Game-Theoretic Procedures for Learning Structured Strategies from Demonstration

Benjamin Rosman

0896970

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

2009

Abstract

Acquiring new skills is an ability which would be very useful for a robot to possess. One conceivable method whereby a robot could learn new skills is if a teacher with expert knowledge of those skills could demonstrate these skills to the learning robot, in a variety of different settings. What is required would be that the robot does not merely learn to mimic the teacher exactly, as the teacher would be unable to demonstrate the skill in every possible state of the world. Instead the robot should be equipped with some robust method for adapting the learned skill to new situations.

The research presented in this document embarks down a path towards solving this problem. Drawing on previous work from the fields of apprenticeship learning and reinforcement learning, an algorithm is presented for teaching an agent a new skill. This algorithm consists of two phases: the first in which the robot learns primitive skills from an expert, and the second which is a mechanism for composing these skills to provide strategies for successfully using the skill in unseen situations. This two phase solution is inspired by curriculum learning, and allows the agent to construct structured strategies in a hierarchical framework.

The learning task is posed as a two-player game of the learning agent against disturbance processes generated by the environment or nature. By doing so, an equilibrium can be found using regret minimisation techniques to find an optimal mixture of the basis strategies into a composite strategy for a new scenario.

A detailed example is provided, where the learning agent is required to learn how to navigate a busy multi-lane road of various shapes and sizes. The results demonstrate that the agent is able to successfully learn the primitive strategies from the expert. Then, in a variety of experiments, it is shown that the second phase of the algorithm can reconstruct parts of the basis that are present in new examples, as well as adapt the learned behaviours to totally unseen terrains, in a robust and flexible manner. By doing this, the agent shows an ability to synthesise its previously acquired knowledge in novel ways to overcome unpredicted challenges.

Acknowledgements

I would like to thank my supervisor, Dr Subramanian Ramamoorthy for all the advice and ideas, for the many long hours spent discussing the finer aspects of this research, and for the constant email communication: be it answering my tirade of obscure questions or sending me interesting articles.

My most sincere thanks also extend to my family and friends for their constant support and encouragement, for the laughs and the smiles, for always putting up with me, and for the aesthetic advice and proofreading. Your never wavering enthusiasm and assistance made all the difference.

Declaration

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

(*Benjamin Rosman*
0896970)

Table of Contents

Chapter 1

Introduction

Robots are becoming increasingly ubiquitous in society as they are adopted for an ever growing number of tasks. However, the question remains as to how a robot may learn a wide variety of actions without the need for them to be hard-coded. Having only hard-coded skills on a robot is not only very limiting to its operational capabilities, but also requires considerable time investment on the part of the developers. Moreover, it may be the case that the actions required to perform some task are very similar to those needed for a second task. If a robot is capable of completing the first task, it should have some process whereby it can utilise and adapt this skill to a different scenario.

One possible solution to these challenges, and indeed an important ability generally, would be if a robot could learn by demonstration. That is the ability for the robot to observe an external agent such as a human or another robot perform some task, and then using those observations in some way learn to perform that same task. This is a form of social learning, and an important aspect of what one may consider intelligent behaviour. Such processes can be observed in humans as well as many other creatures.

An advantage of this approach is that no overhead need be incurred in the teaching process from the point of view of the teaching agent, which may be performing the taught actions as part of some larger routine. As a result, a robot capable of learning behaviours in such a way is considerably more autonomous than one which cannot. Furthermore, the learning agent would hope to learn the control policies from the teacher, rather than the exact positioning of each component of its system. This is also advantageous because the student robot could be expected to have different physical attributes to the teacher, which implies that the simple approach of directly replicating each small change in the teacher's configuration would be inappropriate. In this way, the robot could be considered to have gained a more abstract description of how to

perform a task. An example of this principle is, in the case of learning to use a door handle to open a door, one would want a robot to learn where forces should be applied rather than the exact positioning of each finger, further bearing in mind that the robotic hand may be configured differently to that of a human teacher.

