sub-problems, that commonly referred to as optimising the *exploration-exploitation trade-off*. I first explain the form of this problem for the simplest case: that of a decision process with a single state— commonly termed a bandit task. There are two methods for its solution, one tractable and the other intractable. Both methods are explained (Section 3.3). In Section 3.4 I briefly explain how the multi-state problem can be seen as an extension of the single state case. I then proceed to provide a coherent framework for exploration based upon the reinforcement learning framework outlined in Chapter 2. In this framework methods for controlling exploration are categorised according to four criteria. Furthermore a measure of immediate exploratory worth can be seen as being similar to a reward function. Using this idea the problem of behaving optimally with regard to a particular exploration measure can be posed as the problem of solving Bellmann's optimality equation (Section 3.4.1). A range of heuristic exploration measures are then reviewed (Section 3.4.2). Finally I show how the notion of an exploration bonus fits into this framework (Section 3.4.6).

3.2 Thinking about exploration

The exploration problem for embedded learning systems is actually a set of three related problems. First there are agents which seek to *maximise their knowledge of their interaction with the environment*, without reference to a reward function. In control theory this is the problem of *system identification* under perturbation of the plant by the controller. An example of such a system is a robot which has to map unknown terrain on a planet or under the sea. Secondly there are agents which seek to *maximise their performance on a task over a limited period.* This is similar to the classic bandit problem in statistics. In the reinforcement learning community it is known as the problem of optimising the *exploration-exploitation trade-off.* Any agent which learns throughout its lifetime seeks to solve this problem. Such agents include animals which forage for food; medical researchers seeking to test a new treatment while maximising the total number of cures; and robots which seek to improve their behaviour on-line. Finally there are agents which seek to *maximise their knowledge over a given learning interval about how to perform a particular task.* Any learning task in which the reward function being learned differs from the actual cost during the learning period falls into this category. Examples might include training a robot to carry cannisters of toxic waste, or learning a road following behaviour for a self-driving car. In each case a physical simulation can be used for practice in which the costs of dropping a cannister or driving into a wall are rather different from the corresponding real life costs. The agent, however, can learn the task using the physical simulation with respect to the real-life cost-function. In such a case it is preferable to ensure a controller that is as near optimal as possible. Learning the optimal controller to as high a degree of certainty as possible during the available learning period is not the same as optimising performance over that learning period. This category has not, to my knowledge, been investigated in the field of learning from reinforcement. I shall refer to it as the problem of *identification for control* in the case of general learning agents, and as the problem of *exploration for future exploitation* in the case of agents which learn from reinforcement.

In this thesis we are interested in the second and third problems outlined above. As background to this I now review the body of work in reinforcement learning on the optimisation of the exploration-exploitation trade-off. This is one of the most interesting outstanding problems in learning from delayed reinforcement in multi-state environments. First it is interesting because although we possess a number of useful heuristics for controlling the exploration-exploitation trade-off in tasks with multiple states and delayed reinforcement we still have no rigorous theory. Secondly it is interesting because it can be regarded as an extension of a bandit problem — which has a single state and delayed reinforcement. Lastly it is interesting because in practice the exploration technique used can have as much impact on convergence time as the inference method. Inference and exploration, although formally separate issues, are bound together in practice when we employ cheaper approximations to dynamic programming methods rather than carry out the full dynamic programming calculation to obtain the certainty-equivalent value-function. I first review the bandit literature, and then discuss the general multi-state problem as an extension of this. Finally I outline a number of existing heuristic solutions to the multi-state problem.

3.3 The single state case: bandit tasks

Bandit problems are among the simplest problems in learning from reinforcement. This is not to say that they are trivial, having occupied statisticians for over forty years [8, 9, 28, 29, 63]. Bandit problems are important to understand because they neatly encapsulate the essence of the exploration-exploitation problem. The bandit literature, however, employs a terminology different from that of reinforcement learning. Thus in the interests of clarity and completeness I shall now describe bandit problems within the reinforcement learning framework outlined previously.

A bandit problem concerns the behaviour of a family of bandit processes. A family of bandit processes is a collection of $k \geq 2$ independent stochastic processes (the

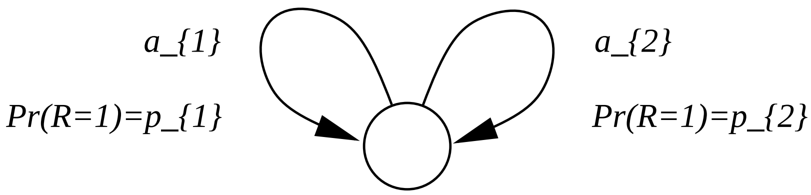


Figure 3.1: A family of two alternative boolean bandit processes specified by a finite-state (Mealy) machine. The inputs are the actions a_i and the outputs are the probabilities of success p_i. These need to be evaluated to generate the observed reward each step. The transition function is the identity function regardless of the action chosen.

bandits) each of which is associated with an arm (or action). Such a family may be represented as being defined by a finite-state machine consisting of a single state and k inputs (the actions). The i^{th} action is denoted a_i and is associated with an output specified by the probability density function (pdf) for the random variable R^i representing immediate reward. The bandits we will examine generate either boolean or real-valued reinforcement[1]. At each step t the agent selects an action a_t and the resulting reinforcement r_t is observed. The agent's task as classically formulated is to maximise expected return according to some discount function. The agent commences with little or no knowledge as to the values of the parameters of the distributions of the R^i. Typically the form of the distributions of the R^i is known. Thus there is a choice at each step between acting to improve estimates of these parameters and acting to gain reinforcement based on these estimates. The bandit *problem* is to calculate a strategy for selecting actions that optimises this trade-off. The actual optimal strategy varies depending on the trade-off between short and long-term reward determined by the discount function.

Any strategy for selecting actions at each stage of a bandit process requires a history of previous selections and observations, or some summary of such. Thus although the family of bandit processes may be usefully depicted as a machine composed of a single state, any strategy for selecting actions is to be regarded as a policy defined on a state-space consisting of all the possible sequences of outcomes for the family of processes. These states are referred to as *belief-states*. It should be clear that the set of belief-

[1] In the bandit literature the term bandit strictly refers to a bandit generating boolean rewards (see [28] p. 18). I will depart from this convention slightly, referring to such bandits as *boolean bandits* and to bandits generating real-valued rewards as *real-valued bandits*.

states grows exponentially as a function of the length of the horizon and the number of arms. It follows that the calculation of an optimal strategy is not a computationally trivial task. Bandit problems have been studied under two measures of return: the finite horizon model, and the infinite horizon model with geometric discounting. The main finding reported in the bandit literature is that problems of the second form turn out to be more tractable [9, 28]. This is convenient, since as previously noted, most work in Artificial Intelligence on learning from delayed reinforcement has also employed geometric discounting.

If a strategy is to be regarded as a policy defined on the set of belief-states then an optimal strategy is one which yields the maximal expected return in every single belief-state. A wide variety of strategies, both optimal and non-optimal, have been studied. Possibly the simplest useful strategy is that studied by Robbins who showed that a *stick on a winner, switch on a loser* strategy uniformly dominates random selection [63]. An optimal strategy can in principle be calculated for any bandit problem by posing it as a standard dynamic programming problem. The optimal strategy is expressed as a recurrence relation. Suppose we have a family of two boolean bandits. We wish to find the optimal strategy for a discount function of finite horizon n. The belief-state of a bandit is specified by the vector $[\varsigma_1, \varphi_1, \varsigma_2, \varphi_2]$, where ς_i and φ_i are the number of successes and failures respectively on arm i. The optimality equation with n stages to go is:

$$V_n^*(\varsigma_1, \varphi_1, \varsigma_2, \varphi_2) = \max \left\{ \begin{array}{lr} \mathbf{I}: & \hat{p}_1(1 + V_{n-1}^*(\varsigma_1 + 1, \varphi_1, \varsigma_2, \varphi_2)) \\ & + (1 - \hat{p}_1)\, V_{n-1}^*(\varsigma_1, \varphi_1 + 1, \varsigma_2, \varphi_2), \\ & \\ \mathbf{II}: & \hat{p}_2(1 + V_{n-1}^*(\varsigma_1, \varphi_1, \varsigma_2 + 1, \varphi_2)) \\ & + (1 - \hat{p}_2)\, V_{n-1}^*(\varsigma_1, \varphi_1, \varsigma_2, \varphi_2 + 1) \end{array} \right\} \tag{3.1}$$

where the $\hat{p}_i$ are the estimated probabilities of success on each arm a_i. The random variables P_i denoting the actual probability of success initially follow Beta distributions. Since the Beta distribution is closed under Bernoulli sampling the posterior distributions are also Beta. The estimate $\hat{p}_i$ is given by the mean of the corresponding distribution. With one stage to go the value of the bandit is given by:

$$V_1^*(\varsigma_1, \varphi_1, \varsigma_2, \varphi_2) = \max \left\{ \begin{array}{ll} \mathbf{I}: & \hat{p}_1 \\ & \\ \mathbf{II}: & \hat{p}_2 \end{array} \right\} \tag{3.2}$$

The recurrence equation may be thus be solved using any of the dynamic programming methods outlined in Section 2.5. The number of calculations required where there are k arms is given by:

$$\frac{(n-1)!}{2k!(n-2k-1)!}$$

Clearly this makes the direct application of dynamic programming unfeasible for anything other than very small problems. If the discount is finite horizon then this is unfortunately the only approach. If the discount scheme is geometric, however, the problem may be simplified somewhat. To understand how let us examine a problem where arm a_2 has a known probability of success p_2. Bellman [8] expressed this problem in terms of the recurrence equation:

$$V^*(\varsigma_1, \varphi_1, p_2) = \max \left\{ \begin{array}{ll} \mathbf{I}: & \hat{p}_1(1+\gamma V^*(\varsigma_1+1, \varphi_1, p_2)) \\ & \quad + (1-\hat{p}_1)\, V^*(\varsigma_1, \varphi_1+1, p_2), \\ & \\ \mathbf{II}: & p_2/(1-\gamma) \end{array} \right\} \tag{3.3}$$

Solving a problem with one arm known is simpler because once it is better to take arm a_2 the estimate of p_1 will not change and hence it will always be preferable thereafter to take arm a_2. Hence the expected return on arm a_2 is $p_2/(1-\gamma)$. The number of calculations required to solve Equation 3.3 if we look n steps into the future is:

$$\frac{(n-1)(n-2)}{2}$$

Gittins and Jones [29] showed how the solution to any geometrically discounted bandit problem with k arms could be reduced to solving k of these two arm bandit problems. Suppose we have a bandit described by Equation 3.3, and we want to find the value ν of p_2 that makes both arms equally desirable at this stage. For this to be the case $\nu > \hat{p}_1$. We can estimate ν by solving Equation 3.3 for successive values of p_2 until we find a value for which $p_2/(1-\gamma) = \hat{p}_1(1+\gamma V^*(\varsigma_1+1, \varphi_1, p_2)) + (1-\hat{p}_1)\, V^*(\varsigma_1, \varphi_1+1, p_2)$.

Returning to the k-arm problem let us denote the value of ν for the i^{th} arm by ν_i. We calculate each ν_i. A policy optimising the trade-off between exploration and exploitation is any policy which selects an action a_j satisfying:

$$a_j = \arg\max_{a_i \in \mathcal{A}} \{\nu_i\} \tag{3.4}$$

The ν_i are referred to as dynamic allocation indices, or more simply as Gittins Indices. Gittins and Jones prove that following the strategy determined by Equation 3.4 is the optimal strategy when discounting is geometric [29]. In general it is quickest to calculate all the indices for a reasonable range of bandit processes off-line. We do this by solving Equation 3.3 for a mesh of values of p_2. Once the $\nu(\varsigma, \varphi)$ have been obtained for all ς, φ the complexity of finding the optimal arm is $\mathrm{O}(k)$. This method works not only with boolean bandits but also with real-valued bandits. It has been shown that the method can not be applied to non-geometrically discounted processes [9].

The Gittins method has not yet been extended to the problem of optimising the exploration-exploitation trade-off in a general MDP. However the multi-state case can be explained as an extension of a real-valued bandit problem. I now proceed to explain this and the work that has been done on heuristic mechanisms for optimising the exploration-exploitation trade-off.

3.4 The multi-state case

A general multi-state Markov decision process can be thought of as a series of bandit problems, in which at each time step we not only choose an arm within a family of bandit processes and receive a reward, but also make a transition to another family of bandits. The aim is therefore to maximise total reward over the succession of families traversed. The problem contains an additional complication to the single state case in that the variance in return depends not only on the variance in the rewards at each stage, but also on the standard error of the estimates of the transition probabilities. No optimal solution has yet been found for the exploration-exploitation trade-off in this general case. There are however a variety of heuristic solutions that have been suggested within the reinforcement learning community [37, 86, 67, 69, 5, 95, 84, 85].

There have also been some attempts to formalise our notions of such exploration rules [86, 84, 22, 83]. In this section I extend these to provide a coherent framework for exploration in terms of the RL framework previously outlined. I distinguish four components of any exploration rule: the *measure* of exploratory worth employed; whether this measure is evaluated with respect to neighbouring states alone or also with respect to distant states; whether the method for inferring exploratory worth is model-based or model-free; and what form the decision rule based on this measure of exploratory worth takes.

3.4.1 Local and distal exploration

We are already familiar with the concept of a reward function. Just as we can define a function specifying immediate reward so we can define a measure of exploratory worth on the Cartesian product of the state and action spaces. This function is termed an *exploration measure*, and is denoted $\chi(s_i, a)$ for the state s_i and the action a. While reward functions may be of virtually unlimited form, possible exploration measures are more limited in number. They will be discussed in the next section. As well as defining a measure of immediate exploratory worth we can define a measure over several steps. The value of a policy with respect to an exploration measure is denoted ξ^π, and is expressed as a recurrence relation:

$$\xi^\pi(s_i) = E[\chi_t^\pi(s_i, 0)] + \gamma \sum_j p_{ij}(\pi(s_i))\xi^\pi(s_j), \; \forall s_i \in \mathcal{S} \tag{3.5}$$

where $\chi_t^\pi(s_i, n)$ is the random variable denoting the exploration value received on the n^{th} step when following policy π; and γ defines the trade-off between long and short-term exploration value. It is assumed that the n^{th} step is made at time t. In other words the local exploration value changes with time. Specifically it changes as the number of visits to that part of the state space rises. In calculating the exploration value function therefore, it is strictly necessary to carry out the dynamic programming calculation over the set of belief-states in the manner of Section 3.3. It has been seen already that this is infeasible even for single state processes. However, it is still sufficient in many cases to assume that the local exploration value does not change. In order to find the optimal exploration policy with respect to a given exploration measure we choose the action at each stage which maximises Equation 3.5. The *exploration value-function* is given by:

$$\xi^*(s_i) = \max_{a \in \mathcal{A}(s_i)} \{E[\chi_t^a(s_i, 0)] + \gamma \sum_j p_{ij}(a)\xi^*(s_j)\}, \forall s_i \in \mathcal{S} \tag{3.6}$$

Rather than use a separate notation for exploration values and exploration action-values, I will simply denote them both by the letter ξ; $\xi^*(x)$ is the exploration value function for the state x; and $\xi(x, a)$ is the exploration action-value for the state action pair x, a. One is defined in terms of the other:

$$\xi^*(s) = \max_{a \in \mathcal{A}(s)} \{\xi(s, a)\} \tag{3.7}$$

Methods which seek to maximise immediate exploratory worth are called *local* exploration methods. Equation 3.6 simplifies to a local measure when $\gamma = 0$. Methods which seek to maximise long term exploratory worth are termed *distal* exploration methods. Distal exploration methods explicitly compile the exploration values of future states into the exploration value of the current state, whereas local exploration rules, by definition, look only at the values of neighbouring states. In either case although it may be possible to find the optimal exploration policy with respect to a given measure it is not necessarily the case that a particular measure will give us the exploratory behaviour we want. The correct choice of exploration measure is thus also important.

The principal difference between an exploration value function and a value function for exploitation, is that the exploration value function is naturally non-stationary. As the agent explores the exploration values of state will change. This makes the inference problem for distal exploration harder than for learning from a stationary reward function. Distal methods are not a new idea, having been implemented previously [67, 53, 62]. They have, however, been criticised for being computationally expensive [84]. This is not necessarily the case. If the exploration values and the rewards received can be combined at the point of reception prior to the dynamic programming based inferences being made then the number of computations required to maintain an accurate estimate of the value function may not grossly outweigh those required to maintain an accurate exploration value function alone. For an embedded agent the number of backups required for each real observation will, however, typically exceed that necessary to construct an estimate of the value function for exploitation. In the worst case the exploration value function may change radically each step; making the inference problem for exploration at each step as demanding as the inference problem for the exploitation value function over all steps.

3.4.2 Exploration Measures

There are a large number of exploration measures described in the literature. I identify four basic classes of measures: *utility-based*, *counter-based*, *recency-based*, and those based on the *error* in estimates of reward or return. I discuss each in turn and proceed to discuss methods for their combination in the next section.

Utility-based measures

Utility based measures determine the exploration value of an action on the basis of its estimated expected value — its $\hat{Q}$-value.

$$\chi(x, a) = \hat{Q}(x, a)$$

If the greedy policy[2]is followed then whether exploration takes place or not depends on the initial values of $\hat{Q}(x, a)$ for all x, a. If the values are a lower bound on the set of possible returns then the agent will try only one action in each state. If, on the other hand, the values are an upper bound on the set of possible returns then the agent will be guaranteed to try all actions in each state. This latter approach has been tried by Kaelbling [37]. The alternative to strictly following the greedy policy is to use some stochastic decision rule. I discuss these in Section 3.4.5. While it is even possible to follow the worst policy there is no justification for it as a useful exploration strategy. Measures based solely on utility have one significant drawback, in that they degenerate to random walk when the value function is completely uninformed and thus perform poorly in task-environments with sparse reward. Although richer reward functions are

[2] The greedy policy is defined as the policy which selects the action with the highest *estimated* return at each time step.

in general a good idea in learning from reinforcement [50] they cannot be guaranteed. Thus finding measures more able to cope with sparse reward is important.

Counter-based measures

Counter-based measures are one such class of measures. The simplest counter-based measure looks at the number of visits for each state, and gives an estimate for each action of the expected count over the immediately succeeding states.

$$\chi(x_t, a) = \hat{E}[C_t(X_{t+1})|x_t, a]$$

where $C_t(x)$ is the number of visits to the state x by time t, and $\hat{E}$ is the estimated expectation. Clearly we would seek to select the action which minimises this measure. Thrun [86] proposed a counter-based measure expressed as the ratio of the count of the current state to the expected count of the next state under a given action:

$$\chi(x_t, a) = \frac{C_t(x_t)}{\hat{E}[C_t(X_{t+1})|x_t, a]} \tag{3.8}$$

It is also necessary to decide what to use as an estimator of $E[\hat{C}(X_{t+1})|x_t, a]$. If we possess a model of the state-transitions then we may use:

$$\hat{E}[C_t(Y)|x, a] = \sum_{y \in \mathcal{S}} p_{xy}(a) C_t(y) \tag{3.9}$$

A method simpler than explicitly maintaining estimates of the state-transitions is to use a neural network model for predicting the next state [85]. If a model of the state transitions is not available in any form then we will have to use some model-free estimator. No such estimator has been investigated to my knowledge.

Another counter-based measure used by Barto and Singh [5] is:

$$\chi(x, a) = \frac{C_t(x) - C_t(x, a)}{C_t(x)}$$

Finally the exploration rule in Moore and Atkeson's original implementation of the prioritised sweeping method can be seen as a counter-based measure, of the form

$$\chi(x,a) = \begin{cases} \tau & \text{if } C(x) \le T_{bored} \\ \hat{Q}(x,a) & \text{otherwise} \end{cases}$$

where T_{bored} is an integer, and τ is a constant upper bound on the value function. The effect of the rule is that until a state has been visited more than T_{bored} times it has a consistently high value. The agent will thus explore until it has visited all states T_{bored} times and will then exploit.

Recency-based measures

Recency-based measures are also useful for controlling exploration in environments with sparse reward. These measures of exploratory value estimate the time that has elapsed since a state was last visited, and are appropriate in non-stationary environments. Sutton first suggested their use [67], and employed an estimate of:

$$\chi(x,a) = \sqrt{\rho(x,a)}$$

Where $\rho(x,a)$ is the time that has passed since the last selection of a in x. Alternatively, Thrun [84] uses an estimator of $E[\rho(X_{t+1})|x_t, a]$, where $\rho(x)$ is the time passed since the last visit to state x.

Error-based measures

Finally there are a variety of measures which use the variance in the estimate of expected return or reward. In the simplest case a one-step horizon is used (or $\gamma = 0$) and the exploration measure is some function of the estimated variance in $R(x,a)$, which is given by:

$$\hat{\sigma}^2 = \frac{\sum_{t \in T_{xa}} (\bar{r}(T_{xa}) - r_t)^2}{n-1}$$

Where T_{xa} is the set of times at which the n previous selections of a in x occurred; and $\bar{r}(T_{xa})$ is the mean reward generated over these times. This kind of estimate is appropriately used as a component of algorithms like the interval-estimation method

[37], which work well in the single state case. In the multi-state case we find that unless we perform the full DP calculation to obtain the certainty equivalent estimate of the value function, then the mean value function is non-stationary. This has the consequence that we cannot construct an unbiased estimate of the natural variance in the value-function. This in turn causes exploration methods relying on unbiased statistics to perform badly in multi-state problems [37]. However, crude estimates of the current rate of change of the value function can still be useful. A simple estimate used [86, 55] is:

$$\chi(x,a) = |\hat{Q}_{t+1}(x,a) - \hat{Q}_t(x,a)|$$

This kind of measure can be used to direct computation as in prioritised sweeping, or to direct exploration as demonstrated in [86].

3.4.3 Combining Measures

All of the measures described in Section 3.4.2 may be combined to form compound exploration measures. Examples of compound exploration measures are [86, 37, 67].

The most widely used of these is that employed in the interval-estimation algorithm [37]. This uses the upper bound of a confidence interval as a measure for exploration-exploitation value. The upper bound is based on combining a measure of utility with a measure of variance in utility. At each step the agent updates its estimates of these and employs them to calculate the upper-bound of the confidence interval for utility. The action with the highest upper-bound is then chosen. The upper bound may be high for one of two reasons: either the confidence interval is wide — we don't know much about the action so explore; or the mean reward is high — the action is good so exploit. Initially the confidence intervals for all the actions are wide, and hence all the upper bounds are high. As the agent tries actions it drives down the upper-bounds of the confidence intervals. As we gain more information the upper-bounds for good actions will stay high, and those for poor actions will be driven down. Thus as trials mount up the agent smoothly changes from exploration to exploitation. The interval estimation measure is also a statistically well founded measure and has been analysed carefully. It performs well when unbiased estimators of utility and its variance are available. This is generally only the case in the single-state case, however.

Once $\chi(x, a)$ and $\xi(x, a)$ have been evaluated for the measure or measures being used for all $a \in \mathcal{A}_x$, there are two further separate components to any exploration rule. First if more than one measure is being used, they must be combined in the appropriate manner to derive the overall exploration value of each action in a particular state. Secondly a decision rule mapping from the final vector of exploration values to a probability distribution across actions is required. I will with deal with the first of these problems in this section and the second in Section 3.4.5.

One of the simplest combinations is a convex combination. Two estimates of the exploration value-function $\hat{\xi}_1$ and $\hat{\xi}_2$ may be combined using:

$$(1 - \Gamma)\hat{\xi}_1 + \Gamma\hat{\xi}_2$$

where $0 \leq \Gamma \leq 1$. A common choice is to combine utility and some other exploration measure in this manner. Such rules produce exploration behaviour which crudely approximates the optimal solution to exploration-exploitation trade-off. There are, however, circumstances where such a combined measure produces poor behaviour. To overcome this problem a method for dynamically adjusting the value of Γ has been devised by Thrun [84]. In Section 3.4.6 I will relate linear combinations of separate value functions to the concept of an exploration bonus.

3.4.4 Model-based vs. Model-free Exploration

In Section 3.4.2 we saw that the exploration measure $\chi(x, a)$ may be estimated using a model-based or model-free estimator. The same division applies to the estimation of $\xi^*(x)$. In the same way that we may use the techniques of Sections 2.4 and 2.5 to infer the value function for control, so we may use them to infer the exploration value function. However, for obvious computational reasons it is not worth using a model-based method for estimating one of these and a model-free method for estimating the other.

As mentioned in Section 3.4.2 in the multi-state case the exploration value function changes through time, i.e. it is non-stationary. This poses quite severe computational problems for any distal exploration method. This is because the agent ideally needs to calculate the certainty-equivalent optimal policy with respect to the exploration value function *each tick*. With existing inference methods and current computational power this is not possible. In addition model-free methods will be expected to behave particularly poorly because of their relatively slow convergence as a function of the number of experiences of the agent. Although the exploration value function is of the same fundamental form as any optimality equation, maintaining a close approximation on line is thus more complicated than in the usual stationary case.

3.4.5 Decision Rules

The most straightforward decision rule is to follow any policy which maximises the estimate of the exploration value at each stage:

$$a_t = \max_{a \in \mathcal{A}(x_t)} \{\hat{\xi}_t(x_t, a)\} \tag{3.10}$$

We can also follow a stochastic strategy. Stochastic decision rules are useful primarily with poor exploration measures. As the exploration measure used more closely approximates the optimal solution to the exploration-exploitation trade-off so it becomes more sensible to follow a deterministic strategy. We can also alter the tendency to explore through the lifetime of the agent by starting with a more random strategy and moving toward a deterministic one. The simplest stochastic decision rule is a uniform distribution across actions:

$$\Pr(a|x) = \frac{1}{|\mathcal{A}_x|}, \quad \forall a \in \mathcal{A}_x$$

This is simply random walk through the state-action space. However, depending on the transition probabilities it will not necessarily generate uniform coverage of the state space. Clearly a uniform distribution is inefficient. Its only advantage being that it requires no information other than knowledge of $\mathcal{A}_x$. It thus ignores any information supplied by any exploration measure. Because of its inefficiency it can only be employed if both learning costs and learning time are negligible.