This problem is more than one of function approximation, as ideally the agent should learn a robust policy rather than merely imitating the behaviour of the expert. To reproduce a given trajectory is a standard supervised learning task which would equip a learning agent with an exact replica of the expert's trajectory. Instead of this, one would want a robot's control policies to be able to handle a variety of contingencies – some of which may not have been previously encountered. The robot could as a result learn more from interactions with the expert, rather than merely copying the expert.

The skills learned by the robot could potentially be greatly diversified if it had a method for composing skills from a set of elementary skills. In this way, a robot could learn to perform simple tasks and then, once these have been mastered, use them together to form composite policies, or adapt them slightly for use in different tasks. This would be a form of using hierarchical strategies as a result of synthesising elementary strategies to ensure robustness in learning to accomplish certain behaviours. In this way, the robot may become proficient in using the skills under variable, and possibly adverse, conditions as well as changing environments. This idea of composing skills is inspired by the concept of curriculum learning, which is discussed in Section 2.4.

This document describes a novel approach towards solving the behaviour acquisition problem in robots or other systems, by using an architecture composed of a system for learning elementary skills from observing expert agents, and then another to synthesise these skills to learn to perform efficiently at more complicated tasks. Chapter 2 presents an overview of related literature in the various fields which were drawn on for this work. Chapter 3 describes the research aims and hypothesis, as well as providing the details of the algorithm that was developed and implemented. Finally, a set of experiments that were run to establish the abilities and restrictions of this system are described, and the results of the algorithm are presented and discussed in Chapter 4. An analysis of these results shows that the performance of the structured strategies generated by the algorithm are superior to the performance of the elementary strategies from which they are constructed.

Chapter 2

Background

2.1 Introduction

There have been many previous approaches to developing agents capable of learning skills, and indeed learning these from an expert, or teacher. However, it is the belief of the author that to date none have embraced a method capable of learning by acquiring structured strategies to perform skills which are robust in that they can withstand a diverse range of conditions and disturbances and still achieve the desired outcomes.

The crux of this research is to extend previous work into learning skills. At the core is a system whereby new skills can be acquired by an agent from the observations of another agent with expert knowledge of the problem at hand. However, this research aims to build on this by incorporating a method for drawing on a combination of these individual skills to enhance their applicability to new scenarios, and so structure this skill acquisition process in a hierarchical fashion.

This chapter reviews previous related research. Section 2.2 discusses previous work which addressed the core challenge of learning from an expert. One approach which is often used in such work for the learning of a policy in goal-directed learning is reinforcement learning. This is covered in more detail in Section 2.3. The idea of using a hierarchical approach towards composing primitive strategies into more sophisticated ones is a form of curriculum learning, which is presented in Section 2.4. In order to synthesise these elementary strategies which have been learned from an expert, this work draws on techniques from the field of game theory, the key components of which are discussed in Section 2.5.

2.2 Learning from an Expert

Learning from an expert is commonly referred to as learning by demonstration or imitation learning, and has recently become a popular technique through which robots can learn to perform a wide variety of tasks [Breazeal and Scassellati 2002]. A framework for imitation learning could be described in three steps [Bakker and Kuniyoshi 1996]: observe an action, represent the action and reproduce the action. The first of these is generally classed as a machine vision problem, and answers the question of exactly what it is that the learning agent is observing and imitating. This subproblem will be considered outside the scope of this research, which instead focuses on reconstructing the motions of a teacher in terms of the basis set of motions available to the student agent.

Imitation learning has been considered through the use of numerous different techniques. Demiris and Hayes [1996] provide early examples where a student robot would follow a teacher around a maze, and so learn the task of maze traversal by formulating rules, with a second experiment on a separate robot platform in imitating the head movements of a human demonstrator. The problem with these methods is that they were generated in an ad hoc, problem specific manner, which is not easily generalisable.

Similar limitations are encountered in the work of Atkeson and Schaal [1997], where a robot could learn how to swing up an inverted pendulum from demonstration, by either learning the parameters of the model of the motion of the pendulum, or learning that model itself. This provides a matching of a single action to that of a demonstrator, but is unsuited to achieving complicated behaviours as it only learns to replicate the model of the action from the demonstrator. Other work, such as that of Billard *et al.* [2004] uses machine learning techniques such as hidden Markov Models to learn actions, but these are again ad hoc and based on merely replicating the actions of the demonstrator. Inamura *et al.* [2004] also use hidden Markov Models to learn motions, but these are represented and recognised as different symbols.