A semi-uniform distribution is a more informed alternative to random walk. Each step the greedy policy is followed with a fixed probability, an action being selected at random otherwise.

$$\Pr(a|x) = \begin{cases} \frac{P_{best}}{|\hat{\Xi}^*_x|} + \frac{1-P_{best}}{|\mathcal{A}_x|} & \text{if } a \in \hat{\Xi}^*_x \\ \frac{1-P_{best}}{|\mathcal{A}_x|} & \text{otherwise} \end{cases}$$

where $\hat{\Xi}^* = \{a : a = \arg\max_{a \in \mathcal{A}_x} \{\hat{\xi}(x,a)\}\}$ is the set of policies that are greedy with respect to the exploration function. Semi-uniform decision rules are inefficient in the worst case, being equivalent to the uniform rule when the agent has no estimate of the exploration value function or when $P_{best} = 0$. In addition semi-uniform distributions fail to take account of how good the sub-optimal actions are. This failing is addressed by the most sophisticated of the stochastic decision rules which uses a Boltzmann distribution over the set of actions.

$$\Pr(a|x) = \frac{e^{\hat{\xi}(x,a)\theta^{-1}}}{\sum_{b \in \mathcal{A}_x} e^{\hat{\xi}(x,b)\theta^{-1}}} \qquad \forall a \in \mathcal{A}_x \tag{3.11}$$

The temperature parameter θ allows us to control the relative importance of exploration and exploitation explicitly. $0 \leq \theta \leq \infty$ and as $\theta \rightarrow \infty$ so the distribution tends to the uniform distribution. If $\theta = 0$ then a greedy policy is followed. The Boltzmann distribution has the advantage over semi-uniform distributions in that it reflects the relative worth of each action. Both semi-uniform and Boltzmann distributions, though more efficient than uniform exploration when the agent possesses at least some information about the exploration value function, degenerate to random walk in the worst case of having no information.

3.4.6 The Exploration Bonus

Exploration measures are often included in the reward function and used to calculate a single combined value function. In such cases the portion of the reward due to the exploration measure is often referred to as an *exploration bonus* [67, 22]. If the combination is linear then this is equivalent to the process of calculating the exploration value function separately and then combining it with the exploitation value function.

Suppose we have an exploration measure χ_3 which can be expressed for each state-action pair as the same convex combination of two other measures, χ_1 and χ_2:

$$\chi_3(x,a) = (1-\Gamma)\chi_1(x,a) + \Gamma\chi_2(x,a) \quad \forall x, a \tag{3.12}$$

For each measure χ_k I define $\chi_k^\pi(x,n)$ as the random variable denoting the immediate exploratory worth on the n^{th} step. The exploration value of the state x under the policy π and the measure χ_k is ξ_k^π. It is simple to show that $\xi_3^\pi = (1-\Gamma)\xi_1^\pi + \Gamma\xi_2^\pi$.

$$\begin{aligned}
\xi_3^\pi(s_i) &= E[\chi_3^\pi(s_i,0) + \gamma\chi_3^\pi(s_i,1) + \gamma^2\chi_3^\pi(s_i,2) + \ldots + \gamma^n\chi_3^\pi(s_i,n) + \ldots\ ] \\
&= E[(1-\Gamma)\chi_1^\pi(s_i,0) + \Gamma\chi_2^\pi(s_i,0) + \gamma\{(1-\Gamma)\chi_1^\pi(s_i,1) + \Gamma\chi_2^\pi(s_i,1)\} + \\
&\quad +\gamma^2\{(1-\Gamma)\chi_1^\pi(s_i,2) + \Gamma\chi_2^\pi(s_i,2)\} + \ldots \\
&\quad +\gamma^n\{(1-\Gamma)\chi_1^\pi(s_i,n) + \Gamma\chi_2^\pi(s_i,n)\} + \ldots\ ] \\
&= (1-\Gamma)E[\chi_1^\pi(s_i,0) + \gamma\chi_1^\pi(s_i,1) + \gamma^2\chi_1^\pi(s_i,2) + \ldots + \gamma^n\chi_1^\pi(s_i,n) + \ldots\ ] \\
&\quad +\Gamma E[\chi_2^\pi(s_i,0) + \gamma\chi_2^\pi(s_i,1) + \gamma^2\chi_2^\pi(s_i,2) + \ldots + \gamma^n\chi_2^\pi(s_i,n) + \ldots\ ] \\
&= (1-\Gamma)\xi_1^\pi(s_i) + \Gamma\xi_2^\pi(s_i)
\end{aligned} \tag{3.13}$$

Thus we can see that the exploration bonus view of learning can be seen as a special case of our framework for combining exploration measures, in which the combination of measures is a convex combination.

3.5 Summary

In this chapter the exploration problem has been reviewed. It has been argued that there are three separate exploration problems. The first is that of identification. The second is the problem of optimising the trade-off between identification and control over a limited time interval. In the context of learning from reinforcement this is known as the problem of optimising the exploration-exploitation trade-off. The third is to identify the optimal controller in as little time as possible. I refer to this problem as that of identification for control, and in the context of learning from reinforcement as the problem of exploration for future exploitation. This thesis addresses aspects of the last two problems. In this chapter a variety of techniques for optimising the exploration-exploitation trade-off have been reviewed. This review began with a discussion of techniques from the bandit literature which produce the optimal solution for the case of an MDP with a single-state. Heuristic techniques for approximating the optimal solution in the multi-state case have also been discussed, and set in a more general framework than has previously been presented. In the next chapter I introduce a measure to tackle the problem of exploration for future exploitation, as well as investigating further heuristic methods in the context of bandit problems. In Chapter 6 I show how these heuristic methods can be improved using some of the insights from the framework presented here.

Chapter 4

Exploration: the single state case

4.1 Introduction

In Chapter 3 I identified the problem of exploration for future exploitation and distinguished it from the better known problem of optimising the trade-off between exploration and exploitation. The former problem, though less general, has a number of useful applications as discussed in Section 3.1. In this chapter I develop methods which tackle this problem. These methods are developed here on processes with a single state, and are extended to multi-state tasks in Chapter 6.

First I develop a measure of the likelihood of each action in a state being the optimal action for that state. I derive estimators for the likelihood of a given action being optimal for processes generating boolean or normally distributed rewards (Section 4.2). An entropy based measure of knowledge about controlling the process is then derived (Section 4.4). This entropy based measure in turn leads to algorithms which control exploration by selecting an action at each stage which is expected to generate the steepest descent on the entropy function (Section 4.4.1). I argue that such algorithms are principled methods for carrying out identification for control when learning from reinforcement. Finally some empirical tests of the algorithms developed are presented.

4.2 $\Pr(a_i = a^*)$

As stated previously, in the problem of identification for control the agent seeks to maximise its knowledge about how to perform a task in a given environment. Specifically in a system which learns from reinforcement the agent's aim is *to maximise knowledge about how to maximise reward.* In other words we seek a method which will identify the optimal control policy as rapidly as possible, regardless of the exploration costs with respect to the task being learned. I start by constructing estimators of the likelihood of each action being an optimal action. For each action a_i in a state x the probability that it is an optimal action is given by:

$$\rho_i^*(x) = \Pr(a_i = a^*|x), \quad \text{where} \;\; a^* \in \mathcal{A}_x^* = \{a_k : E[\mathbf{R}(x, a_k)] \geq E[\mathbf{R}(x, a_j)], \; \forall j \neq k\}$$

If we know these probabilities it is possible to utilize them in a number of ways to guide exploration. I shall discuss some of these in Sections 4.3 and 4.4. I now proceed to derive estimators of $\Pr(a_i = a^*|x)$ given processes with boolean or non-boolean rewards.

4.2.1 Boolean reinforcement

If a bandit generates boolean rewards, then an action a_i is optimal if it has the highest probability of success each trial. The probability of success each trial for a_i is denoted p_i, and its estimate $\hat{p}_i$. Thus

$$(a_i \in \mathcal{A}_x^*) \Leftrightarrow (p_i \geq p_j, \; \forall j \neq i)$$

defines an optimal action, and

$$\hat{\mathcal{A}}_x^* = \{a_i : \hat{p}_i \geq \hat{p}_j, \; \forall j \neq i\}$$

gives the estimated set of optimal actions. Given that our estimates $\hat{p}_i$ are uncertain there is in fact a set of possible sets $\mathcal{A}_x^*$, and we can define a probability distribution over this set of sets. The question is how? If we knew the p_i then we could use the binomial distribution to give the probability of generating ς_i successes and φ_i failures on each arm a_i. The inverse problem is to assign a probability to each possible value of p_i (clearly $0 \leq p_i \leq 1$) given that we have observed ς_i successes and φ_i failures on arm a_i. We may view the different possible values of p_i as hypotheses and the observed values ς_i and φ_i as data. Bayes' rule allows us to revise the probability of each hypothesis on the basis of observed data and an *a priori* distribution across the hypotheses:

$$\Pr(H_1|D) = \frac{\Pr(D|H_1)\Pr(H_1)}{\sum_{H\in\mathcal{H}}\Pr(D|H)\Pr(H)}$$

where H_1 is a hypothesis in the set of hypotheses $\mathcal{H}$[1]. If we apply Bayes' rule to the binomial distribution we obtain a Beta density, which takes the form:

$$f(p) = \frac{1}{B(\varsigma+1,\varphi+1)}p^{\varsigma}(1-p)^{\varphi} \tag{4.1}$$

where $0 \le p \le 1$, and $B(a,b)$ denotes Euler's beta function.

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{4.2}$$

We write the Beta density parameterised by ς, φ as $\mathrm{Beta}(\varsigma,\varphi)$. Thus for any number of actions we can calculate the probability of an action a_i being optimal by evaluating the integral:

$$\Pr(a_i \succeq a_j, \forall j \neq i) = \int_0^1 f_{P_i}(p_i)\prod_{\forall j\neq i}\int_0^{p_i} f_{P_j}(p_j)\,dp_j\,dp_i \tag{4.3}$$

where $f_{P_k} \sim \mathrm{Beta}(\varsigma_k,\varphi_k)$. It transpires that it is only practical to evaluate this integral in the two action case[2]. I now proceed to do so. The case of (4.3) for two actions is:

$$\Pr(a_i \succeq a_j) = \int_0^1\int_0^{p_i} f_{P_i}(p_i)\,f_{P_j}(p_j)\;dp_j\,dp_i \tag{4.4}$$

where $f_{P_i} \sim \mathrm{Beta}(\varsigma_i,\varphi_i)$ and $f_{P_j} \sim \mathrm{Beta}(\varsigma_j,\varphi_j)$. The *cdf* of $\mathrm{Beta}(\varsigma,\varphi)$ is given by

$$\int_0^p \frac{1}{B(\varsigma+1,\varphi+1)}p^{\varsigma}(1-p)^{\varphi}\,dp = \sum_{k=\varsigma+1}^{n-1}\binom{n-1}{k}p^k(1-p)^{n-k-1} \tag{4.5}$$

[1] A condition of Bayes' rule is that the hypotheses are mutually exclusive and exhaustive.

[2] It is possible to substitute in the known values for ς_k, φ_k for each arm a_k, expand the product and then integrate, but the number of terms will rise exponentially with the number of actions, thus making the approach infeasible.

where $n = \varsigma + \varphi + 2$. In order to evaluate [4.4] we first substitute [4.5] and [4.1] into [4.4] and rearrange to give:

$$\Pr(a_i \succeq a_j) = \frac{1}{B(\varsigma_i + 1, \varphi_i + 1)} \int_0^1 \sum_{k=\varsigma_j+1}^{n_j-1} \binom{n_j - 1}{k} p_i^{\varsigma_i+k}(1-p_i)^{n_j+\varphi_i-k-1}\, dp_i \quad (4.6)$$

Since integration is distributive over summation this can be rearranged further as follows:

$$\Pr(a_i \succeq a_j) = \frac{1}{B(\varsigma_i + 1, \varphi_i + 1)} \sum_{k=\varsigma_j+1}^{n_j-1} \binom{n_j - 1}{k} \int_0^1 p_i^{\varsigma_i+k}(1-p_i)^{n_j+\varphi_i-k-1}\, dp_i \quad (4.7)$$

The expression which remains to be integrated is in fact Euler's beta function given previously, which evaluates to:

$$\int_0^1 p^{\varsigma}(1-p)^{\varphi}\, dp = \frac{(\varsigma)!(\varphi)!}{(\varsigma + \varphi + 1)!} \quad (4.8)$$

when ς, φ are integer. Substituting [4.8] into [4.6] we arrive at:

$$\Pr(a_i \succeq a_j) = \frac{1}{B(\varsigma_i + 1, \varphi_i + 1)} \sum_{k=\varsigma_j+1}^{n_j-1} \binom{n_j - 1}{k} \frac{(\varsigma_i + k)!(n_j + \varphi_i - k - 1)!}{(n_i + n_j - 2)!} \quad (4.9)$$

Thus we have obtained a closed form expression for the probability of either action being optimal in a two-armed boolean bandit. This expression can be evaluated in $O(\varphi_j)$ for each action a_i if a look-up table is used for the factorials[3]. This is unfortunate as it means evaluation time is unbounded, rising with the number of trials. Furthermore, as previously stated it is not possible to derive a closed form expression for more than two actions. These difficulties make the direct approach to finding $\Pr(a_i = a^*)$ for boolean bandits unfeasible. I now turn to the non-boolean case, i.e. when the reward model is of type Q or S. I then proceed to show how these estimates of $\Pr(a_i = a^*)$ can be used to control exploration.

[3] An alternative formulation can be derived where the complexity is $O(\varsigma_j)$. So the worst case will be when $\varsigma_j = \varphi_j$ and they are large.

4.2.2 Non-Boolean reinforcement

When an environment generates non-boolean reinforcement the agent has no absolute measure of success or failure. Actions only have value relative to one another. One way of comparing actions is to compare the mean reward they generate. Thus the set of optimal actions is given by:

$$(a_i \in \mathcal{A}_x^*) \Leftrightarrow (\mu_i \geq \mu_j, \forall j \neq i)$$

where $\mu_k = E[R(x, a_k)]$. According to the Central Limit Theorem whatever distribution the reward follows, the mean reward will follow a normal density if the sample is large. Using this fact we can calculate the probability that each action generates a mean reward higher than that for any other action, based on the sequence of rewards observed to follow each action. The integral is of the form:

$$\Pr(\mu_i \geq \mu_j, \forall j \neq i) = \int_{-\infty}^{\infty} f_{X_i}(x_i) \prod_{\forall j \neq i} \int_{-\infty}^{x_i} f_{X_j}(x_j)\, dx_j\, dx_i \tag{4.10}$$

where $f_{X_k} \sim \mathcal{N}(\hat{\mu}_{\bar{k}}, \hat{\sigma}_{\bar{k}}^2)$ is the estimated sampling distribution of the mean reward. $\hat{\mu}_{\bar{k}}$ is thus some estimate of the mean of the sample mean and $\hat{\sigma}_{\bar{k}}^2$ is some estimate of the variance of the sample mean. Equation 4.10 becomes

$$\Pr(\mu_i \geq \mu_j, \forall j \neq i) = \int_{-\infty}^{\infty} \frac{1}{\hat{\sigma}_{\bar{\imath}}} Z\left(\frac{x_i - \hat{\mu}_{\bar{\imath}}}{\hat{\sigma}_{\bar{\imath}}}\right) \left[\prod_{\forall j \neq i} \Phi\left(\frac{x_i - \hat{\mu}_{\bar{\jmath}}}{\hat{\sigma}_{\bar{\jmath}}}\right)\right] dx_i \tag{4.11}$$

where $Z(x)$ is the standard normal density and $\Phi(x)$ is the normal *cdf*. Since evaluating this integral is not possible directly its value may instead be approximated:

$$\Pr(\mu_i \geq \mu_j, \forall j \neq i) \simeq \sum_{x_i = x_{low}}^{x_i = x_{high}} \frac{1}{\hat{\sigma}_{\bar{\imath}}} Z\left(\frac{x_i - \hat{\mu}_{\bar{\imath}}}{\hat{\sigma}_{\bar{\imath}}}\right) \prod_{\forall j \neq i} \Phi\left(\frac{x_i - \hat{\mu}_{\bar{\jmath}}}{\hat{\sigma}_{\bar{\jmath}}}\right) \delta x_i \tag{4.12}$$

where sufficiently small steps δx_i, are taken over a large enough interval $[x_{low}, x_{high}]$, for the error to be negligible. The amount of time taken to evaluate the sum for each action is of order $O(((x_{high} - x_{low})/\delta x_i)m)$, where there are m actions. One way to reduce this complexity would be to learn the mapping for m actions using a function approximator such as a multi-layer perceptron. The accuracy of such methods have not been investigated here. I discuss the estimators $\hat{\mu}_{\bar{\imath}}$ and $\hat{\sigma}_{\bar{\imath}}$ in Section 4.3.

4.3 Probability Matching Algorithms

One method for allocating trials in a process is to choose actions according to how likely they are to be optimal. This can be seen as an extension of *probability matching.* Probability matching means making a particular prediction for a process outcome with the probability that it is the correct prediction. In reinforcement learning an action is correct if it is optimal. When carrying out probability matching in a reinforcement learning framework the number of times each action is selected is thus in proportion to the probability of that action being the optimal action. Probability matching policies provide a simple (though non-optimal) method for guiding exploration. More interestingly they have been recorded as being similar to the decision rates observed as being employed by humans [88]. Finally there has been some investigation of their use for the allocation of trials in non-stationary processes [23]. In this chapter probability matching methods are tested for control of both the exploration-exploitation trade-off and exploration for future exploitation.

Algorithms 7 and 8 (see Figures 4.1 and 4.2) are simple probability matching algorithms for boolean and non-boolean processes respectively. They are implemented here in a Bayesian framework in the sense that they may take account of prior belief concerning the probability distributions of the p_i or $\mu_{\bar{\imath}}$. The distributions may be uniform, the method then conforming to the distributional assumptions of classical statistics. I present the boolean algorithm only for the two action case for which it has proved possible to derive a closed form expression for ρ_i^*. The non-boolean algorithm can also be used for learning from boolean rewards, but is less effective, as we shall see.

Although probability matching approaches are not optimal for controlling the exploration-exploitation trade-off they have the advantage that they are guaranteed to converge. I first present a simple proof that the boolean probability matching algorithm converges for the two arm case. We start by thinking about the general case. As the number of samples on each arm tends to infinity so the maximum likelihood estimate ς_i/φ_i tends

Algorithm 7 BOOLEAN PROBABILITY MATCHING

$t := 0$
$f_{P_1} \sim Beta(\varsigma_1, \varphi_1)$ and $f_{P_2} \sim Beta(\varsigma_2, \varphi_2)$ see Eq. 4.1
choose prior ς_i, φ_i, for all actions a_i
choose $a(t) := a_j$ randomly
loop
  observe r_t
  if $r_t = 1$ then $\varsigma_j = \varsigma_j + 1$
  else $\varphi_j = \varphi_j + 1$
  calculate ρ_i^* for each a_i using Eq. 4.9
  t:=t+1
  choose $a(t) := a_j$ where $\Pr(a(t) = a_i) = \rho_i^*$

Figure 4.1: The boolean probability matching algorithm.

to p_i.

$$\lim_{n_i \to \infty} \varsigma_i / \varphi_i = p_i$$

Thus the limit as $n_i \to \infty$ of the standard error of the estimate ς_i/φ_i is 0. This is equivalent to saying that the variance of the beta density f_{P_i} tends to 0 as $n_i \to \infty$. We also need to assume that there is in fact a single optimal action, i.e. that $\Pr(p_i = p_j) = 0, \forall i, j$. From these two facts it follows that as the number of trials on every arm tends to infinity, the probability of optimality for the optimal arm a^* will tend to 1 and all other probabilities will tend to 0, i.e.

$$\lim_{n_i \to \infty, \forall i} \rho_k^* \to 1 \quad \text{if } a_k = a^*$$

and that

$$\lim_{n_i \to \infty, \forall i} \rho_k^* \to 0 \quad \text{if } a_k \neq a^*$$

To prove convergence we need to show that as the total number of trials n tends to infinity, the number of trials on each arm will also tend to infinity. This will be the case if for any finite number of trials, the probability of being selected each trial, ρ_i^*, is greater than 0 by some finite amount for all arms regardless of the history of successes and failures on each arm. Turning to the two arm case we note that the probability ρ_i^* will tend to 0 most quickly if the number of successes on arm a_i is 0 and there are no failures on the other arm a_j. In this instance Equation 4.9 simplifies to

$$\rho_i^* = \frac{\varphi_i(\varsigma_j + 1)!\varphi!}{(\varphi_i + \varsigma_j + 2)!}$$

For any finite number of trials ς_j and φ_i will be finite, so both the denominator and the numerator will be finite positive integers. This in turn implies that the value ρ_i^* can be expressed as a finite positive rational number. Thus the probability of taking any action a_i will always be non-zero. This completes the proof.

A non-boolean probability matching algorithm will in principle converge under similar conditions, i.e. if each probability $\rho_i^* > 0$ and finite. This will be the case if the standard error of the sample mean is greater than zero and finite, because the distribution of the sample mean is normal, and the tails of the normal density $f(x)$ are finite for all finite values of x. The distribution of the sample mean is assumed to be normal if the sample is large (≥ 30). If there is a finite probability that the observed rewards are identical then use of a vague Bayesian prior will ensure a finite but small posterior for the standard error of the posterior mean. In practice any guarantee of convergence is limited by the accuracy of the approximation to ρ_i^* given by Equation 4.12. The step size should generally be such that a reasonable number of samples are taken from the distribution f_{X_j} with the smallest standard deviation.

Algorithm 7 (Figure 4.1) starts with Bayesian priors for the distributions of the P_i for each of the two arms. From the earlier proof it should be clear that whatever initial values are chosen for ς_i, φ_i, eventual convergence will not be affected, as in the long term the posterior distribution will be dominated by the observed successes and failures. Clearly, however, a poor choice of Bayesian priors will hamper convergence speed considerably. If nothing is known of the process then sensible values for the Beta priors are $\varsigma_i = \varphi_i = 0$ for all a_i. In this case the method conforms to the distributional

Algorithm 8 NON-BOOLEAN PROBABILITY MATCHING

The reward associated with arm a_i follows an unknown distribution with population mean μ_i and standard deviation σ_i. The sample mean at time t for the action a_i is denoted $\bar{r}_t(i)$ and is normally distributed with prior mean $\mu'_{\bar{i}}$ and prior standard deviation $\sigma'_{\bar{i}}$. n'_i is calculated from these using Eq. 4.13. $\mu''_{\bar{i}}$ is the posterior mean.

$t := 0$
$n_i := 0,\ \bar{r}_t(i) := 0, \forall i$
loop until $n_i \geq 30$, $\forall i$
    choose $a(t) := a_j$ according to some exploration scheme
    update sample statistics
loop
    choose $a(t) := a_j$ using $\Pr(a(t) = a_i) = \rho^*_i$
    observe r_t
    $n_j := n_j + 1$
    $\bar{r}_t(j) := \bar{r}_{t-1}(j)(n_j - 1)/n_j + r_t/n_j$
    update $\hat{\sigma}_i$ using Equation 4.15
    calculate μ''_i using Eq. 4.14 and $\sigma''_{\bar{i}}$ using Eq. 4.16
    calculate $\rho^*_i = \Pr(a_i = a^*)$ using Eq. 4.12
    $\mu'_i := \mu''_i$ and $\sigma'_{\bar{i}} := \sigma''_{\bar{i}}$
    $t := t + 1$

Figure 4.2: The non-boolean probability matching algorithm.

assumptions of classical statistics.

The non-boolean algorithm (Figure 4.2) is more difficult in that we need to make more complex assumptions about the prior distributions involved and must carefully govern the parameters of the approximation used (Eq. 4.12). We first assume that the sample is large. In this case the distribution of the sample mean is normal with mean $\mu_{\bar{i}}$ and standard deviation $\sigma_{\bar{i}}$.

Taking a Bayesian approach we express $\hat{\mu}_{\bar{i}}$ in terms of a prior, $\mu'_{\bar{i}}$ and a posterior $\mu''_{\bar{i}}$. We similarly employ $\sigma'_{\bar{i}}$ and $\sigma''_{\bar{i}}$. Having assumed that the sample mean reward for action a_i is normally distributed with priors $\mu'_{\bar{i}}$ and $\sigma'_{\bar{i}}$ we must also determine the relationship between $\sigma'_{\bar{i}}$ and the standard deviation of the reward itself, σ_i. Because the sample is large we may use the sample standard deviation as an estimate of σ_i. The relation between $\sigma'_{\bar{i}}$ and σ_i is given by,

$$\sigma'_{\bar{i}} = \frac{\sigma_i}{\sqrt{n'_i}} \tag{4.13}$$

where n'_i can be thought of as reflecting the strength of our belief in the prior mean reward. If n'_i is small then our prior distribution for the mean reward is vague. If n'_i is large then our certainty is high. In the experiments conducted in this thesis the learner employed a counter-based exploration rule until each action had been sampled thirty times. The value of σ_i can then be estimated reasonably. This estimate is combined with the initial value for $\sigma'_{\bar{i}}$ in order to obtain n'_i. This value is then used in Equation 4.16 to obtain the posterior standard deviation. The prior $\sigma'_{\bar{i}}$ employed in the experiments described here was large in value, thus indicating little prior knowledge about the distribution of the mean.

The boundary conditions for Algorithm 8 are thus defined by the priors $\mu'_{\bar{i}}$, $\sigma'_{\bar{i}}$, for each arm a_i. Each step the corresponding posteriors are calculated by combining the priors with the sample data[4]. The posterior mean is given by,

$$\mu''_{\bar{i}} := \frac{\mu'_{\bar{i}} n'_i + \bar{r}(i) n_i}{(n'_i + n_i)} \tag{4.14}$$

where n_i is the number of samples on the i^{th} arm and $\bar{r}(i)$ is the corresponding sample mean. The sample variance can be calculated using:

$$s_t^2 = \frac{\sum_{j=0}^{t} r_j^2 - 2\bar{r}_t \sum_{j=0}^{t} r_j + n(\bar{r}_t)^2}{n-1} \tag{4.15}$$

when the sample size at time t is n. This requires that we incrementally maintain $\sum_{j=0}^{t} r_j^2$ and $\sum_{j=0}^{t} r_j$. Because we improve our estimate of σ_i through time we must

[4] The priors used here are approximately uniform, and the inference method used is the maximum likelihood method for estimating the parameters of the sampling distribution of the mean reward. Given the prior distribution of the mean it is actually strictly preferable to employ an optimal Bayesian estimator as described in [87].

also adjust n'_i using Equation 4.13. We may then calculate the posterior standard deviation using:

$$\sigma''_{\bar{i}} = \frac{\sigma_i}{\sqrt{n'_i + n_i}} \tag{4.16}$$

The non-boolean algorithm also has the problem that it cannot strictly be used until the sample size for every arm is at least 30. Thus prior to this point another exploration method must be employed. In these experiments a counter-based method was used. The issue may be fudged slightly by requiring that the total $n_i + n'_i \geq 30$, for each arm. This ignores the problem that as the sample size decreases the estimate of σ_i worsens, in turn making the estimate n'_i unreliable. We examine the empiricial performance of both algorithms in Section 4.6, now turning to further uses of the measure $\Pr(a_i = a^*)$ in guiding exploration for future exploitation.

4.4 An entropic measure of task knowledge

Having defined estimators of $\Pr(a_i = a^*)$ for single state processes with both boolean and non-boolean reward models, I now proceed to define a measure of the *degree of uncertainty as to the identity of the optimal policy.* We do this by employing the well-known measure for entropy from information theory. I use ρ^*_i as a shorthand for $\Pr(a_i = a^*)$. The entropy concerning the identity of the optimal policy in a single state process is denoted simply H:

$$H = -\sum_{\forall i} \rho^*_i \log_2(\rho^*_i) \tag{4.17}$$

The greater the degree of uncertainty as to the identity of the optimal action the higher the value of H. As the agent acquires knowledge about the task H will decline to a minimum. If the agent has no information to distinguish the a_i then the ρ^*_i will all be the same and H will be at its maximum for that number of actions. As the number of actions $|\mathcal{A}| \to \infty$ so the maximum value of $H \to \infty$. Thus the measure may not be extendible to the case of continuous action spaces. If one action alone is optimal then as $t \to \infty$, $H \to 0$. If more than one action is optimal this will not be the case. I now discuss a method for guiding exploration in single state tasks using this measure.

4.4.1 An entropy reduction algorithm for exploration

A simple method for guiding exploration for future exploitation in processes with a single state is to choose an action which is likely to lead to a large reduction in entropy. The most *informative* action to take is the action that is predicted as generating the largest reduction on average. How can we determine this action? We need to be able to calculate the expected value of H at the next time step given that we take action a_k this time step[5]. We then select the action that minimises this expected entropy next step:

$$a_t = \arg \min_{\forall a_k} \{E[H_{t+1}|a_k]\}$$

I now describe a non-boolean entropy reduction algorithm based on the insight above, and using Eq. 4.12. Each step we calculate the expected entropy under each possible action at that time. Suppose that the agent executes action a_k at this time. Let us also assume that the estimate $\mu''_{\bar{k}}(t)$ of $\mu_{\bar{k}}$ at time t is accurate, i.e. that the difference $|\mu''_{\bar{k}}(t) - \mu_{\bar{k}}|$ is negligible. Under these circumstances, if we take another observation the underlying variance in reward σ_k will remain the same on average, and the sample size will increase by one, thus reducing the posterior standard deviation of the mean reward:

$$E[\sigma''_{\bar{k}}(t+1)] = E\left[\frac{\sigma_k}{\sqrt{n'_k + n_k + 1}}\right] < E\left[\frac{\sigma_k}{\sqrt{n'_k + n_k}}\right] = E[\sigma''_{\bar{k}}(t)] \qquad (4.18)$$

where n_k is the number of observations made on arm a_k by time t. Thus $E[H_{t+1}|a(t) = a_k]$ can be calculated from the $E[\rho^*_i(t+1)|a(t) = a_k]$, which are in turn calculated using Eq. 4.12 on the assumption that $\sigma''_{\bar{k}}(t+1)$ is given by the expectation of Eq. 4.18.

Because the standard deviation of the mean reward is smaller the entropy will be lower. In general the size of the predicted entropy reduction will depend on several quantities:

[5] We could also calculate the gradient in H given that we took a_k, but differentiating the formula for ρ^*_i is not always trivial.

the distance between the mean reward for the action chosen and the mean rewards for other actions; the ranking of the mean reward for the action chosen; and the value of $n'_k + n_k$. The actual entropy reduction will differ from the expected reduction. If the difference $|\mu''_{\bar{k}} - \mu_{\bar{k}}|$ is large then the entropy may increase on average. This is because the estimated mean of the chosen arm may shift toward the means for the other distributions. Equally any observation which deviates from the estimated mean by more than σ_k will increase our estimate of σ_k and hence our estimate of $\sigma_{\bar{k}}$. This is why it is necessary to assume that the difference $|\mu''_{\bar{k}} - \mu_{\bar{k}}|$ is small. The practical implication of this is that as the estimates of the mean worsen so does the performance of the algorithm.

The main features of Algorithm 9 (Figure 4.3) are the same as for Algorithm 8 previously. Priors must be chosen carefully and a separate algorithm may be chosen to govern exploration until the samples on each arm are large. I now discuss an algorithm for controlling exploration for future exploitation in boolean environments based on confidence intervals. This is followed by an empirical comparison of the performance of the algorithms presented here with that of some well known algorithms for learning from reinforcement.

4.5 A heuristic algorithm

Because of the difficulties of producing algorithms for learning from boolean rewards, techniques for learning based on the confidence intervals were also investigated. A simple algorithm based on this work is presented here (Figure 4.4). The algorithm works in the following manner.

The agent is initially equally uncertain about all actions, so the probability distribution across actions is uniform when $t = 0$. At each time step the action is chosen about which the agent knows least. This is defined as the action a_i with the widest confidence interval on the rate of success p_i. This confidence interval is given by:

$$\mathrm{ci}(\varsigma_i, \varphi_i) = \frac{2\frac{z_{\alpha/2}}{\sqrt{(\varsigma_i+\varphi_i)}}\sqrt{\frac{\varsigma_i}{(\varsigma_i+\varphi_i)}(1 - \frac{\varsigma_i}{(\varsigma_i+\varphi_i)}) + \frac{z^2_{\alpha/2}}{4(\varsigma_i+\varphi_i)}}}{1 + \frac{z^2_{\alpha/2}}{(\varsigma_i+\varphi_i)}} \tag{4.19}$$

Algorithm 9 NON-BOOLEAN ENTROPY REDUCTION

The reward associated with arm a_i follows an unknown distribution with population mean μ_i and standard deviation σ_i. The sample mean at time t for the action a_i is denoted $\bar{r}_t(i)$ and is normally distributed with prior mean μ_i' and prior standard deviation σ_i'. n_i' is calculated from these using Eq. 4.13. μ_i'' is the posterior mean.

$t := 0$
$n_i := 0,\ \bar{r}_t(i) := 0, \forall i$
loop until $n_i \geq 30$, $\forall i$
  choose $a(t) := a_j$ according to some exploration scheme
  update sample statistics
loop
  choose $a(t) := a_j$ such that $a_i = \arg\min_{a_i \in \mathcal{A}}\{E[H_{t+1}|a(t) = a_i]\}$
  observe r_t
  $n_j := n_j + 1$
  $\bar{r}_t(j) := \bar{r}_{t-1}(j)(n_j - 1)/n_j + r_t/n_j$
  update $\hat{\sigma}_i$ using Equation 4.15
  calculate μ_i'' using Eq. 4.14 and σ_i'' using Eq. 4.16
  calculate $\rho_i^* = \Pr(a_i = a^*)$ using Eq. 4.12
  calculate $E[H_{t+1}|a_i]$ for each a_i using Eq. 4.17
  $\mu_i' := \mu_i''$ and $\sigma_i' := \sigma_i''$
  $t := t + 1$

Figure 4.3: The non-boolean entropy reduction algorithm.

where $z_{\alpha/2}$ is the value that will be exceeded by the value of a standard normal variable with probability $\alpha/2$, and ς_i and φ_i are the numbers of successes and failures on arm i respectively. Thus this algorithm simply chooses the actions about which it can learn most, thus controlling exploration for future exploitation.

4.6 Empirical Comparison

Having presented a number of algorithms for controlling both exploration for future exploitation and the exploration-exploitation trade-off I now present results of empirical

Algorithm 10 BOOLEAN CONFIDENCE INTERVAL

$ci(\varsigma_i, \varphi_i)$ is the width of the $100(1-\alpha)\%$ confidence interval for action a_i given by Eq 4.19; $a(t)$ is the action at time t;and n_i, x_i are the number of trials and successes for action a_i.

$t := 0$
$n_i := 0$, $x_i := 0$, $\forall i$
loop
    choose $a(t) := a_j$ such that $ci(\varsigma_j, \varphi_j) \geq ci(\varsigma_i, \varphi_i), \forall i$
    observe r_t
    $n_j := n_j + 1$
    if $r(t) = 1$ then $x_j := x_j + 1$
    t:=t+1

Figure 4.4: The boolean confidence interval algorithm.

investigations concerning their performance on single state tasks. The performance of all algorithms was examined according to both criteria. First, the different tasks are discussed, and then additional algorithms which were tested are described.

4.6.1 Tasks

Eight tasks were used to test the algorithms described. Tasks 1-4 are single state, two action processes in which the agents generate boolean reinforcements, the individual tasks being defined by the rate of success associated with each action (see Table 4.1). Tasks 5 and 6 are two action processes, generating normally distributed reinforcements, with mean μ_i for action a_i, and standard deviation $\sigma_i = .3$ for all actions (see

Task	p_i	
	a_1	a_2
1	.8	.2
2	.55	.45
3	.9	.8
4	.2	.1

Table 4.1: Boolean tasks. $\mathcal{R} = \{0, 1\}$. Each pull on an arm is a Bernoulli trial.

Task	μ_i				
	a_1	a_2	a_3	a_4	a_5
5	.8	.2	-	-	-
6	.55	.45	-	-	-
7	.9	.7	.5	.3	.1
8	.6	.55	.5	.45	.4

Table 4.2: Non-boolean tasks. $R(i) \sim \mathcal{N}(\mu_i, \sigma_i^2)$. $\sigma_i = .3$, $\forall i$.

Table 4.2). Finally Tasks 7 and 8 are single state, 5 action processes again with normally distributed reinforcements (see Table 4.2). In each group of tasks the differences between the actions are relatively large in the first task and smaller in the other tasks. In the boolean tasks the third and fourth tasks have relatively high and low average rates of success respectively.

4.6.2 Agents

In addition to the agents presented above, the behaviour of three other agents on these tasks was examined. The first of these was a reinforcement comparison algorithm due to Sutton [78] (see Figure 4.5).

It was also decided to compare the performance of interval estimation techniques. There are two algorithms based on the interval estimation technique that are appropriate here. The first is the original binomial IE algorithm [37] (see Figure 4.6). The following equation was used to determine the upper bound of the confidence interval on the P_i:

$$\mathrm{ub}(x,n) = \frac{\frac{x}{n} + \frac{z_{\alpha/2}^2}{2n} + \frac{z_{\alpha/2}}{\sqrt{n}}\sqrt{\frac{x}{n}(1-\frac{x}{n}) + \frac{z_{\alpha/2}^2}{4n}}}{1 + \frac{z_{\alpha/2}^2}{n}} \tag{4.20}$$

where $z_{\alpha/2}$ is the value that will be exceeded by the value of a standard normal variable with probability $\alpha/2$, and x and n are the numbers of successes and the number of trials respectively. This is the same formula employed by Kaelbling [37].

The second IE algorithm [37] is non-parametric (see Figure 4.7). As implemented here this employs two different estimators of the upper bound of the confidence interval for the centre of the underlying distribution. If the number of trials on an arm is small

Algorithm 11 REINFORCEMENT COMPARISON

$0 < \beta \leq 1,\quad 0 < \alpha \leq 1.$ $\tilde{r}_t$ is the estimated mean reward. $N \sim \mathcal{N}(w_t, .09)$ is a random variable.

$t := 0$

$w_t := 0,\ \tilde{r}_t := 0$

loop

$a(t) := \begin{cases} 1 & \text{if } N > 0 \\ 0 & \text{otherwise} \end{cases}$

observe r_t

$\hat{r}_{t+1} := r_t - \tilde{r}_t$

$\tilde{r}_{t+1} := \tilde{r}_t + \beta\hat{r}_{t+1}$

$w_{t+1} = w_t + \alpha\hat{r}_{t+1}\{a(t) - \Pr(N > 0)\}$

$t := t + 1$

Figure 4.5: Sutton's reinforcement comparison algorithm.

then u is employed as an estimate of the upper bound, where u is the largest value such that:

$$\sum_{k=0}^{u} \binom{n}{k} .5^n \leq \alpha/2 \tag{4.21}$$

If the sample size is large (> 30) however then we use the fact that the sampling distribution of the mean is normal:

$$u = \bar{r} + z_{\alpha/2} s_{\bar{r}} \tag{4.22}$$

where $s_{\bar{r}}$ is the standard deviation of the sample mean $\bar{r}$. To my knowledge no empirical results for the performance of the non-parametric interval estimation method on bandit tasks have been published.

Algorithm 12 Binomial Interval Estimation

$t := 0$

$x_i := 0,\ n_i := 0, \forall i$

choose $a(t) := a_j$ randomly

loop

    observe r_t

    $x_j := x_j + r_t$

    $n_j := n_j + 1$

    calculate $ub(x_i, n_i)$ for each a_i using Eq. 4.20

    t:=t+1

    choose $a(t) := a_j$ where $a_j = \arg\max_{a_i \in \mathcal{A}}\{ub(x_i, n_i)\}$

Figure 4.6: The binomial interval estimation algorithm.

4.6.3 Method

The algorithms were tested on tasks 1-8, the boolean algorithms only being applied in boolean tasks. The distribution of algorithms over tasks is summarised in Table 4.3. The same number of runs of the same length were carried out for all algorithms on a given task. The length and the number of runs were varied across tasks. For simpler tasks fewer runs of shorter length were carried out. It was found necessary to evaluate algorithms on the harder tasks using more runs of greater length. This information is summarised in Table 4.4.

All algorithms were run across a range of parameter settings. These are detailed in Table 4.5. In a number of the graphs and illustrations the various algorithms are

Tasks	**Algorithms**
1-4	7, 8, 10, 11, 12, 13
5,6	8, 9, 11, 13
7,8	8, 9, 13

Table 4.3: Tasks 1-8. Distribution of algorithms

Algorithm 13 NON-PARAMETRIC IE

$t := 0$
$x_i := 0,\ n_i := 0, \forall i$
choose $a(t) := a_j$ randomly
loop
    observe r_t
    $x_j := x_j + r_t$
    $n_j := n_j + 1$
    calculate $ub(x_i, n_i)$ for each a_i using Eqs. 4.21 and 4.22
    t:=t+1
    choose $a(t) := a_j$ where $a_j = \arg\max_{a_i \in \mathcal{A}}\{ub(x_i, n_i)\}$

Figure 4.7: The non-parametric interval estimation algorithm.

referred to either by an acronym, or by their order in the thesis. To summarise, these are the algorithms tested and the shorthand used to denote them:

Algorithm 7	**BPM**	Boolean probability matching
Algorithm 8	**NBPM**	Non-boolean probability matching
Algorithm 9	**NBER**	Non-boolean entropy reduction
Algorithm 10	**BCI**	Boolean confidence interval method
Algorithm 11	**RC**	Reinforcement Comparison
Algorithm 12	**BIE**	Binomial interval estimation
Algorithm 13	**NPIE**	Non-parametric interval estimation

4.6.4 Results

The performance of each algorithm on each task were evaluated according to two criteria. The classical measure of performance in learning from reinforcement is average reward generated over the learning period. This is maximised by an agent which optimises the exploration-exploitation trade-off. However, I have argued that learning

Task	runs	ticks/run
1	200	100
2	1000	100
3	400	100
4	1000	100
5	200	500
6	200	500
7	200	1000
8	200	1000

Table 4.4: Tasks 1-8. Numbers and length of runs.

Algorithm	**Parameters**
7	$\varsigma_i = \varphi_i = 0,\ \forall a_i \in \mathcal{A}$
8	$\mu_i' = 0,\ \sigma_i' = 100,\ \forall a_i \in \mathcal{A}$
9	$\mu_i' = 0,\ \sigma_i' = 100,\ \forall a_i \in \mathcal{A}$
10	$z_{\alpha/2} = \{1, 2, 3, 4\}$
11	$\beta = .1,\ \alpha = \{.05, .1, .2, .3, .4, .5, .6\}$
12	$z_{\alpha/2} = \{1, 2, 3, 4\}$
13	$\alpha/2 = \{.05, .025, .005, .0005\}$

Table 4.5: Tasks 1-8. Parameter settings.

from reinforcement is also usefully seen in a framework where the costs of learning are different from the cost function being learned. In this instance we seek a measure of the performance of the policy the agent has learned. The simplest measure would simply be to estimate the reward generated by the greedy policy at the end of the learning period. Such a measure tells us only which policy we think is best. It provides no measure of confidence in this prediction.

In Section 1.3 I discussed the notion that one of the appealing qualities of adaptive controllers is that they are often verifiable in the limit. In practical terms, however, we would like to be able to verify controllers after a finite number of steps. Clearly for *any* controller inferred on the basis of observations from a stochastic process we can never provide a guarantee of the controller's performance. We are only ever able to provide probabilistic verification. Such probabilistic verification is neatly supplied by the estimates of $\Pr(a = a^*)$ constructed in Section 4.2. If we denote the action specified by the greedy policy as a_g then we say that our confidence in the greedy policy is $100\rho_g^*\%$, i.e. that the policy constructed by the learning agent is optimal

Task	Algorithm			
	bci ($z_{\alpha/2}$)	rc (α)	bie ($z_{\alpha/2}$)	npie ($\alpha/2$)
1	4.0	.05	4.0	.005
2	2.0	.3	2.0	.05
3	2.0	.2	1.0	.0005
4	3.0	.05	3.0	.025
5	-	.05	-	.025
6	-	.05	-	.005
7	-	-	-	.005
8	-	-	-	.005

Table 4.6: Tasks 1-8. Best parameters for expected future performance.

Task	Algorithm			
	bci ($z_{\alpha/2}$)	rc (α)	bie ($z_{\alpha/2}$)	npie ($\alpha/2$)
1	1.0	.6	2.0	.05
2	4.0	.2	3.0	.05
3	4.0	.3	3.0	.05
4	1.0	.4	1.0	.0005
5	-	.6	-	.025
6	-	.4	-	.0005
7	-	-	-	.005
8	-	-	-	.005

Table 4.7: Tasks 1-8. Best parameters for performance during learning.

$100\rho_g^*\%$ of the time. This gives us a measure of how the algorithm performs with regard to exploration for future exploitation.

Thus for each learning algorithm two values were recorded at each time step. First the average reward generated over the learning period so far was recorded each tick. Secondly, the confidence in the policy believed by the agent to be the best policy, (i.e. the greedy policy) was recorded each tick. For algorithms which are parameterised the best performance observed over a range of parameter values is the one presented here. The best parameter values found for each algorithm and each performance criterion are summarised in Tables 4.6 and 4.7.

The best algorithm with respect to exploration for future exploration is defined to be the algorithm with the highest confidence in its greedy policy at the end of the learning run (see Table 4.8). The best performing algorithm with respect to the exploration-exploitation trade-off is defined to be the algorithm which generates the highest observed average reward on the final step of the learning run (see Table 4.9).

Performance over the entire learning run is shown in the form of graphs (see Figures 4.13– 4.28). Each page shows the graphs for a single task. The top graph shows the confidence in the greedy policy (in %) plotted against time (in ticks). It can be seen that for all algorithms on all tasks confidence in the greedy policy rises through time. The bottom graph on each page shows the average reward generated against time (in ticks). Each point marks the average reward per tick generated since the first tick. This is why the initial performance is erratic, and why the long term performance is almost invariant in some cases. For the plots of average reward, the vertical axis is scaled between the expected rewards for the optimal and worst policies. In Figure 4.20 for example the average reward must fall between .1 and .2, whereas in Figure 4.22 it falls between .2 and .8. This makes it easy to see whether or not the average reward generated by an agent is better than random. The random and optimal performance levels for average reward are also recorded in the far right of Table 4.9.

Finally the results are summarised for each criterion and task in the form of graphs depicting significant dominance partial order among the algorithms (Figures 4.8— 4.11). In each diagram the dominant algorithms are placed at the top of the graph.

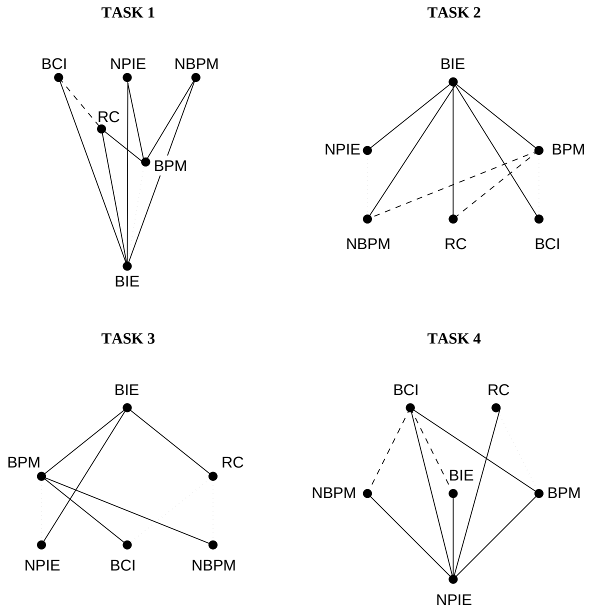


Figure 4.8: Significant dominance partial order among algorithms for Tasks 1-4 with regard to confidence in the greedy policy.

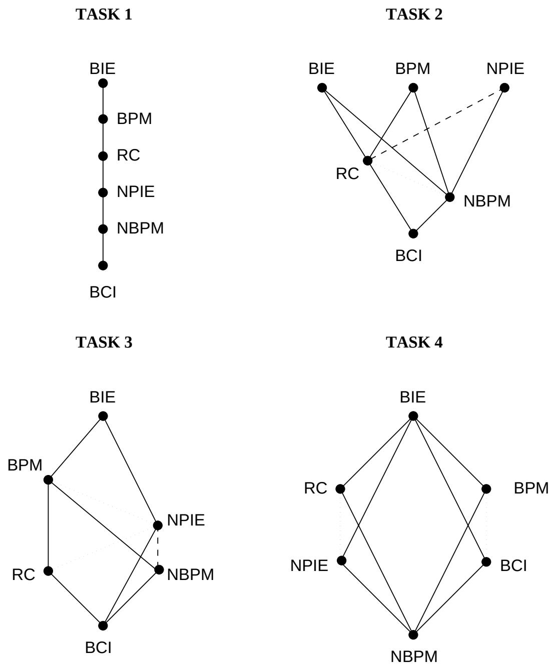


Figure 4.9: Significant dominance partial order among algorithms for Tasks 1-4 with regard to average reward generated.

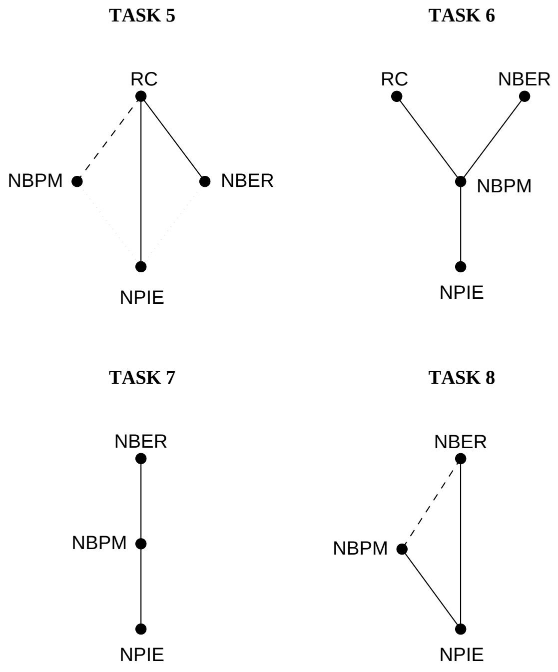


Figure 4.10: Significant dominance partial order among algorithms for Tasks 5-8 with regard to confidence in the greedy policy.

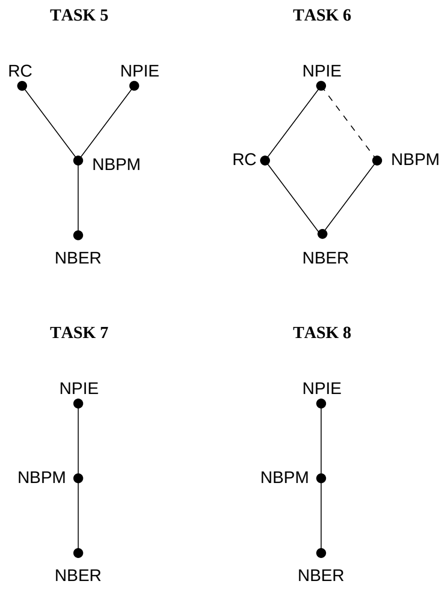


Figure 4.11: Significant dominance partial order among algorithms for Tasks 5-8 with regard to average reward generated.

Task	Algorithm						
	bpm	nbpm	nber	bci	rc	bie	npie
1	99.2112	99.9970	-	99.9999	99.9992	99.0483	99.9978
2	83.6780	82.1163	-	82.5664	82.2763	85.7645	83.1693
3	89.4727	87.0170	-	86.9762	88.3954	92.2421	88.0836
4	84.5259	85.0132	-	86.3487	85.6726	84.9922	80.7467
5	-	99.9968	99.9968	-	99.9971	-	99.9965
6	-	97.8493	99.5592	-	99.5109	-	96.1557
7	-	99.9361	99.9968	-	-	-	99.5442
8	-	92.3941	94.6218	-	-	-	88.3514

Table 4.8: Tasks 1-8. Confidence in greedy policy on final tick of run, averaged over all runs.

Task	Algorithm								
	bpm	nbpm	nber	bci	rc	bie	npie	random	optimal
1	.7703	.6247	-	.5074	.7477	.7888	.7276	.5000	.8000
2	.5178	.5077	-	.5018	.5111	.5191	.5159	.5000	.5500
3	.8721	.8631	-	.8425	.8632	.8803	.8678	.8500	.9000
4	.1712	.1615	-	.1675	.1689	.1766	.1682	.1500	.2000
5	-	.7630	.5205	-	.7886	-	.7896	.5000	.8000
6	-	.5399	.4985	-	.5390	-	.5424	.5000	.5500
7	-	.8407	.5291	-	-	-	.8720	.5000	.9000
8	-	.5748	.5463	-	-	-	.5831	.5000	.6000

Table 4.9: Tasks 1-8. Average reward generated over duration of run, averaged over all runs

An edge between two nodes denotes a significant difference between those two nodes. A solid line denotes a significant difference at the 1% level or above; a dashed line denotes a significant difference at the 5% level; and a dotted line denotes a significant difference at the 10% level. If a line is not marked directly between two algorithms A and C, then a significant difference between A and C may still be inferred if significant differences are marked both between A and B and between B and C. The significance test employed was a two-tailed t-test. It can be quickly seen from these figures that many of the differences recorded were significant.