The main problem with these kinds of machine learning based approaches to imitation learning is that they are primarily concerned with mimicking exactly the trajectories of the teacher agent in some configuration space (the space describing all possible positions that may be attained by an agent). An alternative problem encountered in previous methods is that the focus is placed solely on specific local behaviours. These are not global approaches as too much emphasis is placed on particular nuances of

the actions of the teacher agent, leaving the student without a resilient control strategy which would be robust to adversarial conditions and noise.

A further issue to note is the difference between passive learning from demonstration of the expert vs active interaction with the expert. Passive learning allows the expert to be performing the skill as part of some larger routine, whilst active interaction refers to the expert providing the learner with some sort of feedback to indicate how well the learner is performing, and possibly make corrections where appropriate.

A different approach has been suggested by Abbeel and Ng [2004] under the name of apprenticeship learning. This work suggests that the previous methods of imitating trajectories would be insufficient for most real-world applications, owing to the fact that they cannot accommodate for disturbance processes operating in the vicinity of the learner robot. The algorithm presented in this paper poses a learning problem as one of navigating through a sequence of environmental states by selecting the correct actions. The optimal policy for choosing these actions would be learned through a reinforcement learning process (see Section 2.3). Furthermore, the agent would be presented with a set of reward features generated by the expert in solving the same problem, and by use of reinforcement learning, the agent could minimise the difference between the experts features and its own, resulting in behaviour similar to that of the expert.

These results were extended by Syed and Schapire [2008] to adopt a game theoretic approach to learning, whereby the agent is competing against disturbances generated by nature (the environment). A survey of game theory is presented in Section 2.5. The advantage of this alteration is that the performance of the agent is no longer theoretically bounded by that of the expert, and so it would be possible for the agent to learn to outperform the expert. The algorithm presented in this paper is the Multiplicative Weights Algorithm for Apprenticeship Learning (MWAL), and is used as a key component in the research presented in this document. This algorithm is presented in Section 3.3.

Modifications and variants to this algorithm were proposed by Syed *et al.* [2008]. These involved the adoption of linear programming as a solution technique for the optimisation problem under the name of the Linear Programming Apprenticeship Learning algorithm (LPAL). The primary contribution of this work was an increase in convergence speed, but the results were fairly consistent, although LPAL did perform poorly under certain circumstances. It remains an alternative to be considered in future work.

2.3 Reinforcement Learning

2.3.1 Defining the Problem

Reinforcement learning is an example of goal-directed learning, and involves the learning of a behaviour such that some reward function is maximised [Sutton and Barto 1998]. It deals with the interaction of an agent with an environment, such that the agent can repeatedly choose and implement actions, based on the state of the environment. These actions map the system to new states, but not necessarily in a deterministic manner. This accounts for disturbances in the environment. The results of these actions affect the agent by means of a reward function, which serves as feedback to positively reinforce some states and actions, and dissuade the system from using others [Harmon and Harmon 1999]. These rewards can be immediate or delayed. A reinforcement learning problem is often defined in this way over a Markov Decision Process (MDP).

Formally, an MDP is defined as a tuple $M = (\mathcal{S}, \mathcal{A}, \Theta, \gamma, D, R)$ where $\mathcal{S}$ and $\mathcal{A}$ are the finite sets of states and actions respectively, Θ is the transition function such that $\Theta(s,a,s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ defines the state transition probabilities. The fact that this transition depends only on the state s_t at time t, and not on any previous states, is known as the Markov property. Let $\mathcal{A}(s)$ denote the set of actions available from some state s. $\gamma \in [0,1)$ is a discount factor, D is the initial state distribution from which an initial state s_0 is selected, and $R : \mathcal{S} \mapsto \Re$ is the reward function which indicates the desirability of each particular state. An example of an MDP is shown in Figure 2.1.