4.6.5 Discussion

Tasks 1–4

With regard to the average reward criterion, performance is dominated by the BIE algorithm. This is similar to the result reported by Kaelbling [37]. On every single task it generated an average reward higher than that for any other algorithm. These differences were significant on every task apart from Task 2. This is what we would expect, since no other algorithm tested was designed to perform well on this criterion. The ordering of the other algorithms according to this criterion was also fairly consistent across the tasks. It can be seen that on every task the second highest average reward was generated by the BPM algorithm; followed by the NPIE and RC algorithms — which performed at about the same level. The NBPM algorithm was worse still; with the BCI algorithm bringing up the rear. The BCI algorithm performs very poorly on all tasks with respect to this criterion. All algorithms except the BCI algorithm are expedient (better than random) on all tasks. On Task 3, it actually performs worse than a random agent.

Of the other algorithms the NBPM algorithm is also a poor performer. This is simply due to the fact that a uniform sampling policy was employed prior to the agent generating 30 trials of each action. The probability matching mechanism for non-boolean rewards must have a large sample as noted previously. Thus the average reward generated is comparatively poor, particularly over a run as short as 100 ticks. Finally, the RC algorithm performed well; its best performances occuring over a moderate range of learning rates (see Figure 4.7). In Task 2 the best learning rate was .3, whereas for Task 1 it was .6. This difference is explained simply by the fact that high plasticity on a hard task (e.g. Task 2) can lead to misconvergence. On an easy task (Task 1) the plasticity can be high because the probability of generating misleading observations is low. It is also important to note that the RC algorithm's performance was not robust across the parameter space.

With respect to the confidence criterion, the results are more mixed. First, no one algorithm consistently outperforms the others. On Tasks 1 and 4 the BIE algorithm performs poorly, whereas on Tasks 2 and 3 it is the best performer. Why is this? The cause is the absolute difference between the rates of success on each arm. Where the absolute difference is large an interval estimation method will converge rapidly. With respect to increasing confidence in the greedy policy this is a poor strategy. In the most extreme case (Task 1) the BIE method optimises the exploration-exploitation trade-off so well that it hardly tries arm a_2 at all. The consequence is that on tasks with a large absolute difference in the rate of success the IE algorithm naturally converges rapidly, performing well according to its design criteria; and hence performing badly according to maximising certainty about the optimal policy.

This trend is reflected in the parameter values that optimise the RC algorithm for the different criteria. For the average reward criterion the best performing values of α are generally high, whereas for the expected future performance criterion the best values are generally low. The low learning rate implies an even sampling policy for as long as possible. In tasks with a low ratio between the rates of success for the best and worst actions, the natural rate of convergence is lower, and so a slightly higher α value is better.

The most striking result overall is that difference between successfully optimising the exploration-exploitation trade-off and optimising exploration for future exploitation. While on Tasks 2 and 3 the ordering of algorithms is similar over both criteria; on Tasks 1 and 4 the ordering is radically different. It can be concluded that the two criteria are not always satisfied by the same behaviour.

Tasks 5–8

Again, on the average reward criterion the interval estimation method used (NPIE) does best, followed by the probability matching method (NBPM). The NBER method does badly on all tasks according to this criterion. This is also what we would expect to see. On the expected performance criterion the NBER algorithm performs best. On Task 6 it is outperformed by one of the parameter settings for the RC algorithm, but on the more difficult 5 arm tasks it outperforms both the probability matching method and the interval estimation method. This again is what we would expect. The entropy reduction method makes cost savings over other methods when it has a large number of actions to choose from. In the 2 arm case it effectively degenerates to a

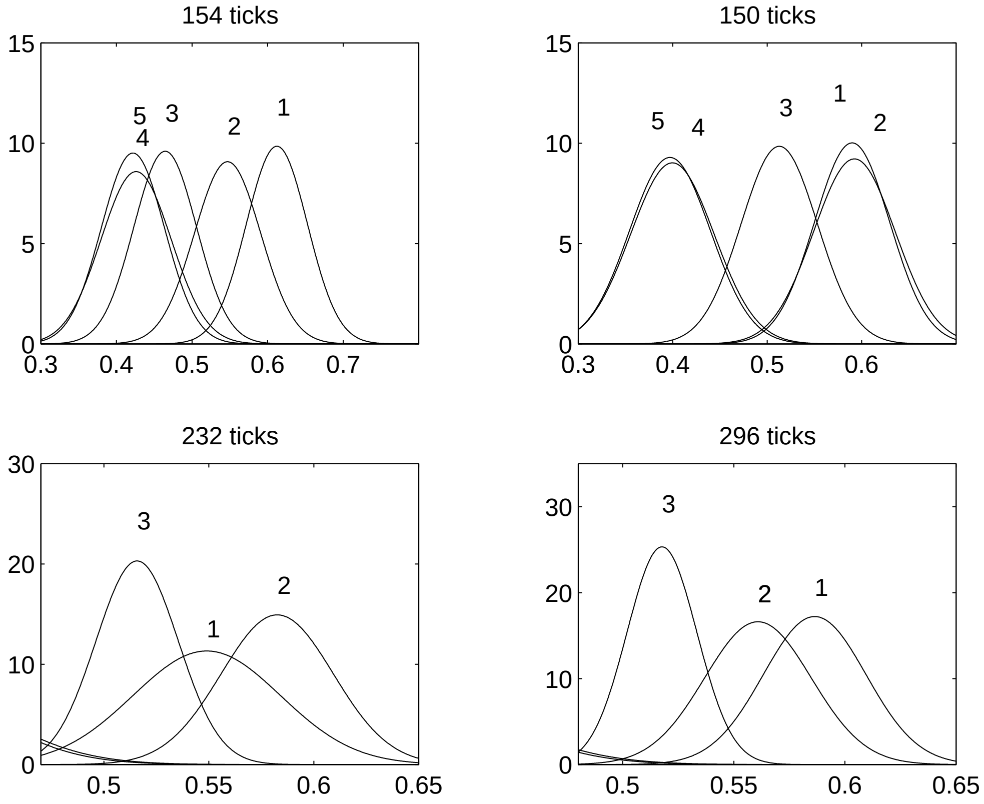


Figure 4.12: Distributions of mean reward for the NBER agent on Task 12. The plots are denoted as follows. (a) upper left; (b) upper right; (c) lower left; (d) lower right.

confidence interval method, like the BCI method used for the boolean tasks. When there are more than two arms it improves convergence speed by heavily selecting arms which are likely to be optimal. A method such as the NPIE algorithm only tries the action it thinks most likely to be best. This may not be an action about which there is much uncertainty. It is easy to imagine, for example, an action which is not as good as the best action, but for which the confidence interval is broader than that for the best action. In such an instance the NBER algorithm will take the second best action because it will lead to a greater increase in certainty about the identity of the optimal policy. This is shown by data from actual runs of the algorithm (see Figure 4.12). In panel (a) the agent believes action a_1 to be optimal, but selects action a_2 because the confidence interval on the estimate of its mean reward is wider. Thus taking action a_2 will lead to a greater reduction in entropy.

The algorithm however, can be mislead. In panels (b)-(d) of Figure 4.12 an example of this is depicted. In panel (b) action a_2 looks much better than it actually is. In this case it's mean reward is very close to that of action a_1. To the agent it appears that if either action a_1 or a_2 were taken then the entropy reduction would be negligible. In this case it takes action a_3 as the best alternative. It carries on taking action a_3 until the ρ_3^* is small (panel (c)). It then selects either action a_1 or a_2. This process repeats until actions a_1 and a_2 become disambiguated (panel (c)). Between panel (b) and panel (c) — a duration of 82 ticks — the agent selected action a_3 53 times, a_2 23 times, and a_1 just 6 times. Note that the mean rewards have actually become more misleading during this period — a_1 has an observed mean reward of just .55, whereas its true mean reward is .6. By this stage a_1 looks like generating the greatest reduction in entropy, and is selected. Over the next 64 ticks the agent selects a_3 26 times, a_2 8 times, and a_1 30 times. As a_1 is sampled more frequently its observed mean reward moves closer to its actual mean reward. The reason a_3 is selected so frequently in this phase is that at some point a_1 and a_2 appear to be equally good. Thus the situation depicted in panel (b) occurs once more. This time, however, the observed mean reward for a_1 converges to the actual mean reward. In consequence a_1 is believed to be better than a_2 and is no longer 'trapped', (panel (d)). Although the NBER method uses extra trials taking an action that is not very likely to be optimal on the basis of its observations, it does eventually right itself. Although the use of an approximation (Equation 4.12) means the algorithm carries no guarantee of convergence, it was not observed to become stuck in any of the runs reported here.

There are two additional points worth making. First, the RC algorithm is limited in its usefulness because it is not easily extended to tasks involving more than 2 actions. An extension of the more general AHC algorithm has been investigated by Lin [43], but it has a large number of parameters, making it difficult and expensive to optimise. It was not employed here for that reason. Finally, as with the boolean tasks, the non-boolean tasks demonstrate clearly that there is an important difference between optimising the exploration exploitation trade-off and optimising exploration for identifying the optimal controller.

4.7 Extensions

There are a number of ways in which this work could be extended. First, it would be useful to more closely analyse the behaviour of the algorithms presented, particularly with regard to the boolean tasks. Second, it would be sensible to investigate cheaper methods for approximating $\Pr(a = a^*)$. The use of generalisation techniques could reduce the evaluation time for the probability matching and entropy reduction methods, from polynomial to linear time. The accuracy and flexibility of such methods would be the key questions to answer. The use of such methods might also allow the principled extension of the boolean method to more than two actions.

Third, it is essential for any practical purpose to be able to extend the measures developed here to multi-state tasks. One way to do this would be to apply the measures developed directly to return rather than reward. As such the boolean case would have no obvious equivalent. The non-boolean method, however, should be extendible, on the proviso that a good estimator of the variance in return can be constructed. Fourth, there are currently practical limitations to the entropy based method in the case where two actions are in fact as good as one another. It would therefore be an essential extension to incorporate some notion of the importance of different reinforcement magnitudes. The use of the measures developed for verification could also be usefully extended to multi-state tasks.

4.8 Conclusions

This chapter has taken a different approach to the problem of controlling exploration when learning from reinforcement. First, measures have been developed for the probability of optimality for processes generating either boolean or non-boolean reinforcement. These have been used in turn to develop simple probability matching methods for controlling exploration. A proof has been presented that at least one of these methods is guaranteed to converge to the optimal policy in the limit. An entropic measure of task knowledge has also been developed. This can be used to guide exploration for future exploitation.

In an empirical study it has been shown that the exploration policy required to optimise confidence in the greedy policy can be radically different from that required to optimise average reward received. Finally the methods developed in this thesis have been shown to outperform existing methods with respect to controlling exploration for future exploitation.

Finally the ρ_i^* also give a method for verifying the probability that the greedy policy is optimal on the basis of the observations so far. This is a useful property because as noted in Section 1.3 learning methods are effectively only verified in the limit. By being able to state the probability that the greedy policy is the optimal policy we effectively give as good a form of verification for a model-free controller as is possible after a finite number of trials. Such a method can be used to analyse controllers after they have been learned. In this chapter the method has only been demonstrated for single state tasks. In principle it may be extended to problems with more than one state. In order to extend such methods to multi-state tasks, however, a good estimator of the variance in delayed reward must be derived.

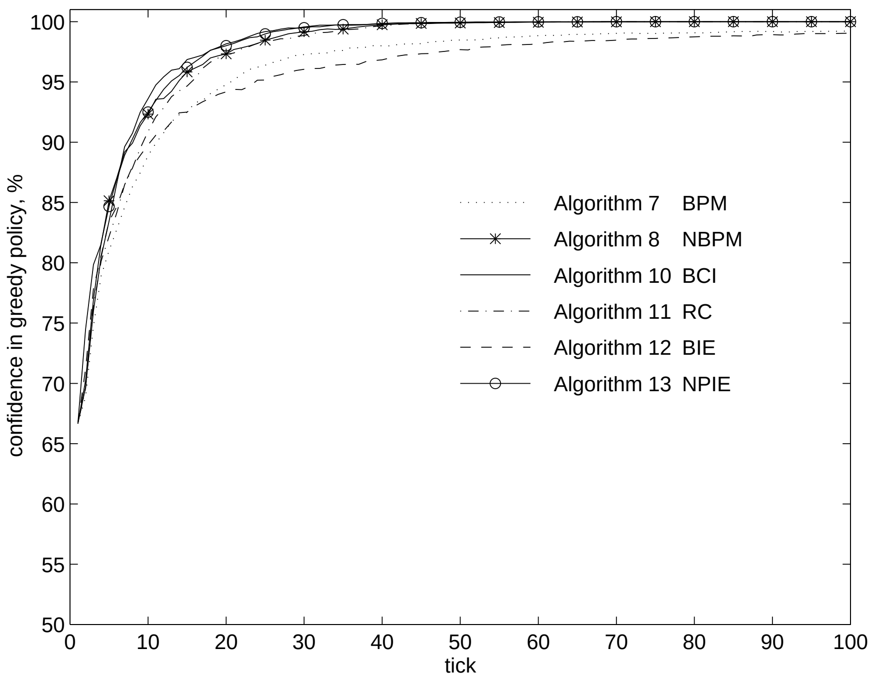


Figure 4.13: Task 1. Confidence in the greedy policy. Averaged over 200 runs.

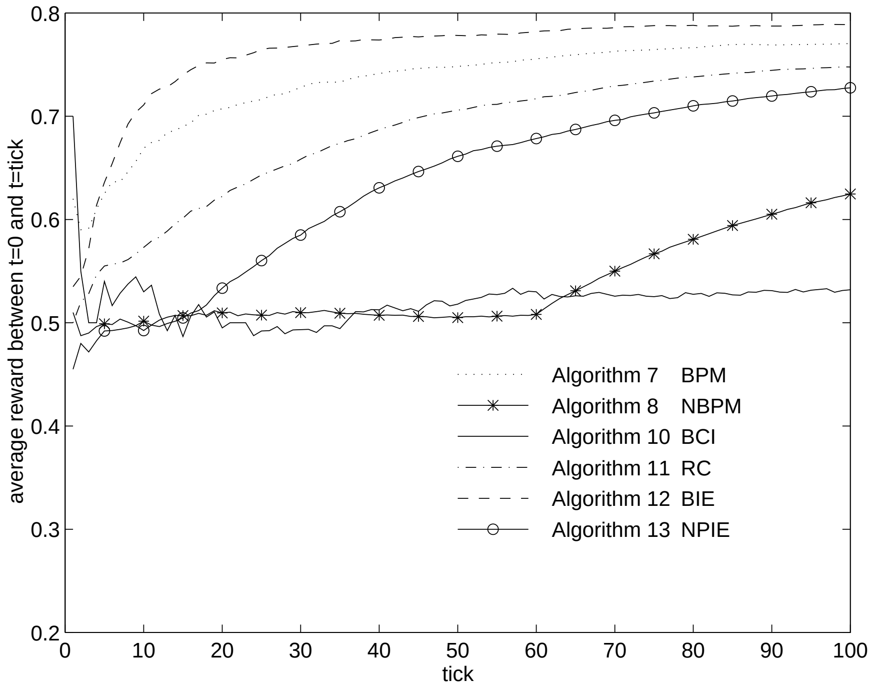


Figure 4.14: Task 1. Average reward generated over run. Averaged over 200 runs.

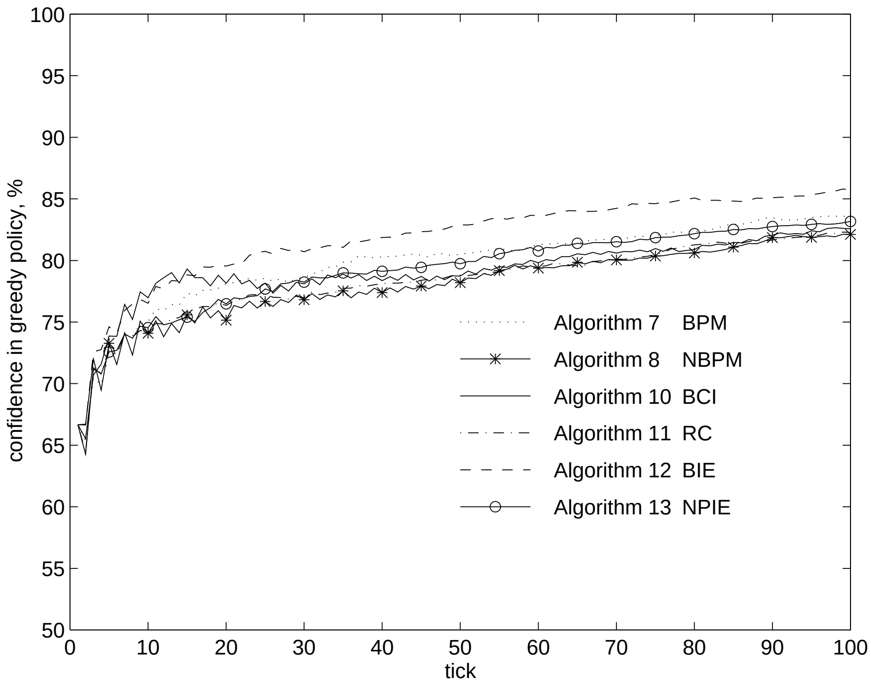


Figure 4.15: Task 2. Confidence in the greedy policy. Averaged over 1000 runs.

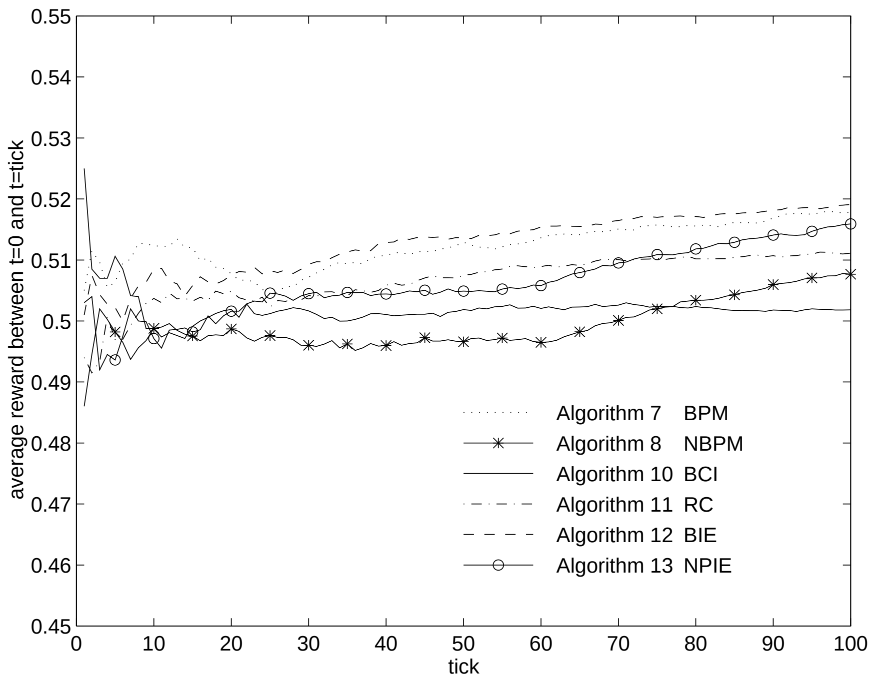


Figure 4.16: Task 2. Average reward generated over run. Averaged over 1000 runs.

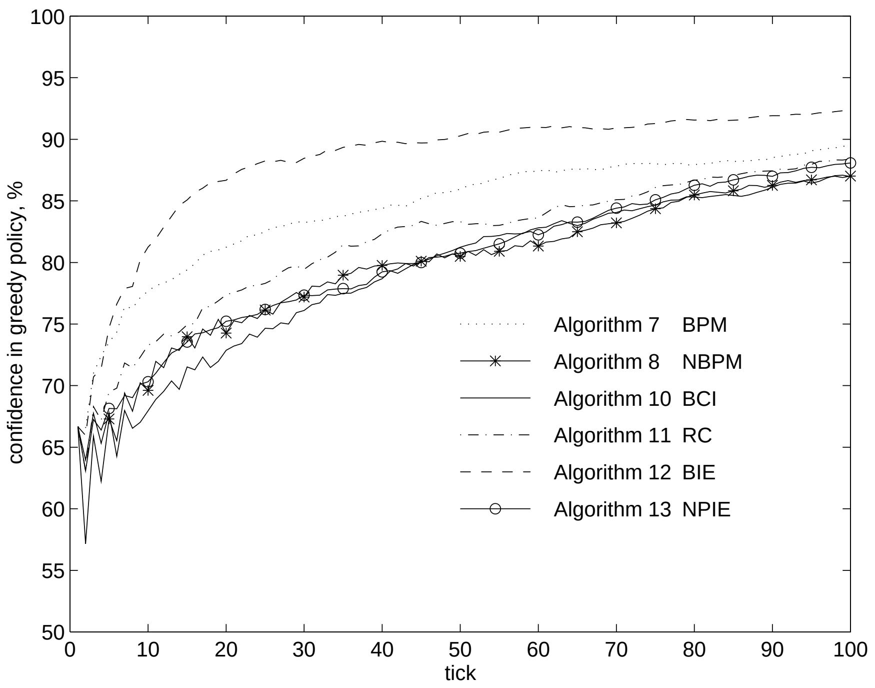


Figure 4.17: Task 3. Confidence in the greedy policy. Averaged over 400 runs.

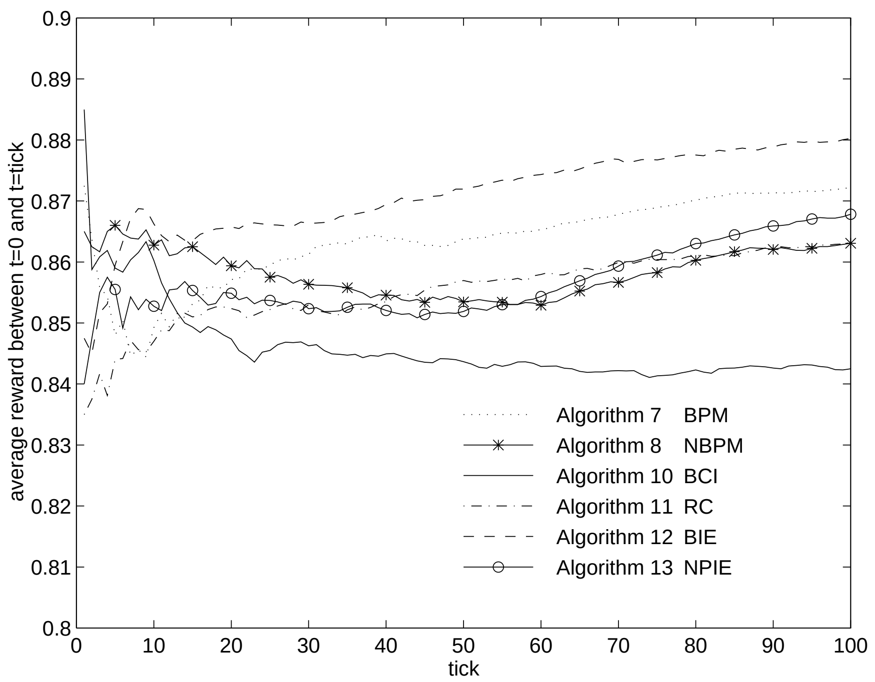


Figure 4.18: Task 3. Average reward generated over run. Averaged over 400 runs.

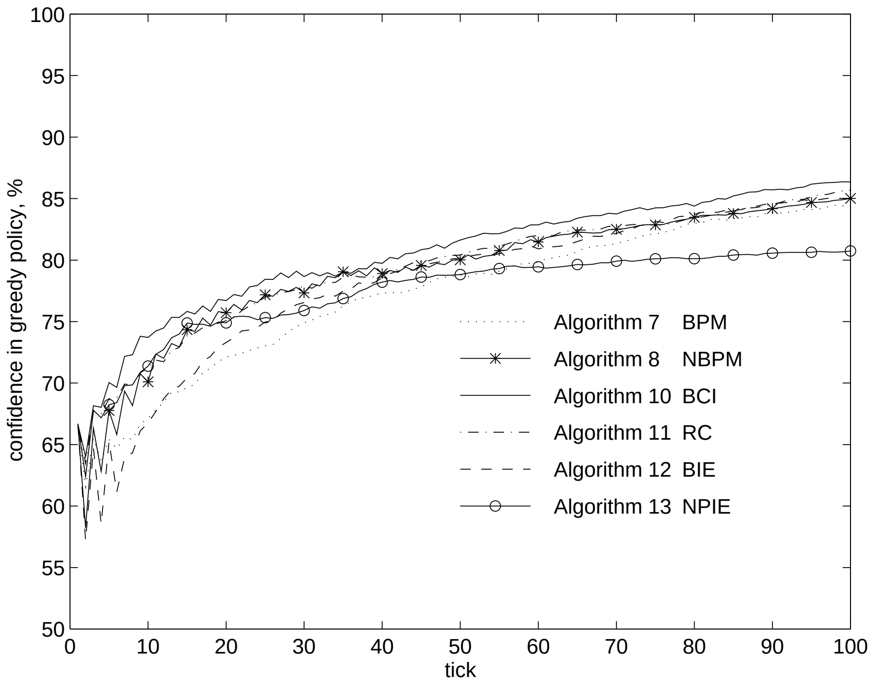


Figure 4.19: Task 4. Confidence in the greedy policy. Averaged over 1000 runs.

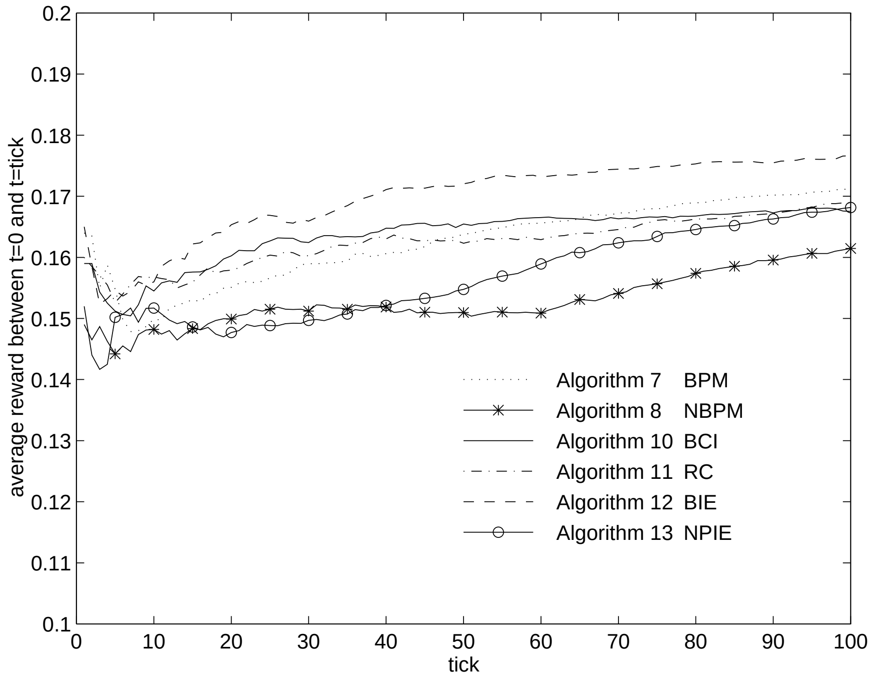


Figure 4.20: Task 4. Average reward generated over run. Averaged over 1000 runs.

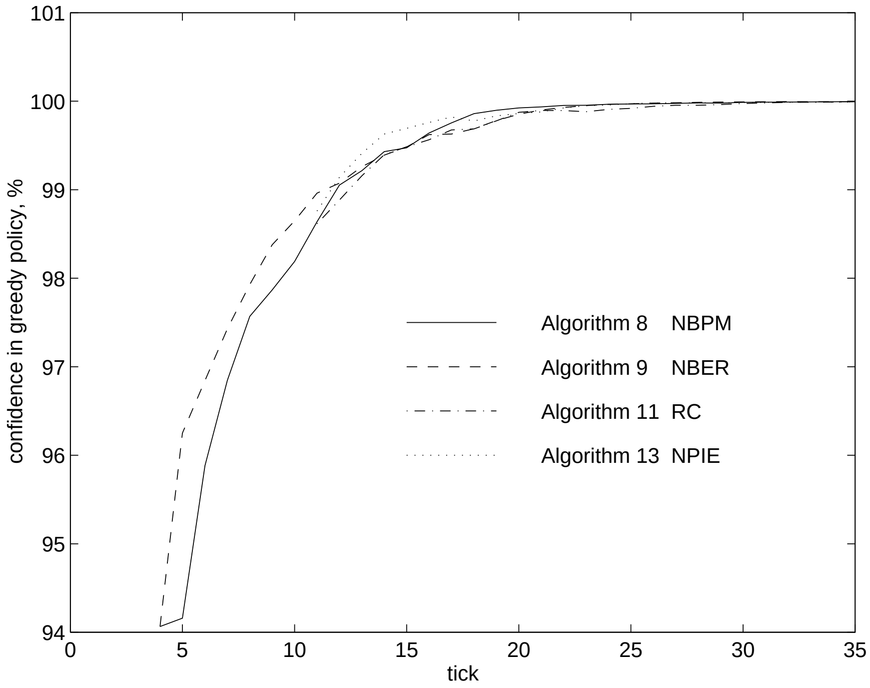


Figure 4.21: Task 5. Confidence in the greedy policy. Averaged over 200 runs.

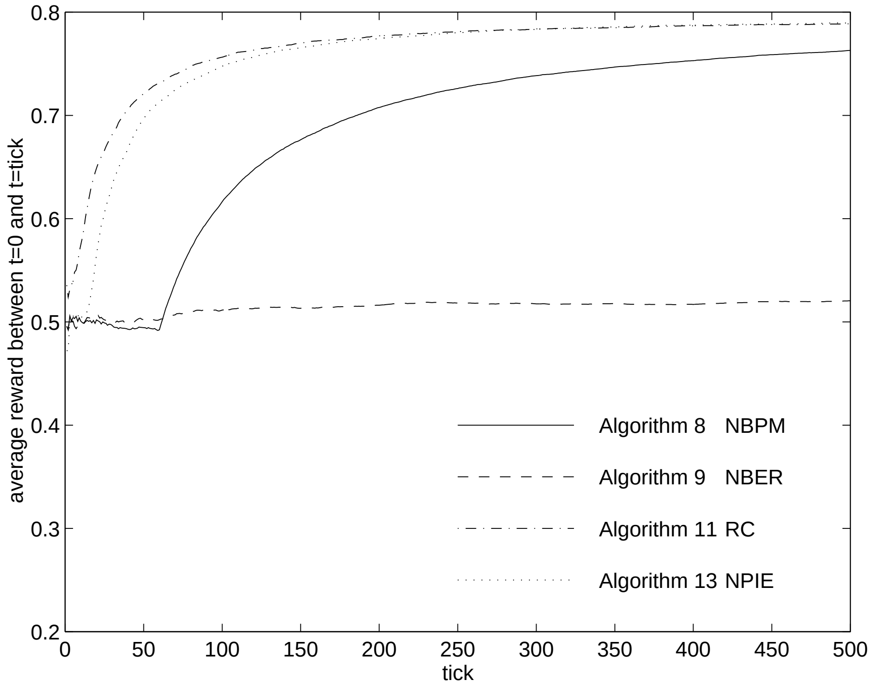


Figure 4.22: Task 5. Average reward generated over run. Averaged over 200 runs.

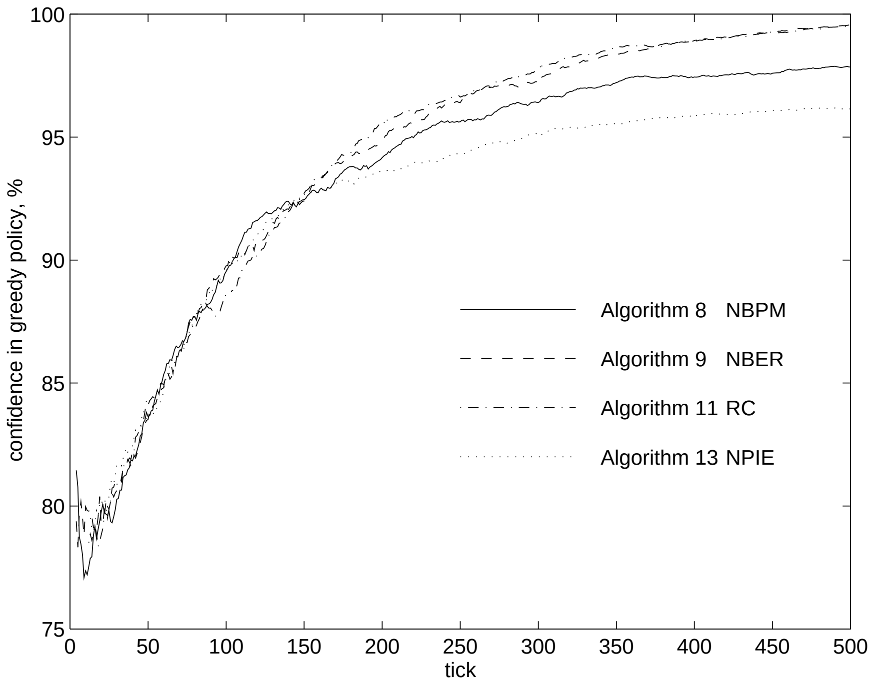


Figure 4.23: Task 6. Confidence in the greedy policy. Averaged over 200 runs.

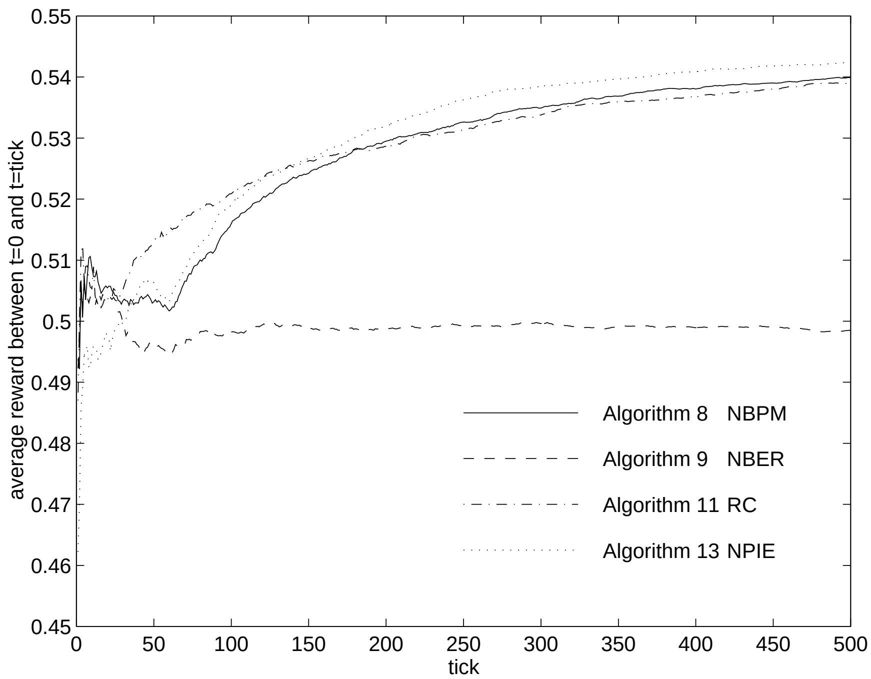


Figure 4.24: Task 6. Average reward generated over run. Averaged over 200 runs.

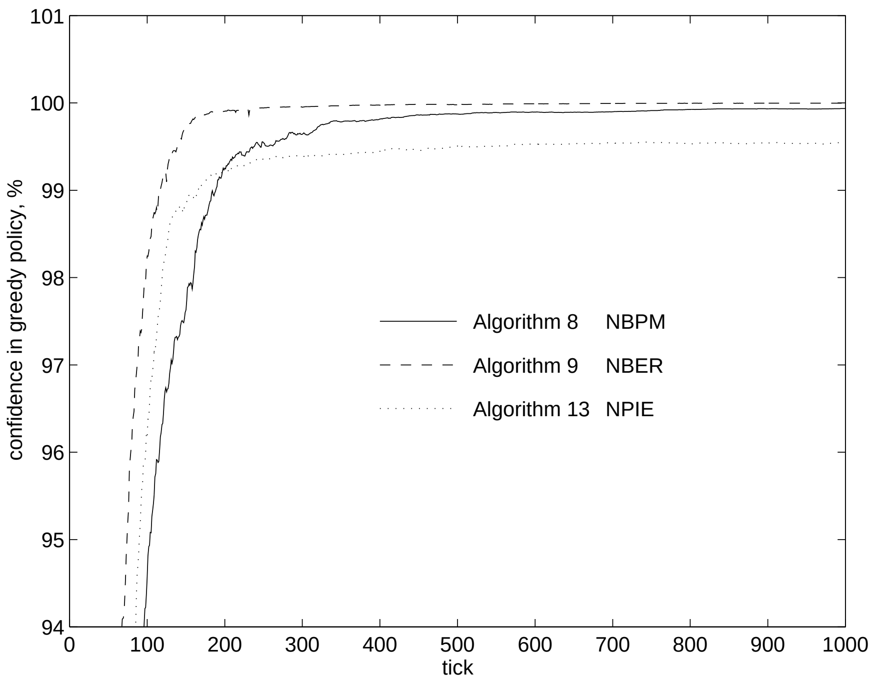


Figure 4.25: Task 7. Confidence in the greedy policy. Averaged over 200 runs.

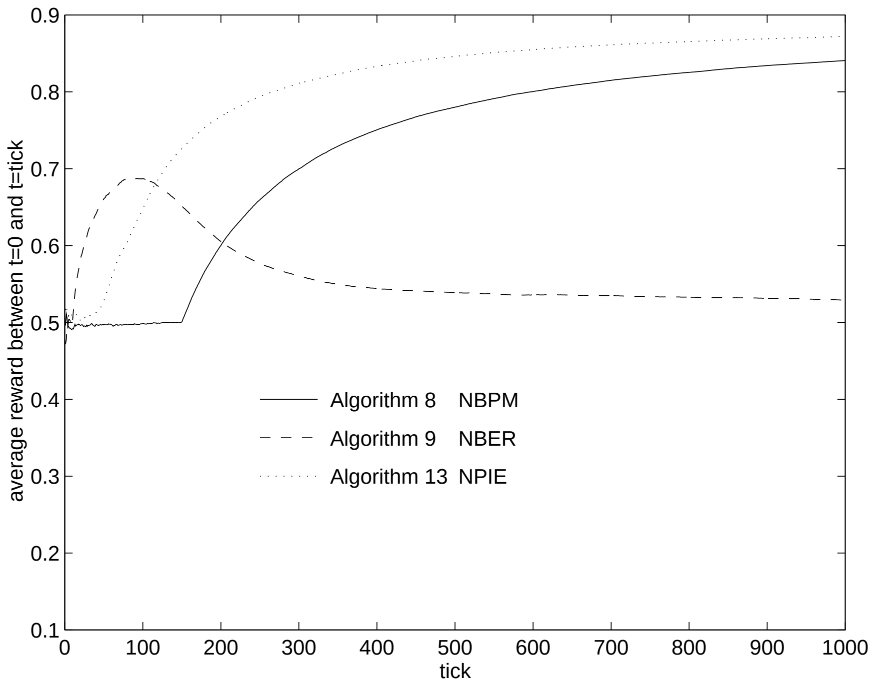


Figure 4.26: Task 7. Average reward generated over run. Averaged over 200 runs.

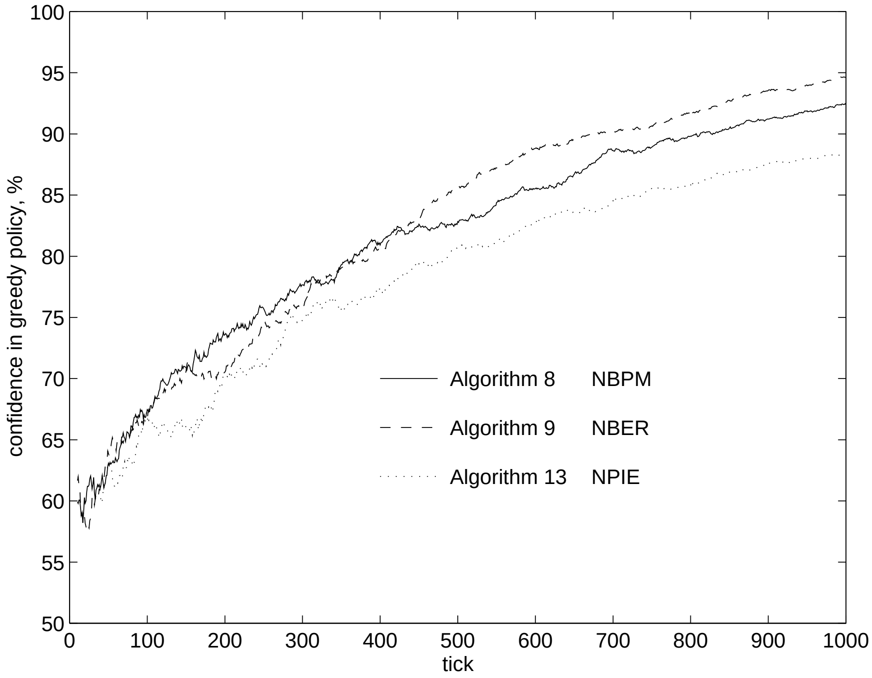


Figure 4.27: Task 8. Confidence in the greedy policy. Averaged over 200 runs.

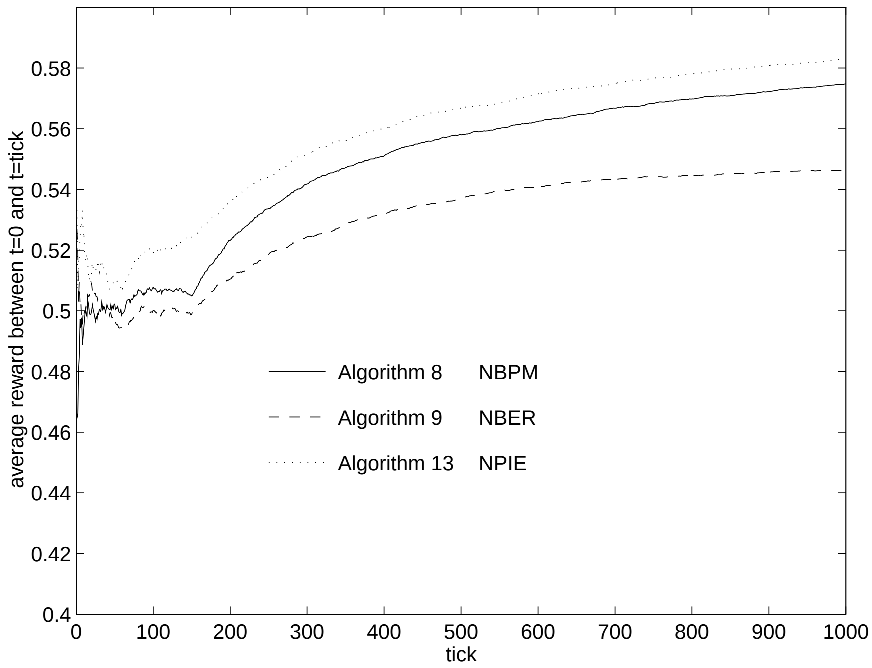


Figure 4.28: Task 8. Average reward generated over run. Averaged over 200 runs.

Chapter 5

Inference

As detailed in Chapter 2 there are a variety of methods for learning control from reinforcement. Model-free methods are generally regarded as using computation efficiently each step, but can take a large number of steps to converge. Model-based methods are generally regarded as being computationally more expensive each step, and as converging after fewer observations. Because this thesis focuses on the idea of exploration as an inference problem, it is important that we understand the characteristics of each approach. Consequently this chapter presents some work investigating the behaviour of some model-free and model-based methods on a number of tasks. First, the behaviour of a well known model-free method, Q-learning, is examined. The effects of extreme deviations from the greedy policy on two forms of Q-learning are investigated. Second, the inferential power of Q-learning is compared with that of prioritised sweeping on a number of tasks.

5.1 Investigating the behaviour of Q(λ)

There are a number of different forms of eligibility traces which may be used with Q-learning. Peng and Williams [60] used traces in which $\lambda_t > 0, \forall t \in T$. Such a formulation provides rapid credit propagation but makes the algorithm exploration sensitive, i.e. its estimates of the Q-values are corrupted when the agent deviates from the greedy policy. Watkins [89] suggested the use of traces where $\lambda_t = 0$ if the agent deviates from the greedy policy, thus curtailing the traces and preventing the corruption of $\hat{Q}$-values, but reducing the extent of credit propagation. This work investigates

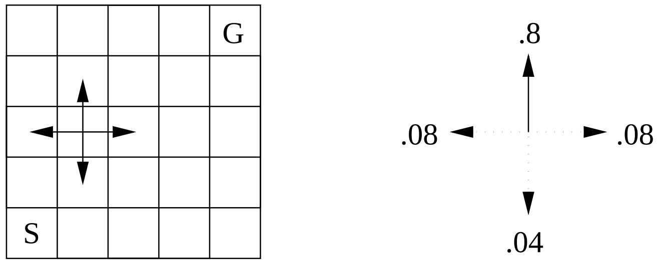


Figure 5.1: The solid arrows in the grid represent the most likely directions of travel for each of the four actions. On the right the solid arrow represents the most likely direction of travel for one action and the dotted arrows represent the other probabilities. The probability distributions for other actions may be derived by a series of 90° rotations.

the exploration sensitivity of both versions of Q(λ) more closely by measuring their performance in two cases. First, when the agent consistently deviates from the greedy policy. Second, when the rate of deviations from the greedy policy is relatively low. In Experiment 5.1 the behaviour of Peng and Williams' algorithm is examined. In Experiment 5.2 the behaviour of Watkins' rule is examined. I shall refer to the two different kinds of eligibility traces as *uncorrected* and *corrected* traces respectively.

5.1.1 Experiment 5.1

Method

Uncorrected Q(λ) was tested on a Markov decision process represented in the form of a square grid of side 5 (Figure 5.1). In each state there are four possible actions, each with stochastic transitions; the start and goal states being in diagonal corners. The goal state is absorbing and on entry generates reinforcement of 1. All other states carry reinforcement of 0. In order to generate deviations from the greedy policy a local model-based counter-driven exploration method was used. At each step the action selected was determined by maximising the counter-based measure given by Equation 3.8, where $\hat{E}[C_t(X_{t+1})|x_t, a]$ is estimated by Equation 3.9. The use of this model-based exploration method does not make Q(λ) a model-based inference method, because the $\hat{Q}$-value updates are still model-free. Model-based exploration was used solely to achieve persistent deviation from the greedy policy. The agent was also run on the same task using a semi-uniform exploration method[1] with the probability of the optimal action being .95, and a uniform distribution across all actions being taken otherwise. We used each of these learners with both accumulating and replacing traces.

For the counterbased exploration method it was found to be sufficient to take 10 runs of 20 trials each, and for the semi-uniform based exploration method we took 200 runs of 50 trials each[2]. The parameter values used are detailed in Table 5.1.

Parameter	**Values**
α	.2 .3 .4 .5 .6 .7 .8 .9
λ	.1 .2 .3 .4 .5 .6 .7 .8 .9 1

Table 5.1: Task 9. Parameter values for Algorithm 6.

Results

Figures 5.2 and 5.3 show the performance of each variant across a range of values of α and λ. Each line shows the performance of the algorithm at a particular α value. Performance was measured by calculating the total absolute error in the Q-values at the end of 20 trials[3]. This error measure is given by[4]:

$$error = \sum_{x,a} |Q(x,a) - \hat{Q}(x,a)| \tag{5.1}$$

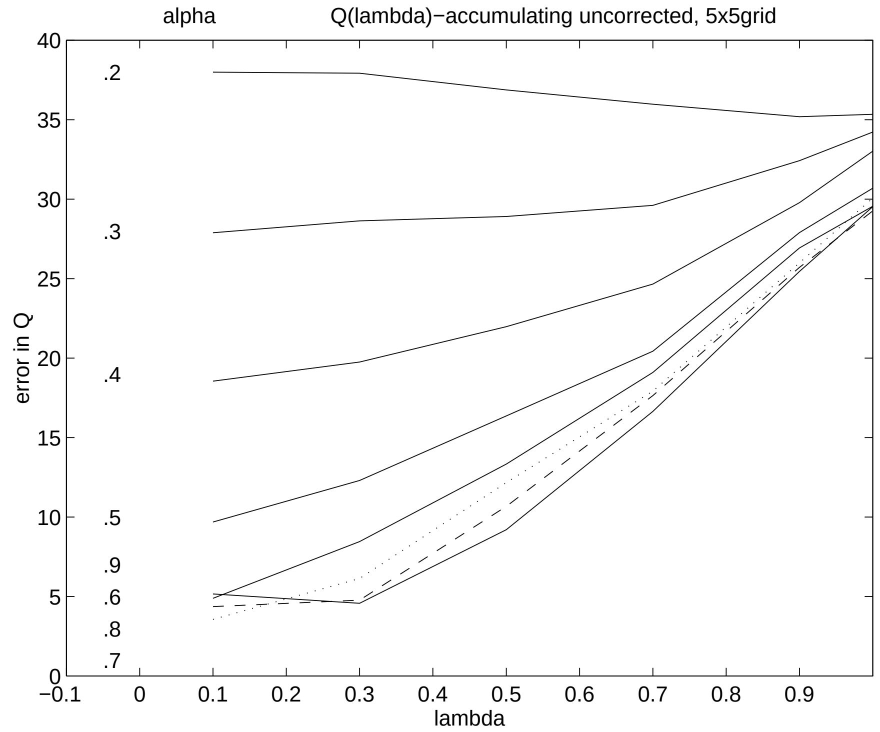
It can be seen that the difference in the performance of agents with accumulating and replacing traces is negligible for most values of λ, whether or not the agent deviates from the greedy policy. This is largely because for this task the rate of revisits to each

[1] We must deviate from the optimal policy with a finite probability each step. Otherwise the $\hat{Q}$-values will not converge. This is the exploration method which gives us long term convergence while staying reasonably close to the optimal policy.

[2] For the counter-based learner the performance of each parameter set stayed the same after this point. Because the agents with uncorrected traces were the only agents not to converge in the long run to zero error for at least some parameter values it was found better to run other agents for up to 50 trials. Also, in order to ensure the significance of the differences in performance other agents were run for 200 runs for each parameter set.

[3] This performance measure ignores the primary goal of any reinforcement learner, which is to maximise the performance of the greedy policy. In the next section the relationship between the accuracy of the Q-values and the utility of the greedy policy is examined in greater detail. It is also more usual to employ the Root Mean Squared Error in the Q-values as an estimate of their accuracy.

[4] The actual Q-values were obtained by running value iteration on the transition function and the reward function.



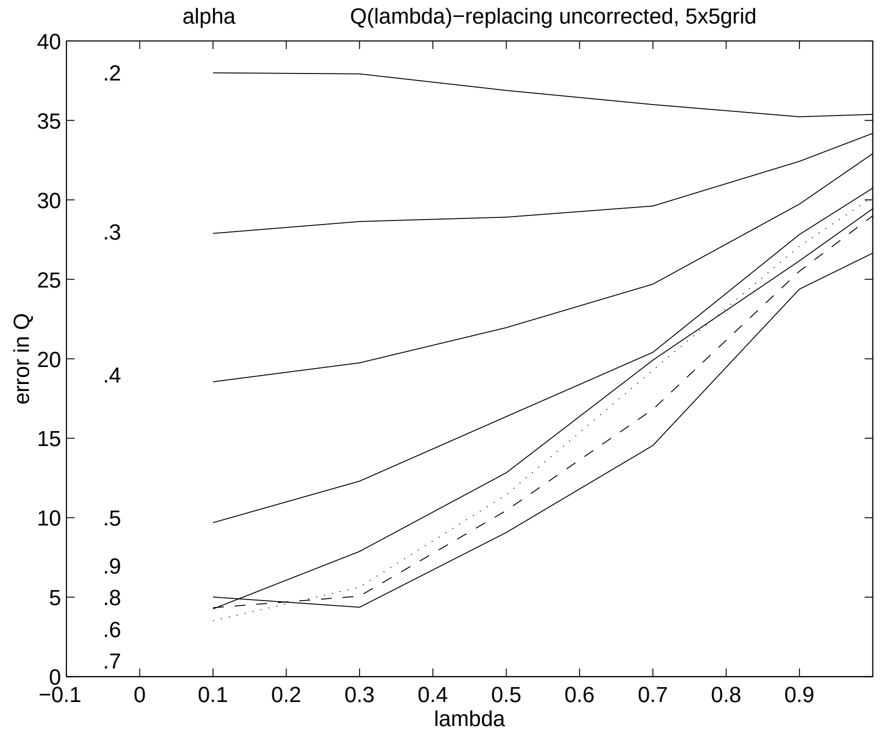


Figure 5.2: Performance of uncorrected $Q(\lambda)$ when deviating persistently from the greedy policy (pure counterbased exploration) on a 5x5grid. Upper plot shows accumulating traces; lower plot replacing traces.