The objective of a reinforcement learning problem is to learn an optimal policy π^* for traversing the states of an MDP in such a way that the received rewards are maximised. This policy is of the form $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$ subject to the constraint that $\sum_{a_i \in \mathcal{A}} \pi(s_j, a_i) = 1, \forall s_j \in \mathcal{S}$. This implies that the policy provides a mapping from state-action pairs to probabilities, or alternatively, that the policy suggests a probability distribution for which actions should be selected in each state, in order to maximise the total reward.

The total reward accumulated from an episode of T state transitions is called the return. The return starting from a state s_t is calculated as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{2.1}$$

where r_t is the reward received at time t [Sutton and Barto 1998]. The return is discounted by the parameter γ to ensure that R_t does not grow without bounds as $T \to \infty$.

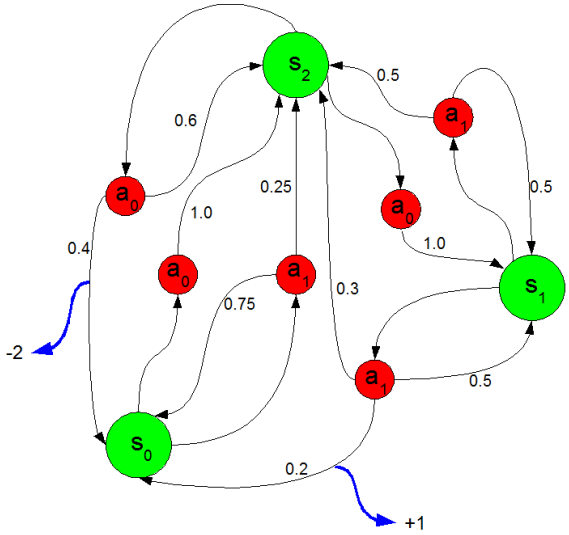

Figure 2.1: A sample Markov Decision Process. The green circles are the states $\{s_0, s_1, s_2\}$ and the red circles are the actions which can be taken at each state $\{a_0, a_1\}$. The outgoing labels at each action are the probabilities of that state transition given that action. The blue arrows correspond to rewards.

The more immediate rewards are thus also weighted more heavily than future rewards. The optimal policy is therefore required to choose a sequence of actions in order to maximise this discounted reward.

2.3.2 Solving for a Policy

There are many approaches to solving a reinforcement learning problem. In order to obtain a solution, two further definitions are required. The state-value function for a policy π is defined as

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} \tag{2.2}$$

$$= E_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s \Big\} \tag{2.3}$$

which is the expected return received from starting in state s and following policy π thereafter. Furthermore, the action-value function for a policy π is similarly defined as

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} \tag{2.4}$$

$$= E_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a \Big\} \tag{2.5}$$

which is the expected return received by the agent from starting in state s where it takes action a, and then proceeds to follow π [Sutton and Barto 1998]. These two functions are used to determine the desirability of particular states and actions under a given policy. An optimal policy π^* is one which would have the highest (and therefore optimal) value functions V^* and Q^* over all states and actions, giving that a maximal return can be received by following the optimal policy rather than any other policy. This is formalised as $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

Algorithm 1: The ε-soft on-policy Monte Carlo algorithm

Input: An MDP M

Output: A policy $\pi(s,a)$

begin

$Q(s,a) \longleftarrow$ arbitrary $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

$Returns(s,a) \longleftarrow$ empty list $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

$\pi \longleftarrow$ an arbitrary policy

for $t = 1, \ldots, \infty$ **do**

Generate an episode using π

for *each pair* s,a *appearing in the episode* **do**

$R \longleftarrow$ return following first occurrence of s,a

Append R to $Returns(s,a)$

$Q(s,a) \longleftarrow$average$(Returns(s,a))$

end

for *each* s *in the episode* **do**

$a^* \longleftarrow \arg\max_a Q(s,a)$

for *all* $a \in \mathcal{A}(s)$ **do**

$$\pi(s,a) \longleftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

end

end

end

return *policy* π

end

The ε-soft on-policy Monte Carlo algorithm [Sutton and Barto 1998] is presented in Algorithm 1. The idea behind this algorithm is to ensure that the policy remains explorative, and continues to sample from actions which may be suboptimal in particular

states. This is ensured as follows: the best action a^* is greedily selected in a state s as the one giving the highest action-value for that state, $a^* = \arg\max_a Q^\pi(s,a)$. However, it is likely to be the case that not all actions have as yet been tried in that state. As a result, the algorithm assigns a small probability ε to every other action in that state, meaning the policy retains the ability to take an action a' that may be better than the currently assumed best, but has not yet been tried. So most of the time the current estimate for the best action will be taken, but another action will be chosen at random with probability ε.