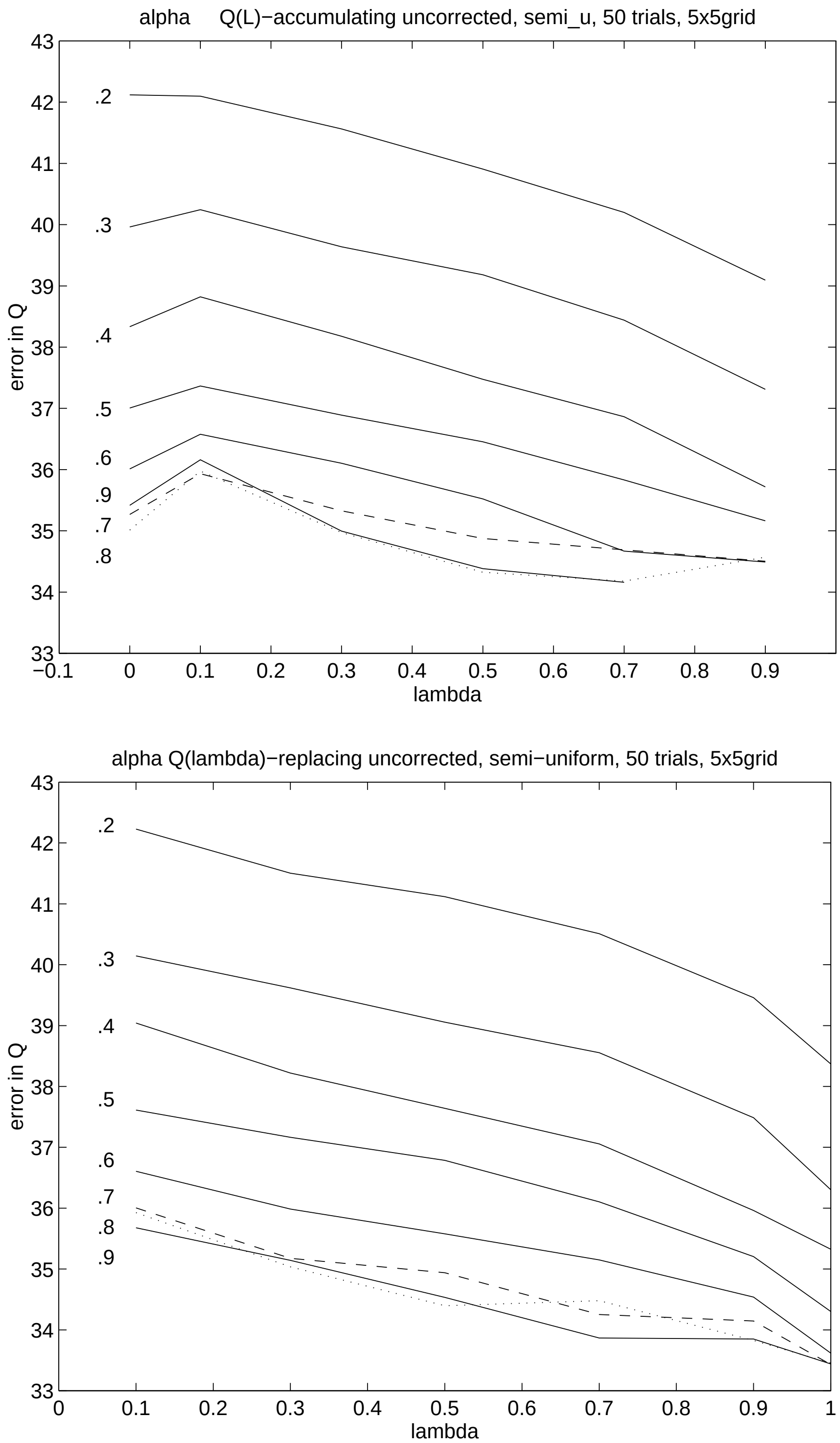


Figure 5.3: Performance of uncorrected $Q(\lambda)$ when deviating to a small degree from the optimal policy (using semi-uniform exploration with $P_{best} = .95$) on a 5x5grid. Upper plot shows accumulating traces; lower plot replacing traces.

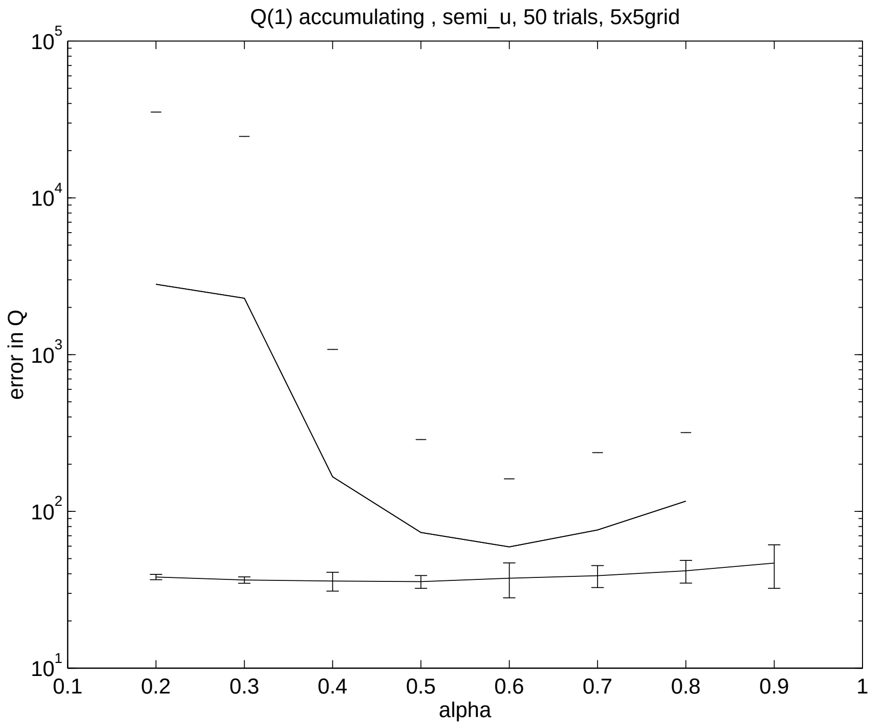


Figure 5.4: Performance of $Q(1)$ using both corrected (lower line) and uncorrected (upper line) accumulating traces with semi-uniform exploration on a 5x5grid. Because the y-axis is logarithmic it can clearly be seen that the variance across 200 runs is still so large that the differences are not significant.

state-action pair is relatively low, and consequently accumulating traces do not build up to a point where the estimated Q-values become unstable. In short the Monte Carlo every-visit estimate is close to the first-visit estimate. The significant difference is not between trace types, but between exploration methods. When the agent persistently deviates from the greedy policy the performance of both kinds of trace deteriorates rapidly as $\lambda \to 1$ (Figure 5.2). But, when the deviations from the optimal policy are small (the semi-uniform case: Figure 5.3) the accuracy continues to improve until λ reaches quite high values. Performance for replacing traces continued to improve all the way to $\lambda = 1$, whereas accumulating traces did produce unstable behaviour at $\lambda = 1$, as shown in Figure 5.4. Thus some difference between the performances of accumulating and replacing traces was observed.

5.1.2 Experiment 5.2

Method

From Experiment 5.1 it can clearly be seen that with uncorrected traces the $Q(\lambda)$ algorithm is exploration sensitive. Watkins points out that if λ is varied according to whether or not the agent follows the greedy policy, then exploration sensitivity can be avoided. To test this empirically the previous experiment was repeated, using Equation 5.2 to determine the value of λ. The important point is that when a non-greedy action is taken $\lambda = 0$, so that no previous Q-values are contaminated by the effects of experimentation.

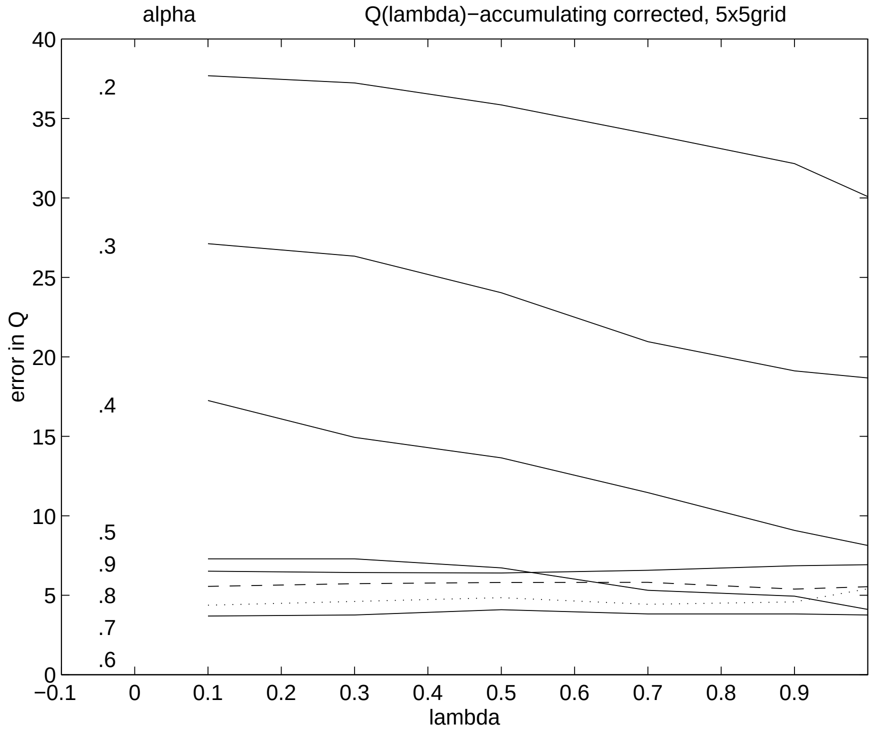
$$\lambda_t = \begin{cases} \lambda & \text{if } a_t = \arg\max_a(\hat{Q}(x_t, a)) \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

As in the previous experiment, 200 runs of 50 trials each were made for both exploration methods using the range of parameter values listed in Table 5.1.

Results

The results are shown in Figures 5.5 and 5.6. It can be seen that again the difference in performance between accumulating and replacing traces is negligible except at very high λ. By using corrected traces, however, we see marked improvement in performance over uncorrected traces. Specifically, when we deviate from the optimal greedy policy, performance improves as $\lambda \to 1$ rather than the other way round. The only exception to this is when we use accumulating traces with semi-uniform exploration and $\lambda = 1$. In this instance behaviour becomes unstable, although performance still appears to be an improvement over that of uncorrected traces[5]. The semi-uniform exploration method produced very similar performance to when it was used with the uncorrected algorithms. Consequently it may be concluded that at every point corrected traces either performed as well as or better than uncorrected traces with respect to minimising the total error in the Q-values.

[5] Unfortunately it was not possible to obtain significant results at this point, as can be seen in Figure 5.4.



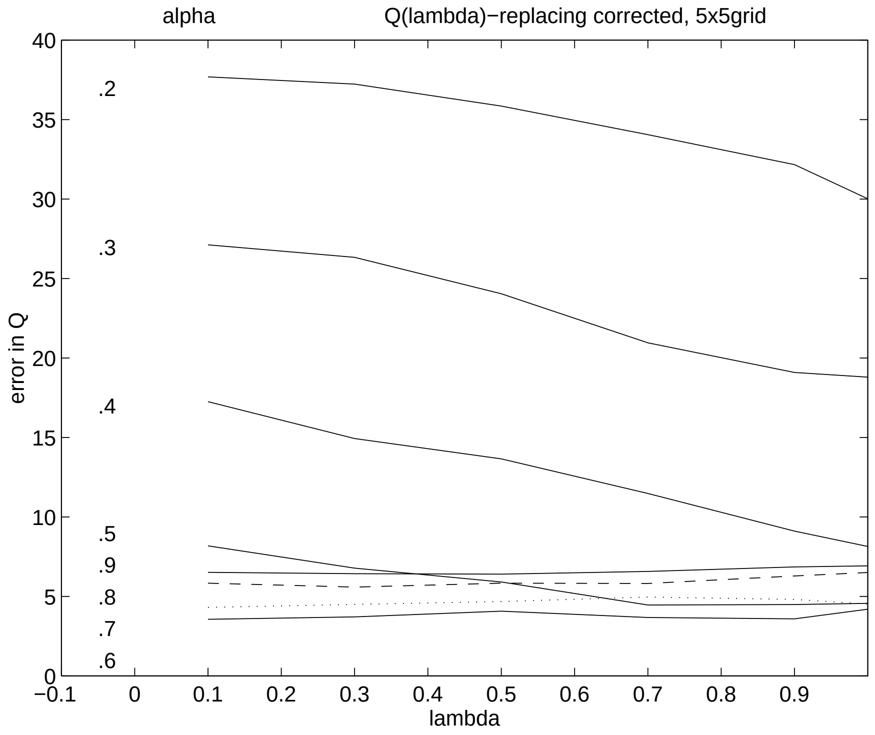


Figure 5.5: Performance of the corrected $Q(\lambda)$ with pure counterbased exploration on a 5x5grid.

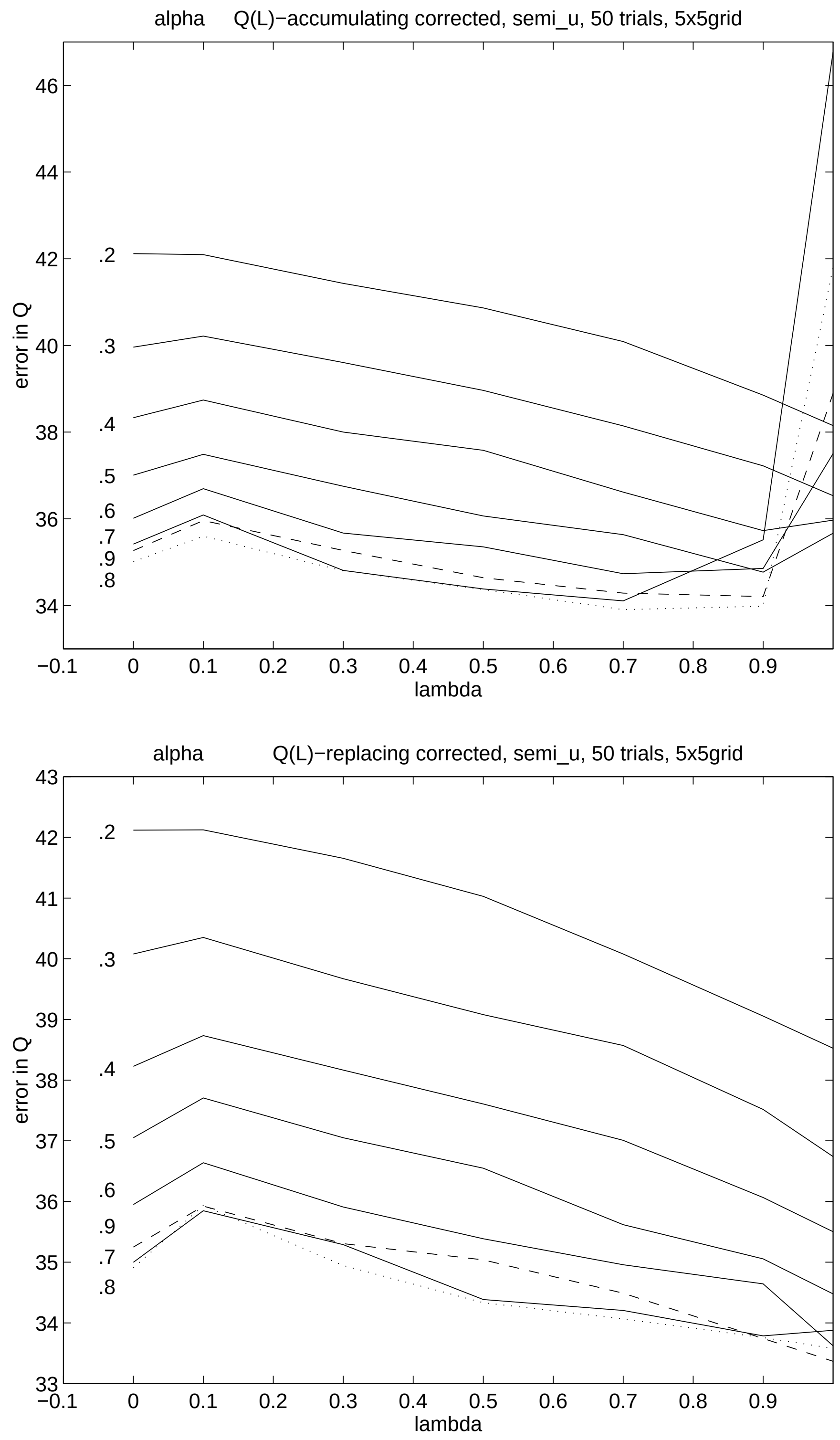


Figure 5.6: Performance of the corrected $Q(\lambda)$ with semi-uniform exploration on a 5x5grid. The performance of accumulating traces with $\lambda = 1$ is shown in Figure 5.4.

5.1.3 Discussion

These experiments have shown that using corrected traces improves performance of $Q(\lambda)$ with high λ when the agent deviates from the greedy policy. They do so by removing exploration sensitivity. This is important because the ability to explore an environment efficiently requires such a property. Eligibility traces are essential for fast credit propagation. These results are, however limited in their scope. The measure of performance used was the absolute error in the estimated Q-values. The error in the Q-values may in fact be high, and yet still specify an optimal or near-optimal policy. Thus, as observed by Peng and Williams, although uncorrected traces may not converge to the correct estimates they may generate a near optimal policy more quickly than corrected traces, when deviations from the optimal policy are not persistent. This problem requires further investigation. Specifically it will be necessary to examine the quality of the greedy policy using each kind of trace, on a variety of larger problems and under a number of different exploration schemes.

5.2 Model-Based Versus Model-Free Methods

As previously stated one of the interesting questions in learning from reinforcement is the nature of the trade-off between model-free and model-based methods. In this section we compare the performance of the leading algorithms from each genre. From model-based learning we take prioritised sweeping (Algorithm 5, see Figure 2.9), and from model-free learning we take Q(λ) (Algorithm 6, see Figure 2.10). The performance of model-free and model-based methods has been compared previously by Moore and Atkeson [53]. In such work, however, the relationship between the available computation each tick and the performance of the algorithms has not been fully explored. In the first instance Moore and Atkeson [53] only implemented one step Q-learning. In one step Q-learning the amount of computation is constant each step. In $Q(\lambda)$ learning it may be varied. Thus when comparing $Q(\lambda)$ and prioritised sweeping it must be ensured that the actual maximum computation carried out in a single tick is taken into account. Comparing the average computation per tick is also useful, though not sufficient by itself .

In order to estimate these quantities, it is necessary to keep a count of the number of basic computations performed each tick. For convenience I shall refer to these as basic computations as tocks. All counted operations were deemed to have the same unit cost. The operations counted were: arithmetic operators; relational operators; and any operation for copying a value into a complex data-structure. While this is a crude indicator of computational load, it is less machine specific than the real or cpu-time taken each tick, while being more informative about the actual computational load of the algorithms on a given task than a complexity analysis. The number of tocks used each real observation by both algorithms is now detailed.

5.2.1 Computational cost per tick

Prioritised sweeping

For prioritised sweeping this depends on the number of backups carried out for each observation. For each individual backup there are two primary sources of cost: updating the $\hat{Q}$-values, and updating the priority queue $\mathcal{P}$. Updating the $\hat{Q}$-values for each state removed from the top of the queue costs:

$$2|\mathcal{A}_i| - 1 + 2\sum_{a\in\mathcal{A}(i)} |\mathrm{succs}(i,a)|$$

where $1 + 2\,|\mathrm{succs}(i,a)|$ is the cost of calculating $\hat{Q}(i,a)$; and $|\mathcal{A}_i| - 1$ is the cost of finding the maximum $\hat{Q}$-value from all the $\hat{Q}$-values in state i. The function $\mathrm{succs}(i,a)$ returns the set of states which have been observed to immediately succeed i, a:

$$\mathrm{succs}(i,a) = \{j : \hat{p}_{ij}(a) > 0\}$$

The cost of updating the priority queue is given by:

$$\sum_{i'\in\mathrm{preds}(i)} |\Upsilon(i',i)| + |\mathrm{preds}(i)| + \sum_{k=0}^{|\mathrm{preds}(i)|} \log_2(|\mathcal{P}| + k) \tag{5.3}$$

where $\log_2(n)$ is the cost of accessing a priority queue[6] of size n; and $\mathrm{preds}(i)$ is the

[6] This assumes that the implementation of the priority queue uses a heap (see [20]).

set of states which have been observed to immediately precede state i:

$$\text{preds}(i) = \{i' : \exists a : a \in \mathcal{A}_{i'} \wedge \hat{p}_{i'i}(a) > 0\}$$

and $\Upsilon(i', i)$ returns the set of actions under which the transition $i' \rightsquigarrow i$ has been observed:

$$\Upsilon(i, i') = \{a : a \in \mathcal{A}_{i'} \wedge \hat{p}_{i'i} > 0\}$$

Formula 5.3 is composed of three costs. $\sum_{i' \in \text{preds}(i)} |\Upsilon(i, i')| - |\text{preds}(i)|$ is the cost of finding the maximum $\hat{p}_{i'i}(a)$ across $\mathcal{A}_{i'}$ for all predecessor states. The second cost is $2\,|\text{preds}(i)|$, that of calculating $\hat{p}_{i'i}(a)\Delta > \varepsilon$ for each predecessor state[7]. The third cost is $\sum_{k=0}^{|\text{preds}(i)|} \log_2(|\mathcal{P}| + k)$, the cost of inserting the predecessors into the priority queue. We ignore the cost of maintaining the model required as this is small, constant and incurred only once for each observation.

$\mathbf{Q(\lambda)}$

The costs for $Q(\lambda)$ are given in three parts. First the cost of calculating ε' and ε each tick is given by:

$$4 + |\mathcal{A}_i| + |\mathcal{A}_j|$$

where the transition $i \rightsquigarrow j$ has just been observed. The second set of computations occur in updating the eligibility traces. We denote the set of state-action pairs which are eligible to be updated by Λ:

$$\Lambda = \{(i, a) : \bar{e}(i, a) > 0\}$$

Thus the cost of updating the traces each tick is:

$$|\Lambda| - 1$$

[7] This assumes that we first find $\max_{a \in \mathcal{A}_{i'}} \{\hat{p}_{i'i}(a)\}$ and *then* evaluate $\hat{p}_{i'i}(a)\Delta > \varepsilon$. This method requires fewer computations than the straightforward method described in Figure 2.9.

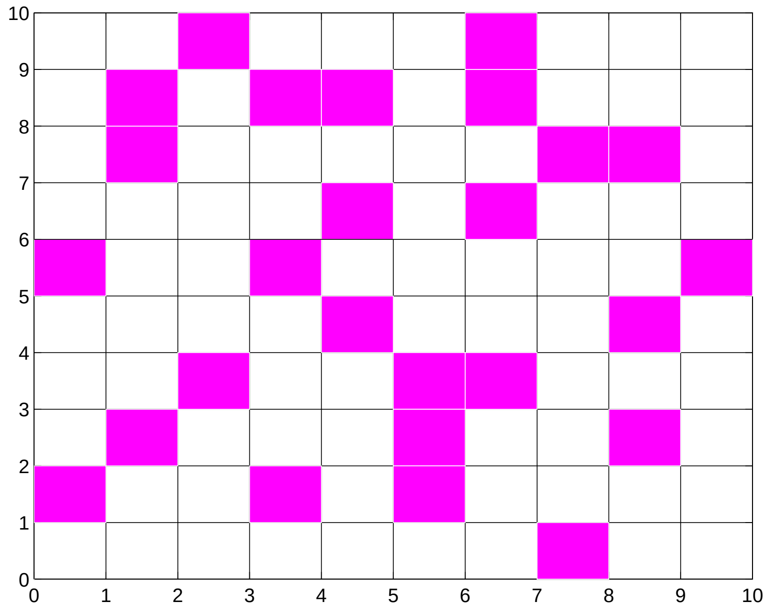


Figure 5.7: Task 10. 10x10 maze task. The initial state is in the lower left corner. The goal state in the upper right.

for replacing traces as used here. Finally the cost of updating the corresponding $\hat{Q}$-values each tick is $2|\Lambda|$, making the total cost of the $Q(\lambda)$ algorithm each tick:

$$5 + |\mathcal{A}_i| + |\mathcal{A}_j|) + 3|\Lambda|$$

This completes the analysis of the number of computations carried out each step by each algorithm.

5.2.2 Experiment 5.3

In order to compare the inferential power of Q(λ) and prioritised sweeping they were both tested on a maze task, defined on a square grid of side 10 (see Figure 5.7). The start and goal states were in diagonal corners. On entry into the goal state the agent received a reward of 100. All other states generate rewards of 0. For all agents on this task $\gamma = .99$.

To restrict the variation between each method to the inference mechanism, both were employed with a local model-based counter driven exploration method. Each step the action which maximises the agent's estimate of the exploration measure given

	Psweep	Q(λ)	optimum	initial
final RMSE(Q)	4.3035	66.4213	0	72.2461
final $V^{\pi}(x_0)$	74.5571	68.1256	78.0213	1.3399
mean peak computation	1627	668	-	-
mean computation/tick	899	100	-	-

Table 5.2: Task 10. Performance of prioritised sweeping and Q(λ).

by Equation 3.8 is chosen. This estimate is maintained using Equation 3.9. The parameters chosen were as follows. Q(λ) employed an initial value for α which was then declined exponentially. The initial value $\alpha_0 = 1$. It was declined every step in which at least one backup was made, by $\delta_\alpha = .998$, according to the rule:

$$\alpha_{t+1} = \begin{cases} \alpha_t\delta_\alpha & \text{if } \alpha_t\delta_\alpha > \alpha_{\min} \\ \alpha_{\min} & \text{otherwise} \end{cases}$$

where $\alpha_{\min} = .2$. In addition $\lambda_0 = 1$, λ being declined in the same manner as α, with $\delta_\lambda = .998$ and $\lambda_{\min} = 0$. These values were obtained as good ones by informal experimentation[8]. For prioritised sweeping, $\varepsilon = 10^{-3}$, and $k = 2$. Each algorithm was run on the maze task for 100 runs of 5000 ticks each. During a run, when the goal state was reached the agent was automatically sent back to the start state to commence a new trial.

Results

For each agent a number of performance criteria were recorded. These are summarised in Table 5.2. Every tick the root mean squared error in the estimated Q-values was calculated. To calculate this error the actual Q-values were calculated using value iteration and the process transition function. The figures presented in Table 5.2 give the mean RMSE on the final tick of each run, over all runs. The initial error is also recorded for reference. A graph (Figure 5.9) plots the mean RMSE over time.

The value of the greedy policy in the initial state was also recorded every 10 ticks. This was calculated in the same manner as the actual Q-values: using the process transition

[8] It is notable that declining the learning rate in this way violates the conditions for the convergence of Q-learning outlined in Chapter 2. There may be considerable differences between the ways in which parameters are best declined in order to achieve performance guarantees and rapid convergence.

function and the policy believed by the agent to be optimal. The mean value over all runs of each agent's policy on the final tick was calculated. This value is recorded for each algorithm in Table 5.2 as $V^{\pi}(x_0)$. The value of the optimal policy in the initial state, and the value of the random policy are also recorded. In addition the mean over all runs of the value of the greedy policy in the initial state is plotted against time for both algorithms (Figure 5.8).

Using the equations detailed in Section 5.2.1 the number of basic computations (tocks) per tick were recorded every tick. The mean over all runs of the peak computation was calculated and recorded in Table 5.2. In addition the mean over all runs of the average computation per tick was calculated and also recorded in Table 5.2. The mean over all runs of the actual computation each tick was also calculated, and is plotted against time in Figure 5.10.

The differences observed between algorithms for all values in Table 5.2 are significant at the 1% level or above. In addition the differences observed in the graphs are significant at the 1% level or above. In Figures 5.8 and 5.9 the differences become significant after about 100 ticks. In Figure 5.10 all the differences are significant except for where the plots cross. The significance test used for all these calculations was a two-tailed t-test.

5.2.3 Discussion

In terms of the value of the greedy policy each step and the RMSE for the estimated Q-values, prioritised sweeping consistently outperforms Q(λ) with corrected traces. In terms of computational cost each step, however, prioritised sweeping is far more expensive than Q(λ). This is the case both in terms of average computation per tick (a ratio of 9:1); and peak computation per tick (a ratio of more than 2:1). In addition, although Q(λ) is outperformed on the first two criteria it still performs well in terms of the value of its greedy policy, reaching 87% of optimum performance after 5000 ticks compared with the figure for prioritised sweeping of 95.6%. Thus it provides good performance at a much lower cost than prioritised sweeping.

It must remembered, however, that the two agents were given similar experiences by means of a *model-based* exploration method. Using this method each agent completed

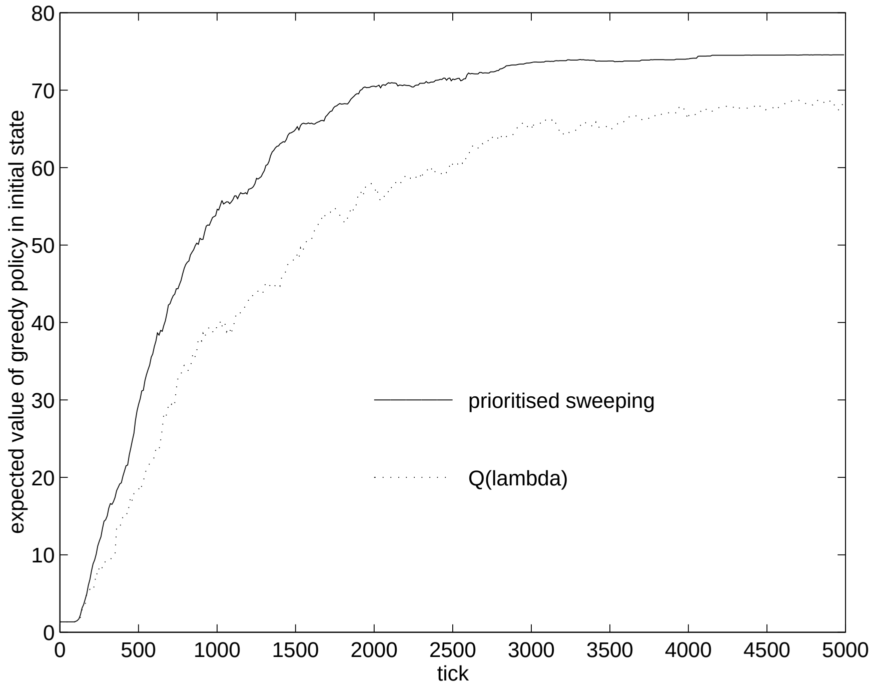


Figure 5.8: Task 10. Value of the greedy policy in state x_0.

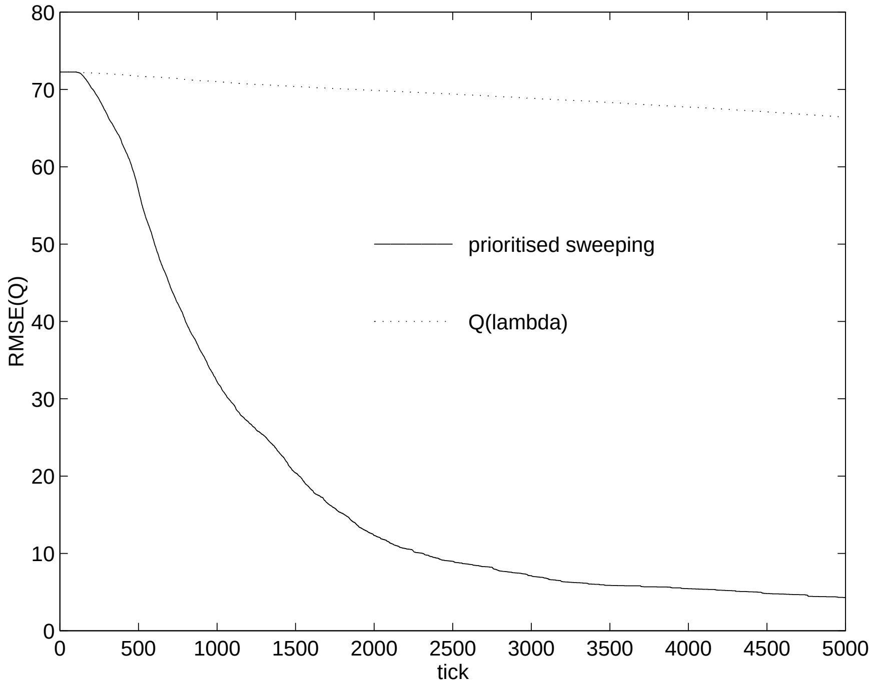


Figure 5.9: Task 10. Root mean squared error in the estimated Q-values.

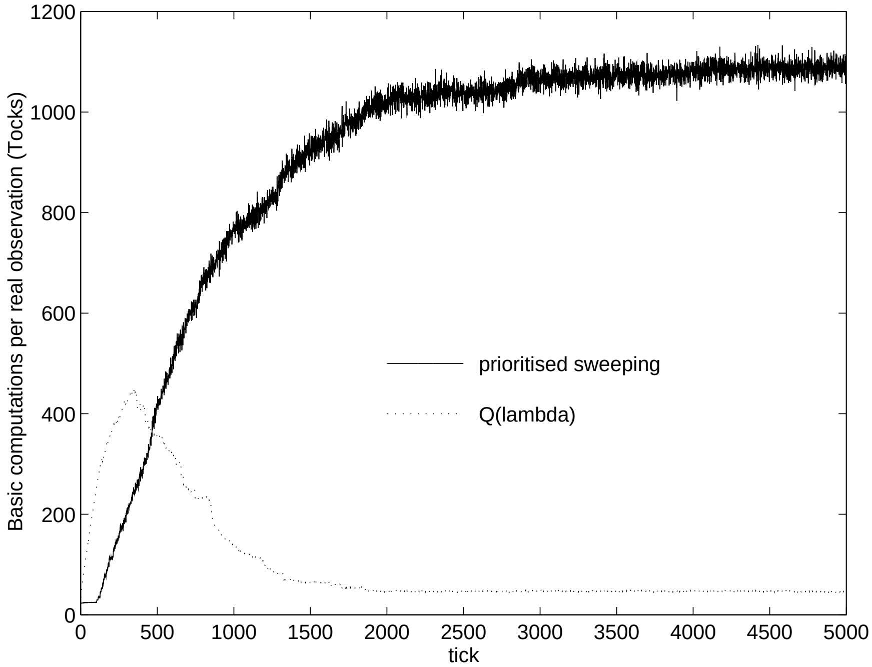


Figure 5.10: Task 10. Average number of basic computations performed each tick

an average of 31 trials in 5000 ticks. As will be shown in Chapter 6 the equivalent model-free rule does not explore as effectively[9]. In consequence the results here are only pertinent to the algorithms presented as inference methods, not as complete techniques for learning an embedded controller. As has been previously noted [53] the speed of convergence of an agent rests heavily on the exploration policy employed.

The other interesting point is that $Q(\lambda)$ performs so well on the greedy policy criterion, while leaving such a large errors in the estimated Q-values. This feature of Q-learning is one of the reasons it performs well for low per observation cost. The optimal policy does not require that the agent have accurate estimates of the Q-values, merely that the relative magnitudes of the estimated Q-values in each state be the same as for the actual Q-values. This is not just a feature of $Q(\lambda)$ learning, but of temporal difference methods in general.

[9] The local model-free rule presented in Chapter 6 was also used to control exploration on Task 10. Over 100 runs the agent completed an average of just 9 trials in 5000 ticks.

5.3 Conclusions

In summary this chapter has investigated some of the properties of two of the leading algorithms from the classes of model-based and model-free methods. First it was shown that if a Q(λ) agent deviates persistently from the greedy policy then the estimated Q-values will not converge to the actual Q-values when uncorrected traces are employed. In the the second half of the chapter, however, it was shown that the estimates of the Q-values need not necessarily be accurate in order to specify a good exploitation policy. The question of the extent to which an agent can deviate from the greedy policy and still learn a good policy when using uncorrected traces remains. Finally in a comparison of the inferential power of prioritised sweeping and Q(λ), it was shown that prioritised sweeping outperforms Q(λ) with corrected traces on a discrete maze task, but at greater computational cost.

Chapter 6

Exploration: the multi-state case

6.1 Introduction

Having investigated some new ways of thinking about the exploration problem in single state tasks; and having understood more about the nature of the inference methods to be used, this chapter turns to the multi-state case. Chapter 3 discussed how the task of guiding exploration can be framed as the problem of inferring an optimal policy given an exploration measure and a sequence of experiences. It also proposed the categorisation of exploration methods according to four criteria: into distal vs. local methods; into model-based vs. model-free methods; by the measure of exploratory worth used; and by the decision rule employed. In this chapter we examine issues arising out of the first two criteria. Specifically this chapter tests two hypotheses:

1. *Model-based estimates of exploration measures outperform their model-free counterparts.*
2. *Distal methods outperform their local counterparts.*

Although the results presented in this chapter are in principle extendible to many exploration measures the ideas presented are tested using one such measure, a counter based measure due to Thrun [86]. First, model-based and model-free exploration methods will be discussed(Section 6.2). Simple model-based and model-free rules for estimating a local counter based exploration measure are described. Following this, distal forms of both update rules are given. Finally the performance of all four methods are compared (Section 6.3). For the purpose of inferring the exploration policy the model-based and model-free algorithms investigated in Chapter 5 are used. In each instance grid tasks represented as Markov chains are employed for comparing the different methods[1].

6.2 Model-based vs. model-free estimates: local methods

6.2.1 Counter based measures revisited

Counter based exploration rules use information about the number of visits to each state or state-action pair in order to drive the agent to the less visited parts of the state space. Counter based exploration rules were first discussed in depth by Thrun [86]. As previously mentioned, counter based measures can be powerful exploration tools when rewards are naturally sparse. The simplest possible counter based exploration rule is to choose an action such that $\chi(x_t, a)$ is minimised over the set $\mathcal{A}(x_t)$, where:

$$\chi(x_t, a) = E[C(X_{t+1}|x_t, a)] \tag{6.1}$$

This will drive the agent to the least visited neighbouring state. A difficulty arises with this rule, in that as we explore the environment $E[C(X_{t+1}|x_t, a)]$ will rise through time. If we wish to combine $\chi(x_t, a)$ with some measure of utility then it is desirable to have its value converge through time. This is achieved in a local counter based exploration rule due to Thrun [86]. The exploration value of a state-action pair is the ratio of visits to the current state and the expected visits to the next state.

$$\chi(x_t, a) = \frac{C(x_t)}{\hat{E}[C(X_{t+1})|x_t, a]} \tag{6.2}$$

An action is chosen that maximises $\chi(x_t, a)$ over the set of actions. The use of a ratio damps the absolute range of values $\chi(x_t, a)$ is likely to take in the long run[2]. This makes

[1] These are commonly described in the literature as navigation tasks. Use of this term is avoided here because of its connotations outside the field of reinforcement learning.

[2] The ratio will be ∞ if the action is untried. In practice the maximum value of the measure is taken to be some arbitrarily large finite number. This is important if we want to define an exploration value function based on such a measure.

Algorithm 14 LOCAL MODEL-BASED COUNTER DRIVEN EXPLORATION

Let $\frac{z}{0} := \infty$, for all $z > 0$.

$t := 0$
$C_t(x) := 0, \forall x \in \mathcal{S}$
$\hat{p}_{xy}(a) := 0, \forall x, y \in \mathcal{S}, \forall a \in \mathcal{A}(x)$
observe x_t
repeat
    $C(x_t) := C(x_t) + 1$
    select $a_t := \max_{a \in \mathcal{A}(x_t)} \left\{ \frac{C(x_t)}{\sum_{y \in \mathcal{S}} \hat{p}_{x_t y}(a) C(y)} \right\}$
    observe the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$
    update $\hat{p}_{x_t x_{t+1}}(a_t)$
    $t := t + 1$

Figure 6.1: A model-based counter driven exploration algorithm.

it useful when we want to employ it as an exploration bonus in a combined exploration-exploitation rule. The principal issue remaining is how to estimate $E[C(X_{t+1})|x_t, a]$. If the agent possesses a model then it may be estimated using:

$$\sum_{y \in \mathcal{S}} \hat{p}_{x_t y}(a)\, C(y) \tag{6.3}$$

Alternatively, if the agent does not possess a model then $E[C(X_{t+1})|x_t, a]$ may be approximated using a model-free method[3]. After observing a transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$ the estimate $\hat{E}[C(X_{t+1})|x_t, a_t]$ is updated according to:

$$\hat{E}_{t+1}[C(X_{t+1})|x_t, a_t] = (1 - \alpha)\hat{E}_t[C(X_{t+1})|x_t, a_t] + \alpha C_t(x_{t+1}) \tag{6.4}$$

where $0 < \alpha < 1$. Using these estimators we can construct simple algorithms for guiding exploration. The model-based algorithm (Figure 6.1) is a version of Thrun's

[3] It should be noted that although only one model-free method is presented here, four different counter-based measures with similar update rules were implemented to eliminate the possibility that a poorly performing model-free method was being compared to the model-based method. The rule presented is the one that proved the most successful.

Algorithm 15 LOCAL MODEL-FREE COUNTER DRIVEN EXPLORATION

$\epsilon < 1$ is some small finite number. Y is the random variable denoting the state succeeding a state x. $0 < \alpha < 1$

$t := 0$
$C_t(x) := 0, \forall x \in \mathcal{S}$
$\hat{E}_t[C(Y)|x, a] := \epsilon, \forall x \in \mathcal{S}, \forall a \in \mathcal{A}(x)$
observe x_t
repeat
    $C(x_t) := C(x_t) + 1$
    select $a_t := \max_{a \in \mathcal{A}(x_t)} \left\{ \frac{C(x_t)}{\hat{E}_t[C(X_{t+1})|\, x_t, a]} \right\}$
    observe the transition $x_t \overset{a_t}{\leadsto} x_{t+1}$
    update $\hat{E}[C(X_{t+1})|x_t, a_t]$ using Equation 6.4
    $t := t + 1$

Figure 6.2: A model-free counter driven exploration algorithm.

[86] counter based exploration method. At each step the agent selects the action with the highest value for the exploration measure; the transition $x_t \overset{a_t}{\leadsto} x_{t+1}$ is observed; and the estimates of $C(x_{t+1})$ and $\hat{p}_{x_t x_{t+1}}(a_t)$ are updated. The model-free algorithm (Figure 6.2) is identical except that Equation 6.4 is used to maintain an estimate of $E[C(Y)|x, a]$ instead. I now proceed to extend both these local estimates to distal ones, before presenting the results of an empirical comparison.

6.3 Distal vs. Local Exploration

Previously in Chapter 3 the exploration problem was framed in terms of a reinforcement learning problem, albeit a non-stationary one. On this view distal methods for controlling exploration ought, in principle at least, be able to outperform their local counterparts. This is the hypothesis that is tested by the experiments presented in this section. In these experiments the exploration value function is defined by:

Algorithm 16 (DISTAL MODEL-BASED COUNTER DRIVEN EXPLORATION)

$\mathcal{P}$ is the priority queue. $\Psi(i)$ is the priority of the state i. $0 < \varepsilon$. k is a positive integer.

$t := 0$, $\mathcal{P} := \emptyset$
$\hat{\xi}^*_t :=$ arbitrary bounded function
observe x_t
$C_t(x_t) := 1, \quad C_t(x) := 0, \forall x \neq x_t$
repeat
    choose $a_t := \arg\max_a \left\{ \frac{C_t(x_t)}{\hat{\xi}_t(x_t,a)} \right\}$
    observe the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$
    update $C(x_{t+1})$ and $\hat{p}_{x_t x_{t+1}}(a_t)$
    add x_t to $\mathcal{P}$, $\Psi(x_t) := \max_{s\in\mathcal{P}}\{\Psi(s)\} + \varepsilon$
    repeat k times or until $\mathcal{P} = \emptyset$
        for $i := \arg\max_{s\in\mathcal{P}}\{\Psi(s)\}$
        $\hat{\xi}^*_{t+1}(i) := \min_a\{\sum_{j\in\mathcal{S}} \hat{p}_{ij}(a)[C_t(j) + \gamma\hat{\xi}^*_t(j)]\}$
        for each (i', a) such that $\hat{p}_{i'i}(a) > 0$
            if $\hat{p}_{i'i}(a)\Psi(i) > \varepsilon$
            add i' to $\mathcal{P}$, $\Psi(i') := \max\{\Psi(i'), \hat{p}_{i'i}(a)|\hat{\xi}^*_{t+1}(i) - \hat{\xi}^*_t(i)|\}$
        remove i from $\mathcal{P}$
    $t := t + 1$

Figure 6.3: A distal model-based counter-driven exploration algorithm.

$$\xi(x, a) := \sum_{y\in\mathcal{S}} \hat{p}_{xy}(a)[C(y) + \gamma\xi^*(y)] \tag{6.5}$$

and

$$\xi^*(s) = \min_{a\in\mathcal{A}(s)} \{\xi(s, a)\} \tag{6.6}$$

It is important to note that the exploration value function must be minimised at each stage. This means that the exploration measure used here could not be combined with an estimate of utility in the manner of an exploration bonus (as was discussed in Section 3.4.1), unless utility were also being minimised (i.e. the optimal policy would

Algorithm 17 DISTAL MODEL-FREE COUNTER-DRIVEN EXPLORATION

$\hat{\xi}^*(x) = \min_a \hat{\xi}(x, a)$. $0 \leq \lambda \leq 1$. ε'_t and ε_t are error signals. Boltzmann(.) specifies the Boltzmann distribution.

$t := 0$
$\hat{\xi}^*(x, a) := 0$ and $\bar{e}_t(x, a) := 0, \forall x, a$
observe x_t
$C_t(x_t) := 1$, $C_t(x) := 0, \forall x \neq x_t$
repeat
  choose a_t from Boltzmann$\left(\frac{C_t(x_t)}{\hat{\xi}_t(x_t,a)}\right)$
  observe the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$
  update $C_t(x_{t+1})$
  calculate ε'_t and ε_t using Eq. 6.7 and 6.8
  update $\bar{e}(x, a)$ for all $x \in \mathcal{S}, a \in \mathcal{A}$ according to Eq. 2.19 or 2.20
  update $\hat{\xi}_{t+1}(x, a)$ for all $x \in \mathcal{S}, a \in \mathcal{A}$ using
    $\hat{\xi}_{t+1}(x_t, a_t) := \hat{\xi}_t(x_t, a_t) + \alpha\varepsilon'_t\bar{e}_t(x_t, a_t)$
    $\hat{\xi}_{t+1}(x, a) := \hat{\xi}_t(x, a) + \alpha\varepsilon_t\bar{e}_t(x, a)$ for all $\hat{\xi}(x, a)$ except $\hat{\xi}(x_t, a_t)$
  t:=t+1

Figure 6.4: A distal model-free counter-driven exploration algorithm.

be the policy which minimised expected return each step).

Maintaining a model-free estimate of this measure requires the definition of two temporal differences. Suppose the agent observes the transition $x_t \overset{a_t}{\rightsquigarrow} x_{t+1}$. The errors generated will be:

$$\varepsilon'_t := C_t(x_{t+1}) + \gamma\hat{\xi}^*_t(x_{t+1}) - \hat{\xi}_t(x_t, a_t) \tag{6.7}$$

$$\varepsilon_t := C_t(x_{t+1}) + \gamma\hat{\xi}^*_t(x_{t+1}) - \hat{\xi}^*_t(x_t) \tag{6.8}$$

A number of other minor changes are necessary to both algorithms. For clarity of exposition, these algorithms are specified as implemented in Figures 6.3 and 6.4. The

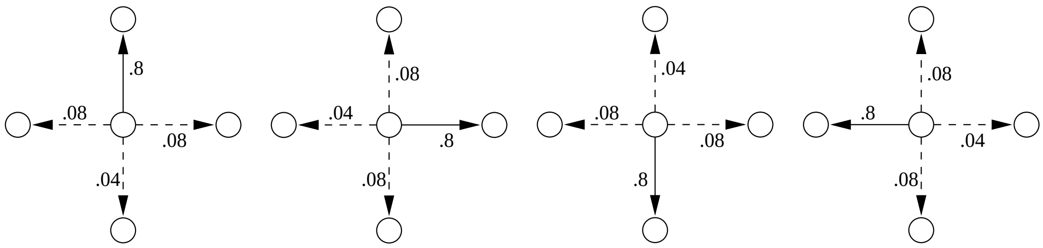


Figure 6.5: Tasks 11-13. Transition probabilities for each action to neighbouring states.

most notable change is that a decision rule based on the Boltzmann distribution was employed for the model-free algorithm. This meant that corrected traces as specified in Chapter 5 had to be used. The reader may compare the remaining differences by reference to the more general versions.

6.3.1 Empirical Comparison

In order to evaluate the efficacy of these methods, they were tested on a series of grid tasks. Each state in each grid has four permissible actions. The transitions between states are stochastic. The probabilities of travelling to each of the four adjoining states from any particular state given each action are as depicted in Figure 6.5. The solid arrow represents the most likely direction of travel given a particular action. The dashed arrows represent the other probabilities. Three square grids were used, of side 10, 30 and 50 respectively. At the edges and corners of the grids, transitions not leading to any other state instead lead back to the same state. Each process is thus composed of a single ergodic set.

Each agent was run on each task for a number of trials. Each trial the initial state of the process was one of the corner states. Each step the agent selected an action and observed the resulting transition. A trial completed when all states had been visited at least once. At the end of each trial the number of steps taken was recorded. Fifty trials were carried out for each algorithm on each task. For Algorithm 15, the following parameter values were used[4]: $\alpha = .3$ and $\epsilon = e^{-9}$. The parameters for Algorithm 16 were $\varepsilon = 10^{-3}$, and $k = 20$. For Algorithm 17 a range of parameter values were tried,

[4] Although results for a wide-range of parameter values are not presented here, informal experiments showed the values used to be good ones.

Parameter	**Values**
θ	$.01^{\dagger}$ .05 .1 .3 .5
α	.1 .3 .5
λ	.5 .6 .7 .8 .9 1

Table 6.1: Parameter values for Algorithm 17 on Tasks 11–13. † $\theta = .01$ was only used for Task 11.

Task	**Parameter**		
	θ	α	λ
11	.01	.1	.7
12	.05	.5	.5
13	.05	.3	.8

Table 6.2: Best observed parameter values for Algorithm 17 on Tasks 11–13

in order to optimise its performance (Table 6.1). For Tasks 12 and 13, 72 parameter sets were tried. For Task 11, runs were also conducted with $\theta = .01$, thus creating another 18 parameter sets.

In addition to Algorithms 14–17, a certainty equivalent estimate of the value-function, calculated using value-iteration, was also used to guide exploration. This last estimate of the value function is the best that can be achieved bar resorting to fundamentally intractable methods. The performance of the certainty equivalent method answers two questions. First, can a distal exploration rule outperform a local exploration rule in principle? Second, how far short of this benchmark performance will the cheaper methods based on prioritised sweeping and Q(λ) fall? All algorithms and parameter sets were run for 50 trials on all tasks. The best performing parameter values for Algorithm 17 for each task are shown in Table 6.2. For all algorithms on all tasks $\gamma = .99$.

Results

The mean number of steps (ticks) until completion — rounded to the nearest integer — is given for each algorithm on each task in Table 6.3. The best performing agent for each task is defined as the agent which minimises the mean number of ticks required to visit every state in the process at least once. As the number of states rises across tasks so does the time taken to complete those tasks. Figures 6.7– 6.9 show plots of

Task	# states	mean # steps				
		lmb	lmf	ce	dmb	dmf
11	100	632	701	575	591	718
12	900	7522	9866	6245	5983	9760
13	2500	22972	30736	18370	19013	29896

Table 6.3: Tasks 11-13. Counter driven exploration. Mean number of steps to visit all states.

the actual data points obtained. These figures also mark the mean and its standard error for each agent/environment combination. The algorithm to which each set of points corresponds is marked on the left hand margin of each graph. It can be seen from the graphs that there was considerable variability in the time taken to complete each task. This variability increases in tandem with the mean completion time for each algorithm/task combination.

To determine which differences between performances are statistically significant, a two-tailed t-test was used to determine the probability of obtaining each observed difference given the null hypothesis that the observations were drawn from populations with the same mean value. Graphs showing the significant dominance partial ordering among algorithms illustrate these differences in Figure 6.6. In each diagram the dominant algorithms are placed at the top of the graph. An edge between two nodes denotes a significant difference between those two nodes. A solid line denotes a significant difference at the 1% level or above; a dashed line denotes a significant difference at the 5% level; and a dotted line denotes a significant difference at the 10% level.

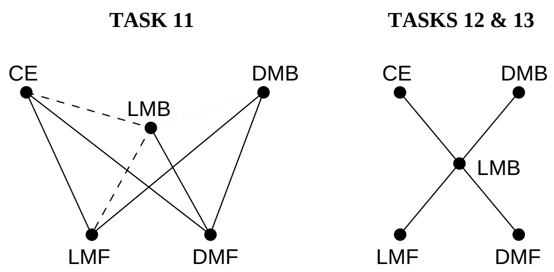


Figure 6.6: Significant dominance partial order among algorithms for Tasks 11-13.