By repeatedly generating new episodes, the algorithm is able to approximate the expected action-value function under the current policy, $E[Q^\pi(s,a)]$. This is an example of generalised policy iteration, and is Monte Carlo in the sense that it estimates the value function through the repeated visiting of chains of states. It has been shown [Sutton and Barto 1998] that as the number of visits to each state tend towards infinity, the approximated value functions converge to their optimal counterparts.

There are more sophisticated Monte Carlo policy iteration algorithms which move the probabilities of selecting the optimal action gradually, rather than immediately choosing greedily, but the asymptotic convergence of Algorithm 1 was deemed sufficient for the purposes of this research.

2.4 Curriculum Learning

It is a widely acknowledged fact that most humans learn better by means of a curriculum [Elman 1993]. Throughout the schooling of a person, that person is gradually introduced to increasingly complicated concepts. It has been hypothesised that a machine learning algorithm could greatly benefit from a similar approach, in terms of both speed of convergence and quality of the optimum [Bengio *et al.* 2009]. The idea is that the iterated increase in difficulty of a task would guide the learning agent towards convergence. Bengio *et al.* [2009] successfully use this curriculum learning approach to train neural networks in a variety of examples, such as gradually introducing noise to data, and shape recognition.

Shaping also refers to a learning task which becomes increasingly more complicated with each instance. Erez and Smart [2008] suggest that this could be successfully implemented in reinforcement learning problems in several ways. The difficulty of the problem could be altered by modifying the reward function, the dynamics of the problem, the internal parameters, the initial state, action space or extending the time

horizon.

An extension of this for learning a control policy would be through the use of motion primitives. Motion primitives represent an alternative view to learning complete actions, and are an extension of the approach of representing different types of motions as symbols. An agent's motion primitives are the simple actions which it can easily perform. These can be used in conjunction as building-blocks to form more complicated behaviours, rather than the agent learning an entire motion sequence from scratch [Choset *et al.* 2005]. This method is used by Nakanishi *et al.* [2004] for the simulation of biped locomotion where the primitives are based on different phase oscillators and the parameters are learned using weighted regression. Schaal *et al.* [2004] establishes the concept of dynamic motion primitives, which are point attractors with nonlinear dynamics. These can be combined to describe complicated motions, the parameters of which can be learned through either supervised or reinforcement learning techniques.

This is a different problem to that solved by hierarchical reinforcement learning [Singh 1992]. Hierarchical RL consists of learning a task at various levels of abstraction, such that more environmental details are included at each subsequent level. In this way a task can be broken down into subtasks which are easier to solve. This is different to the approach considered here which involves composing a hierarchy of known policies into a new policy which can achieve some desired performance in certain unknown conditions.

The representation of actions in terms of a basis of elementary motions still requires a mechanism for learning how to compose these primitives. A solution method for combining a sequence of moves into a viable and robust global strategy in a closed-loop form, which is as yet unexplored, may be taken from game theory. An overview of game theory is presented in Section 2.5. The idea of composition is thus not a new one, however using it within such a learning framework to synthesise robust strategies as presented in this document is, to the knowledge of the author, novel.

2.5 Game Theory

2.5.1 Preliminaries

Game theory is a field which has seen much development in recent years from a variety of different disciplines, such as economics. It is a topic which is based around the idea of multi-agent decision making, and the interactions between these agents as they

attempt to achieve certain goals, which may be cooperative or adversarial [Basar and Olsder 1999]. Although in theory any number of interacting agents may be modeled, for the purposes of this dissertation it is sufficient to deal with only two agents.