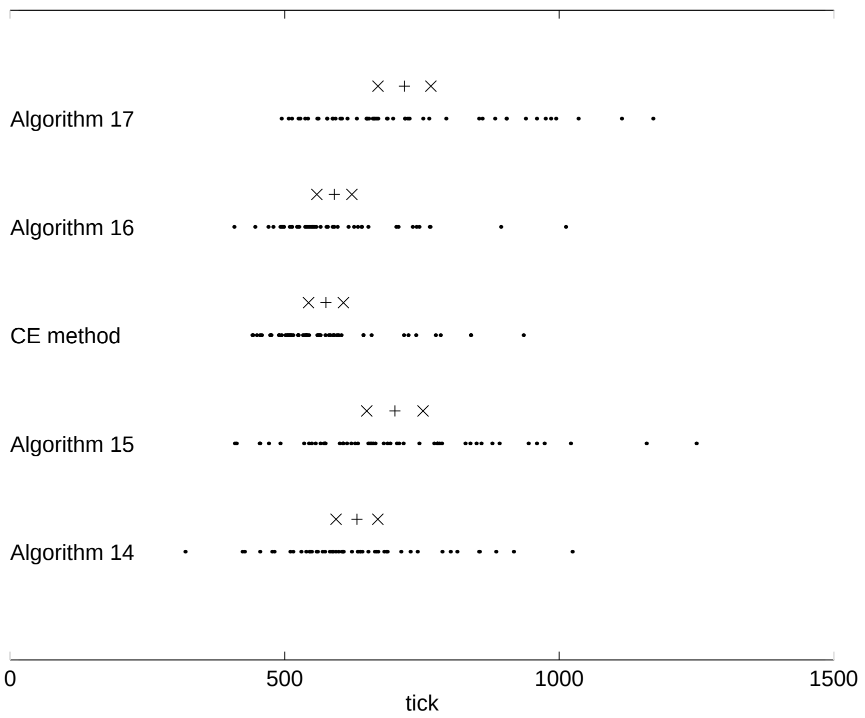


Figure 6.7: Task 11. Time steps until completion. Averaged over 50 trials. The individual data points are marked as dots; the + marks the mean trial length; each × marks one of the bounds on the 95% confidence interval for the mean trial length.

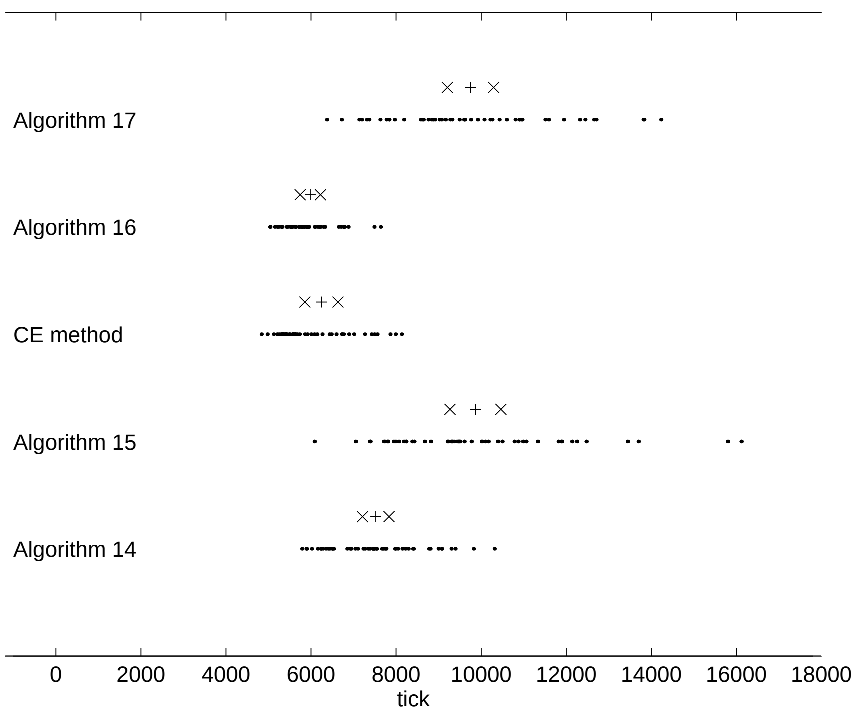


Figure 6.8: Task 12. Time steps until completion. Averaged over 50 trials. The individual data points are marked as dots; the + marks the mean trial length; each × marks one of the bounds on the 95% confidence interval for the mean trial length.

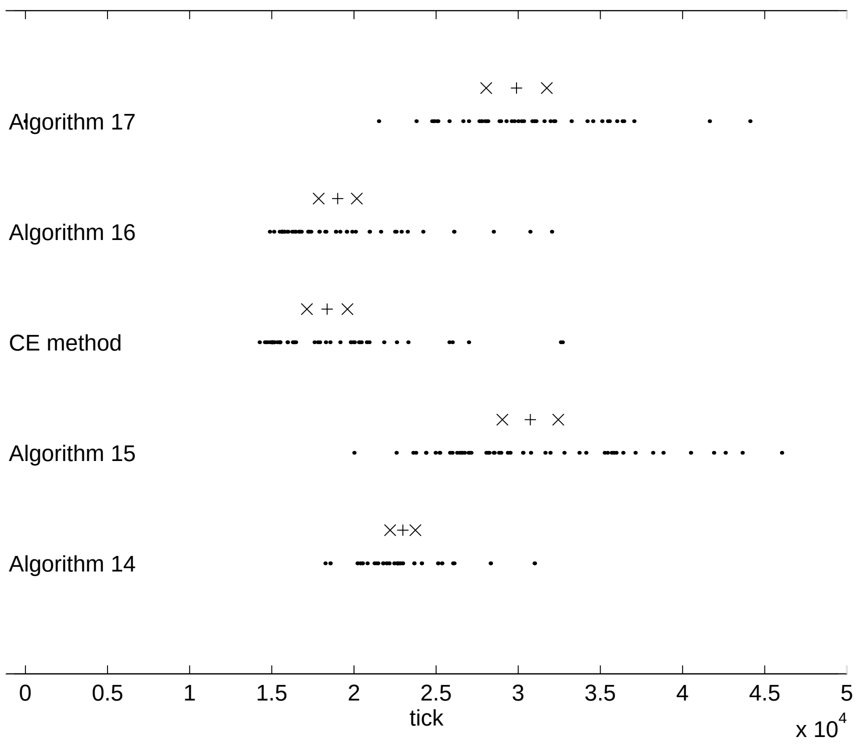


Figure 6.9: Task 13. Time steps until completion. Averaged over 50 trials. The individual data points are marked as dots; the + marks the mean trial length; each × marks one of the bounds on the 95% confidence interval for the mean trial length.

6.4 Discussion

When considering the results the first important point is that the CE method was used as a baseline for comparison. It is not a technique that can practically be considered for controlling exploration in an embedded agent. This is because of its high computational costs per real observation. It represents an ideal solution given the certainty equivalence assumption. Given this fact the best performing method is the distal model-based technique employing prioritised sweeping. It outperforms all the other algorithms on all three tasks. It also performs very close to the CE baseline, actually completing in fewer mean runs than the CE method on Task 14 (though this difference is not significant). It is significantly better than the local model-based method on each task. Thus at least for the model-based method and the exploration measure tested, the second hypothesis stated at the beginning of the chapter holds.

The same is not true of the distal model-free method chosen. Despite extensive search of the parameter space, only small improvements in performance were recorded over the local model-free method. None of these differences were significant with a sample size of 50 trials. In fact, most of the parameter sets for the distal methods performed considerably *worse* than the local model-free method. Many of *these* differences *were* significant. Trials were also carried out with a deterministic decision rule (rather than the Boltzmann rule reported here). This allowed the use of uncorrected traces (because the agent always followed the greedy exploration policy). Even with the additional inferential power afforded by uncorrected traces no performance improvement over local model-free methods could be found. In fact the deterministic model-free rule fared much worse than the Boltzmann rule used here. In the light of these results it must be concluded that distal methods are not necessarily an improvement over local methods for model-free algorithms. In this case the second hypothesis must be rejected.

The reason that the model-free distal method performed so poorly was that the inference technique was not powerful enough to maintain an accurate enough estimate of the non-stationary exploration value-function. When taking the action minimising this function locally the agent became trapped in numerous local minima. In an attempt to overcome this the Boltzmann decision rule was employed. By using a stochastic decision rule, some of the bumps in the exploration value function can be smoothed out. Unfortunately the improvement is not sufficient.

Finally it should be noted that the local model-based method was an improvement on the local model-free method for all tasks. Thus the first hypothesis stated at the beginning of the chapter has not been rejected. For the exploration measure, tasks and inference techniques used, model-based methods can be said to outperform their model-free counterparts.

6.5 Extensions

The results presented in this chapter should be intuitive in the light of the work of previous chapters on inference. If distal methods are to be useful general methods, however, they must be applied to a wider range of heuristics. Such an investigation would form the most sensible extension of the work presented in this chapter. In particular it would be useful to extend methods based on estimating the variance in rewards. Kaelbling [37] reported results showing that methods relying on estimates of variance in reward do not extend well to multi-state tasks. In the light of this work it may be posited that this is because of the non-stationary nature of the exploration value function determined by the variance in return. If a reliable estimator of such variance could be found then not only Kaelbling's IE method, but also all the variance based methods for controlling exploration reported in Chapter 4 could be extended to multi-state tasks. Such work would form a natural and important continuation of the work presented in this thesis.

6.6 Conclusions

In this chapter two hypotheses concerning exploration have been tested. The first was that model-based methods are an improvement on their model-free counterparts. This hypothesis can not be rejected on the basis of the results presented here, and is therefore strengthened. The second hypothesis was that distal methods are an improvement on their local counterparts. While this can be accepted in the model-based case on the basis of the results presented here; it must rejected in the model-free case.

Chapter 7

Conclusion

This thesis has been concerned with extending our understanding of the problem of controlling exploration in embedded agents which learn from reinforcement. Several contributions have been made.

- The problem of controlling exploration for future exploitation has been identified. Techniques for its solution have been developed and tested. Specifically, two novel measures of an agent's knowledge about its task have been developed. These have in turn been used to derive novel algorithms for the control of exploration. These algorithms have been extensively tested on tasks with a single state, and shown to outperform other possible candidate algorithms with respect to exploration for exploitation. A proof of convergence has been presented in one case. It has also been established that the solutions to exploration for future exploitation and the exploration-exploitation trade-off are necessarily different.

- The possibility of model-free verification methods for controllers learned from reinforcement has been established as a by-product of this work. These methods have been used to analyse the controllers for single state tasks learned from reinforcement.

- A consistent framework for heuristic approaches to exploration control has been presented. In addition simple distal exploration methods have been implemented and tested. It has been shown that distal methods have the potential to outperform local methods in the model-based case, but that this advantage does not necessarily extend to cases when existing model-free inference methods are employed.

7.1 Exploration for future exploitation

This view of the purpose of exploration is different to that of previous work on learning from reinforcement. It is motivated by a desire to utilise reinforcement learning algorithms as part of the process of designing embedded controllers. It has been an often stated aim of researchers in robotics to be able to raise the level of abstraction of robot programming, and at the same time to formalise the design process in order that the controllers constructed might have verifiable performance bounds. In the opening chapter an argument was presented that learning algorithms, and particularly algorithms which learn from reinforcement, have the potential to fulfill both these aims. This work can be seen as a step towards realising these aims.

It has also been argued that when trying to identify an optimal controller during a relatively cost free learning period, exploration should be controlled in such a way as to maximise the agent's task knowledge rather than its performance during the learning period. While this criterion has not been previously investigated within the field of learning from reinforcement there has been considerable practical interest in applying reinforcement learning to just these sorts of tasks [19]. For complex problems it may in fact be easier to use reinforcement learning techniques in controller design than in truly autonomous learning. This is because for practical purposes it is tremendously difficult to design good reinforcement functions without at least some knowledge of the task to be solved. In recent work it has been argued that learning from reinforcement does not easily free us from the implement-test-debug cycle familiar to designers of robot controllers [31].

There are limitations to the methods presented in thesis for controlling exploration for future exploitation. Firstly the methods have only been developed for single state tasks. There are potential difficulties with their extension. Most significantly the methods presented all require a reliable and unbiased estimate of variance in reward or return. Previous work [37] making just this assumption has foundered on the problem of constructing such estimates in multi-state tasks. Clearly the most important and useful extension of the work presented here would be to develop methods for maintaining reliable estimates for such tasks.

7.2 Distal exploration

Although the view of exploration control as an inference problem is hardly new, this thesis has developed the idea further. It has also made some useful steps toward understanding the relative benefits of model-free and model-based methods for controlling exploration. The results presented may have some interesting implications for reinforcement learning in general. This is because the method used to solve either one of the exploration or inference problems determines the method used to solve the other. If a model-based method is used for learning control, then it is clearly sensible to use a model-based learning method for inferring the correct exploration policy. Conversely, if available computation is only sufficient for model-free methods to be used for learning control, then the agent must rely on those same methods for controlling exploration. This may cause problems for model-free learners. If efficient exploration is required then it appears from the initial results presented here that model-free methods are poor performers. Although results were only presented for a single exploration measure, it is plausible that exploration value functions are non-stationary in their very nature, and thus hard for model-free methods to track. Testing this hypothesis by the development of further distal exploration methods based on existing local exploration measures is an obvious and necessary extension of the work presented here. Finally the framework for exploration presented here also needs to be extended. Recent work [22, 70] has taken the field in radically new directions, and the work presented here needs to be extended and formalised to account for these developments.

Bibliography

[1] P.E. Agre. Computational research on interaction and agency. *Artificial Intelligence*, 72:1–52, January 1995. Special Double Issue on Computational Research on Interaction and Agency.

[2] P.E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, volume 1, pages 268–272. Morgan Kaufmann, 1987.

[3] A. G. Barto. Connectionist learning for control. In W. T. Miller R. S. Sutton & P. J. Werbos, editor, *Neural Networks For Control*, pages 5–58. MIT Press, 1990.

[4] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. Technical Report CMPSCI-TR-93-02, University of Massachusetts, March 1993. revised version of TR-91-57, Real-time learning and control using asynchronous dynamic programming.

[5] A.G. Barto and S.P. Singh. On the computational economics of reinforcement learning. In D.S. Touretsky, editor, *Connectionist Models: Proceedings of the 1990 Summer School*, pages 35–44. Morgan Kaufmann, 1991.

[6] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Learning and sequential decision making. COINS Technical Report 89-95, University of Massachussetts, September 1989. Later published in '*Learning and Computational Neuroscience*' edited by M.Gabriel & J.W. Moore.

[7] R.D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72:173–215, January 1995. Special Double Issue on Computational Research on Interaction and Agency.

[8] R. Bellman. A problem in the sequential design of experiments. *Sankhyā*, 16:221–229, 1956.

[9] D.A. Berry and B. Fristedt. *Bandir Problems: Sequential Allocation of Experiments.* Chapman and Hall, 1985.

[10] D. Bertsekas. *Dynamic Programming and Stochastic Control.* Academic Press, 1976.

[11] R.A. Brooks. A robust layered control system for a mobile robot. A.I. Memo 864, MIT, September 1985.

[12] R.A. Brooks. Achieving artificial intelligence through building robots. A.I. Memo 899, MIT, May 1986.

[13] Rodney A. Brooks and Maja J. Mataric. *Real robots, real learning problems*, chapter 8, pages 193–214. Kluwer Academic Publishers, 1993.

[14] D.E. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[15] Pawel Cichosz. Truncating temporal differences: on the efficient implementation of td(λ) learning. *Journal of Artificial Intelligence Research*, 2:287–318, January 1995.

[16] Harvey, I. Cliff, D. and Husbands, P. Incremental evolution of neural network architectures for adaptive behaviour. Research Paper CSRP 256, School of Cognitive and Computing Sciences, University of Sussex, December 1992.

[17] Marco Colombetti and Marco Dorigo. Training agents to perform sequential behaviour. Submitted to the Journal of Evolutionary Computation, September 1993.

[18] Jonathan H. Connell. A colony architecture for an artificial creature. Technical Report 1151, MIT AI Lab, August 1989.

[19] S. Mahadevan & J.H. Connell. Automatic programming of behaviour-based robots using reinforcement learning. Research Report RC 16359 (72625), IBM Research Division, July 1990.

[20] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[21] G. Dayan, P. & Hinton. Feudal reinforcement learning. In Hanson S.J. & Lippmann R.P. Moody J.E., editor, *Advances in Neural Information Processing Systems 5*, pages –. Morgan Kaufmann, 1993.

[22] Peter Dayan and Terrence Sejnowski. Exploration bonuses and dual control. Presented at the AAAI Symposium on Active Learning. MIT, November 1995.

[23] Goldberg D.E. Probability matching, the magnitude of reinforcement and classifier system bidding. *Machine Learning*, 5(4):407–425, October 1990.

[24] B.R. Donald. On information invariants in robotics. *Artificial Intelligence*, 72:217–304, January 1995. Special Double Issue on Computational Research on Interaction and Agency.

[25] M. Dorigo and U. Schnepf. Genetics-based machine learning and behaviour-based robotics: A new synthesis. *IEEE Transactions On Systems, Man, and Cybernetics*, 23(1):141–154, January 1993.

[26] Marco Dorigo and Marco Colombetti. Robot shaping: Developing situated agents through learning. Technical Report TR-92-040 (revised), International Computer Science Institute, April 1993.

[27] Gary L. Drescher. A mechanism for early piagetian learning. In *Proceedings of AAAI-87: The Sixth National Conference on Artificial Intelligence*, volume 1, pages 290–294. Morgan Kaufmann, 1987.

[28] J.C. Gittins. *Multi-armed Bandit Allocation Indices.* Interscience Series in Systems and Optimization. John Wiley & Sons, 1989.

[29] J.C. Gittins and D.M. Jones. A dynamic allocation index for the sequential design of experiments. In em et al. Gani, J, editor, *Progress in Statistics*, volume 1, pages 241–266. North-Holland, 1974. Proceedings of the European Meeting of Statisticians, Budapest, Hungary, 1972.

[30] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company Inc., 1989.

[31] John Hoar. Reinforcement learning applied to a real robot task. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, September 1996.

[32] J.H. Holland. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R.S. Mishalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning II*, pages 593–623. Kaufman, 1986.

[33] Jonas Karlsson JoshTenenberg and Steven Whitehead. Learning via task decomposition. In *From Animals to Animats 2: Proceedings of the 2nd International Conference on the Simulation of Adaptive Behaviour*, pages 337–343. MIT Press, 1992.

[34] David Chapman & Leslie Pack Kaelbling. Input generalisation in delayed reinforcement learning: an algorithm and performance comparison. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1991.

[35] L. Kaelbling. Goals as parallel program specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[36] L. Kaelbling. *Learning in Embedded Systems.* MIT Press, 1993.

[37] Leslie Pack Kaelbling. *Learning in Embedded Systems.* PhD thesis, Dept of Computer Science, Stanford, 1990.

[38] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: preliminary results. In *Machine Learning: Proceedings of the 10th International Conference*, pages 167–173, 1991.

[39] Littman M.L. Kaelbling L.P. and Moore A.W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, forthcoming.*, 1995.

[40] Chrisman L. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, 1992.

[41] Y. Lespérance and H.J. Levesque. Indexical knowledge and robot action — a logical account. *Artificial Intelligence*, 73, 1995. Double Issue on Computational Research on Interaction and Agency.

[42] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the ninth national conference on artificial intelligence*, pages 781–786. AAAI Press/MIT Press, 1991.

[43] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3/4):293–321, 1992.

[44] Long-Ji Lin. Hierachical learning of robot skills by reinforcement. In *Proceedings of the IEEE International Conference on Neural Networks 1993*, pages 181–186, 1993.

[45] Long-Ji Lin and Tom M. Mitchell. Reinforcement learning with hidden states. In *Proceedings of the Second International Conference on the Simulation of Adaptive Behaviour*. MIT Press, 1992.

[46] M.L. Littman, T.L. Dean, and L. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, 1995.

[47] M. Littmann. The witness algorithm: Solving partially observable markov decision processes. Technical report CS-94-40, Brown University, Department of Computer Science, December 1994.

[48] Pattie Maes. How to do the right thing. *Connection Science*, 1(3):291–323, 1989.

[49] R.A. Maes, P. & Brooks. Learning to coordinate behaviours. In *Proceedings of the 8th National Conference on AI*, 1990.

[50] M. Mataric. Reward functions for accelerated learning. In W.W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 181–9. Morgan Kaufmann, February 1994.

[51] Matthew A.F. McDonald and Philip Hingston. Approximate discounted dynamic-programming is unreliable. Technical Report 94/6, University of Western Australia, Dept of Computer Science, October 1994.

[52] Ryszard S. Michalski. Understanding the nature of learning: Issues and research directions. In R.S. Michalski, J.G. Carbonnell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Vol II*, pages 3–24. Morgan Kaufmann, Los Altos, 1986.

[53] Andrew W. Moore and Christopher G Atkeson. Prioritised sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[54] Andrew William Moore. *Efficient memory-based learning for robot control*. Technical report, University of Cambridge, Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, November 1990.

[55] A.W. Moore and C.G. Atkeson. *Memory-Based Reinforcement Learning: Converging with Less Data and Less Real Time*, chapter 4, pages 79–103. Kluwer Academic, 1993. Abridged version of their 1993 Machine Learning Article.

[56] K. Narendra and M.A.L. Thathachar. *Learning Automata: An Introduction.* Prentice-Hall, 1989.

[57] Ulrich Nehmzow. *Experiments in Competence Acquisition for Autonomous Mobile Robots.* Ph.d. thesis, Department of Artificial Intelligence, Edinburgh University, 1992.

[58] Nils J. Nilsson. Shakey the robot. Technical Note 323, Stanford Research International, April 1984.

[59] Mark Pendrith. On reinforcement learning of control actions in noisy and non-markovian domains. Tech report UNSW-CSE-TR-9410, University of New South Wales, School of Computer Science and Engineering, August 1994.

[60] Jing Peng and Ronald J. Williams. Incremental multi-step q-learning. In W.W.Cohen and H.Hirsh, editors, *Machine Learning: Proceedings of the 11th International Conference*, pages 226–232, 1994.

[61] D.A. Pomerleau. Rapidly adapting artificial neural networks for autonomous navigation. In J.Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems 3*, pages 429–435. Morgan Kaufmann, 1991.

[62] M.B. Ring. Finding promising exploration regions by weighting expected navigation costs. Presented at the AAAI Symposium on Active Learning. MIT, November 1995.

[63] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

[64] Kenneth J Rosenblatt and David W Payton. A fine grained alternative to the subsumption architecture for mobile robot control. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, pages –, 1989.

[65] S.J. Rosenschein. Formal theories of knowledge in ai and robotics. *New Generation Computing*, 3(4):345–357, 1985.

[66] S.J. Rosenschein and L. Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 73, February 1995. Special Double Issue on Computational Research on Interaction and Agency.

[67] Sutton R.S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Bruce W. Porter and Ray J. Mooney, editors, *Machine Learning: Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[68] D.E. Rumelhart, J.L. McClelland, and The PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Vols I and II.* MIT Press, 1986.

[69] J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Technische Universität München, April 1991.

[70] J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Note IDSIA-59-95, IDSIA, June 1995.

[71] Jurgen Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In Hanson S. Moody J. and Lippmann R., editors, *Advances in Neural Information Processing Systems 3*, pages 501–506. Morgan Kaufmann, 1991.

[72] A Schwartz. A reinforcement learning method for maximising undiscounted rewards. In *Machine Learning: Proceedings of the Tenth International Conference*. Morgan Kaufmann, 1993.

[73] Satinder P Singh. The efficient learning of multiple task sequences. In S. Hanson J. Moody and R. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 251–258. Morgan Kaufmann, 1992.

[74] Satinder P Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3/4):323–339, 1992.

[75] S.P. Singh. Reinforcement learning algorithms for average-payoff markovian decision processes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI Press/MIT Press, 1994.

[76] T. Smithers. Are autonomous agents information processing systems? In L. Steels and R. Brooks, editors, *The Artificial Life Route to "Artificial Intelligence": Building Situated Embodied Agents*. Lawrence Erlbaum Associates: New Haven, 1994.

[77] Luc Steels. Artificial intelligence and complex dynamics. A.I. Memo 88-2, Brussels VUB AI Lab, 1987. Presented at the IFIP workshop on tools, concepts and kbs. Mnt Fuji, Japan.

[78] R.S. Sutton. *Temporal Credit Assignmenment in Reinforcement Learning*. PhD thesis, University of Massachusetts, School of Computer and Information Sciences, 1984.

[79] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[80] R.S. Sutton and A.G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88:135–170, 1981.

[81] R.S. Sutton and S.P. Singh. On step-size and bias in temporal difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96, 1994.

[82] Satinder Singh & Richard Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 1996. Accepted for Publication.

[83] Wilson S.W. Explore/exploit strategies in autonomous learning. Presented at the 1996 AISB Workshop on Mobile Robotics, Brighton, UK, April 1996.

[84] S.B. Thrun. The role of exploration in learning control. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches,*. Van Nostrand Rheinhold, 1992.

[85] S.B. Thrun and K. Moller. Active exploration in dynamic environments. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 532–538. Morgan Kaufmann, 1992.

[86] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, School of Computer Science, January 1992.

[87] Vladimir Vapnik. *Estimation of dependences based on empirical data.* Springer-Verlag, 1982.

[88] Lee W. *Decision Theory and Human Behaviour.* John Wiley and Sons, 1971.

[89] C.J.C.H Watkins. *Learning from delayed rewards.* Thesis, University of Cambridge, King's College, Cambridge, England, May 1989.

[90] C.J.C.H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[91] S.D. Whitehead and Long-Ji Lin. Reinforcement learning in non-markov decision processes. Submitted to the Special Issue of the AI Journal on 'Computational Theories of Ineraction and Agency'.

[92] Steven D. Whitehead and Dana Ballard. Learning to perceive and act. Technical Report TR-331 (revised), University of Rochester, Department of Computer Science, June 1990.

[93] Steven D. Whitehead and Dana H. Ballard. Active perception and reinforcement learning. In Bruce W. Porter and Ray J. Mooney, editors, *Machine Learning: Proceedings of the Seventh International Conference on Machine Learning*, pages 179–188. Morgan Kaufmann, 1990.

[94] R.J. Williams and L.C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-11, Northeastern University, College of Computer Science, November 1993.

[95] Ping Zhang and Stèphane Canu. Entropy-based trade-off between exploration and exploitation. In *Learning in Robots and Animals, The Second Biennal AISB Worshop Series,*, pages 112–120, Brighton, UK, April 1996.