Consider such a game of two agents or players, creatively named *Player 1* (P1) and *Player 2* (P2). The game consists of P1 choosing one of N actions, and P2 simultaneously choosing one of M actions. The normal form of a game between these players, is represented as an $(N \times M)$ matrix $\mathbf{A}$, where each element of the matrix corresponds to an outcome of the game, being the reward, or payoff, received by the agents for playing particular actions. Furthermore, consider that the game is zero-sum, meaning that any loss incurred by one player corresponds to the exact gain by the other. The actions available to P1 are the rows of the matrix, and hence P1 is the row player. Similarly, P2 is referred to as the column player. This situation is shown in the matrix below.

		Player 2				
		Action 1	$\cdots$	Action j	$\cdots$	Action M
Player 1	Action 1 $\vdots$ Action i $\vdots$ Action N	a_{ij}				

In this case a_{ij} is the payoff received by P1, when P1 plays action i and P2 plays action j. Owing to the fact that this is a zero-sum game, the payoff to P2 would be $-a_{ij}$. In this set up, P1 wishes to receive as much payoff as possible, and is hence also known as the maximising player. By also attempting to maximise personal gain, P2 must decrease the gains of P1 and is thus the minimising player.

One common description of a solution to such a game is a Nash equilibrium solution. This is an action pair for the two players (i^*, j^*) with $i^* \in [1,\ldots,N]$ and $j^* \in [1,\ldots,M]$ such that neither player can gain in reward by unilateral deviation from this strategy [Basar and Olsder 1999]. If the game is played repeatedly and the equilibrium is obtained by P1 continually selecting a particular row i^* while P2 selects the j^{*th} column then i^* and j^* are known as pure strategies. An alternative type of solution is a mixed strategy. This means that there is no single action which guarantees an equilibrium. Instead, the player should choose between a subset of its actions with some probability distribution. Thus the players could be said to be playing an equilibrium solution if they are selecting actions according to this optimal probability distribution.

Let p^* and q^* be the probability distributions over the actions available to P1 and P2 respectively which give a Nash equilibrium. Then the optimal payoff given from P2 to P1, otherwise known as the value of the game, is defined as $v^* = p^* \mathbf{A} q^*$ [Nisan *et al.* 2007].

2.5.2 Learning in Games

Much research has been conducted into games played by two players in an attempt to reach equilibrium, because finding a Nash equilibrium is a non-trivial task. There are several different methods for achieving this in a repeated game. Fictitious play is a simple learning rule which involves an agent choosing a best response to its opponent based on the empirical frequencies of the actions taken by that opponent. Extensions to this involve using additional information such as approximations of the gradients of action frequencies [Shamma and Arslan 2005].

Another related approach involves identifying and matching patterns in the most recent moves made by the other player [Lambson and Probst 2004]. A best response strategy can be played using the history of the other player in this way. Young [2009] presents a different approach for when the payoffs are not all known, in the form of interactive trial and error learning to determine effective long-term strategies. This works by the agent selecting actions in an explorative manner to determine a best strategy, but is also sensitive to a strategy no longer being the best one if the behaviour of another player changes. Other approaches include generating and testing hypotheses to describe the behaviour of the opponent [Foster and Young 2006].

Regret minimisation [Nisan *et al.* 2007] is another computational technique for finding a mixed equilibrium solution. For each time the game is played, the payoff for the current approximation of the optimal mixed solution is compared to the payoffs which would have been obtained by playing any of the available actions. These are known as regrets. The approximate solution is then updated in some way to draw more heavily on the actions with higher regrets.

The multiplicative weights algorithm [Freund and Schapire 1999] is one such algorithm. The learning player starts with an arbitrary initial strategy p_1. At each round of play t, the learning player updates its mixed strategy with the following computation

$$p_{t+1}(i) \quad = \quad p_t(i) \frac{\beta^{M(i,q_t)}}{Z_t} \tag{2.6}$$

where $p_t(i)$ is the weight of the i^{th} pure strategy of the learning player in its approximation of the optimal solution at time t, $\beta \in [0,1)$ is a parameter of the algorithm, $M(i,q_t)$

is the regret from not having played the i^{th} pure strategy against the strategy q_t which was used by the opponent at time t, and Z_t is a normalisation factor. This algorithm has been shown to be optimal under the correct conditions [Freund and Schapire 1999].

2.6 Conclusion

This chapter examined a number of techniques that could be used in the process of learning control policies. Learning from demonstration remains the one key focus of this work. It has been addressed from several different angles, and has achieved much success in that the trajectories of an expert can be learned by the apprentice. However, a common limitation of these methods is that the learned strategies are not robust in the face of unknown conditions and adversaries. This challenge would be overcome with the incorporation of curriculum learning. By assembling a basis of simpler tasks, it is hoped that the agent could utilise and combine these in some way so as to increase the sophistication of the learned skills.

Two valuable techniques used in this work are reinforcement learning and game theory. Reinforcement learning provides a technique whereby optimal policies can be learned for specific situations, using feedback in the form of reward functions. Game theory then offers approaches towards the mixing of different strategies which may perform better of worse against different adversarial behaviours.

The following chapter draws on the concepts and principles outlined in this literature review in the construction of a hypothesis and algorithm for the learning of structured control policies from demonstration.

Chapter 3

Hypothesis and Architecture

3.1 Introduction

Although learning proficiency at an arbitrary skill, from initially having no knowledge about that skill, is not a trivial task, it is important that robots or other agents be equipped with some mechanism for broadening their range of skills. As illustrated in Section 2, learning a skill by watching or interacting with an expert has been shown to result in the successful transference of skills from teacher to student.

The problem is that traditionally this teaching process has not been structured, and the apprentice does not synthesise the acquired knowledge to result in skill sets which are robust in the presence of arbitrary disturbances, and successful in a range of unpredictable circumstances. An approach to overcome this drawback in learning methodologies is presented here, in the form of a hierarchical learning system, where a skill is taught to the agent in various elementary scenarios, and these skills are subsequently combined to extend the domain of applicability of the learned skill to unknown and more sophisticated environments.

This chapter presents the methodology of the research. Section 3.2 formalises the hypothesis addressed by this research, and links that with an overview of the hierarchical approach taken. The details of the method for learning a task in the base cases from interaction with an expert is discussed in Section 3.3, and the method used for the composition of these skills to manage more complicated scenarios through the use of game theoretic techniques, is detailed in Section 3.4. The details of the integration of the complete system comprising these two stages are presented in Section 3.5.

Chapter 4 then proceeds to present a sample domain of navigating a car through traffic on a multi-lane road, and illustrates the performance of the algorithm in this

setting. These results draw attention to particular findings such as the ability for the algorithm to draw primarily on a particular base scenario when that is presented as an 'unseen' case, as well as other intuitive mixing of strategies.

3.2 Hypothesis

The general problem addressed in this dissertation is one of apprenticeship learning [Abbeel and Ng 2004], otherwise known as learning from demonstration, or imitation learning. The goal of apprenticeship learning is for an apprentice, or student agent, to learn a skill in order to complete a specific task from the observation of an expert agent, or teacher. In this context, an expert is considered to be some agent with a sophisticated control policy for performing the task at hand. The apprentice agent is equipped with a set of basis actions which it knows how to perform, and is required to synthesize these into a composite policy, based on the behaviour of the expert in the problem domain. Often, in imitation learning, it is assumed that the apprentice and expert have the same considerations, but in general this may not be true at all. For instance, the expert may be trying to minimise solution time, but this may be of no concern to the apprentice. Furthermore, even the action set available to each agent may be different.

An important component of such a learning system would be a mechanism for observing the expert, and translating its behaviours into movement through a state space. This 'vision' task is problem-specific and would depend on the type of robot and the sensors available to it. This project aims at the development of a general algorithm for learning control policies, and thus assumes sensory input has been preprocessed to allow the algorithm to operate exclusively in the domain of trajectories through state space. This state space could be considered to be a configuration space enumerating all the possible physical configurations of the agent, and is interpretable as a manifold, or high dimensional surface that locally (and not necessarily globally) resembles Euclidean space. Trajectories in this space therefore describe the motion of the agent through a sequence of poses [Choset *et al.* 2005], and so learning a control policy at this level of abstraction corresponds to the acquisition of a skill by the agent.

The algorithm constructed to address this learning problem consists of two stages. The first involves the generation of policies for performing some task in a set of base cases. The method whereby this is done draws on reinforcement learning as well as regret minimisation in game theory to imitate the expert agent. The second stage is concerned with presenting the agent with an unseen scenario, and the agent is expected to

synthesise policies learned in the first stage to successfully perform the task in the new setting. This stage again utilises game theory, with regret minimisation used to find a Nash Equilibrium between the policies used by the agent, and adversarial conditions presented by the environment.

The key reason for invoking a game theoretic approach lies in the fact that the conditions faced by the learning agent in some environment are likely to be adversarial. For a learning approach to deal with such a wide range of environmental effects as may be encountered by the agent would typically require a large amount of knowledge about the 'world' encountered by the agent, and the dynamics thereof. However, the game theoretic approach allows for the use of procedures which could form approximations to a Nash equilibrium with very little information about the processes in this environment. It is thus an approach which alleviates some drawbacks in dealing with uncertainty using other modeling paradigms, as the ultimate goal of this line of research would be to have an algorithm which could learn from and respond to a potentially infinite and constantly changing environment.

This approach is thus summed up as:

1. Given a task to be learned, a set of problem domains $\{B_1, B_2, \ldots, B_N\}$ on which that task can be performed, and an expert with a policy corresponding to each domain $\{\pi_{E_i}\}(i = 1 \ldots N)$: the agent learns a policy $\{\psi_i\}(i = 1 \ldots N)$ from the expert for each domain.

2. Given a new problem domain $\tilde{B}$ for the same task, synthesise a policy for this domain as a function of the previous policies $\tilde{\psi} = f(\psi_i)$.

The proposed framework for this architecture is illustrated in Figure 3.1. This indicates that observing and learning from an expert would allow the agent to acquire a primitive skill. Once a library of these primitive skills has been assembled, they can be synthesised using a different mechanism to generate new variants of the skill which are suitable for different conditions to those initially demonstrated by the expert.

The hypothesis of this research concerns the viability of using such an approach to robustly learning a skill, and can be formalised as

- Using a synthesised, hierarchical approach to apprenticeship learning, using interactions with an expert to learn a basic set of strategies and drawing on game theoretic techniques for composing these to obtain an equilibrium, an agent can successfully learn a complex strategy for performing some task under an arbitrary set of conditions and natural disturbances.

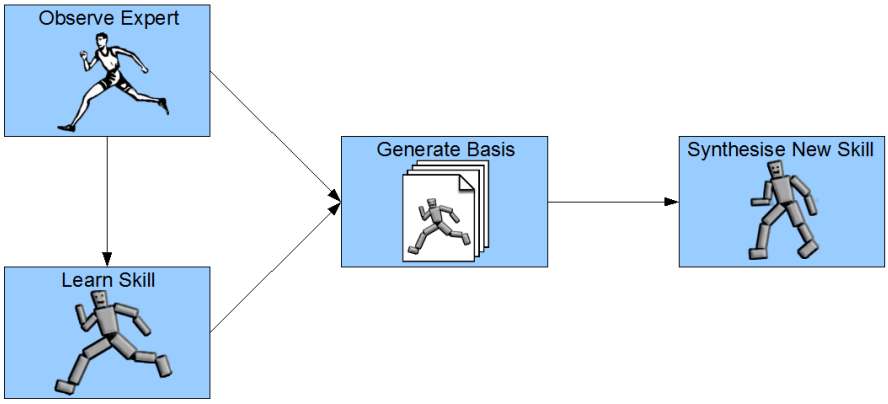

Figure 3.1: The proposed framework

This hypothesis should now be formally considered and clarified. The task to be completed should be arbitrary, that is, the learning algorithm should make no assumptions about the nature of the task, other than that the expert from which the agent learns would have some (not necessarily optimal) strategy for performing that task. Strategies should be demonstrated and learned in simple cases, but then the agent should combine these elementary strategies to be able to handle more sophisticated tasks. Robustness is ensured using a game theoretic approach, as this allows the agent to synthesise strategies in order to respond to worst case adversarial conditions.

The success of such a method seems difficult to determine, as proficiency at a task is difficult to rank. However, if the agent is able to improve performance over any of the base strategies with a combined one on an unseen case of the problem, and yield results similar to those obtained by an expert agent, then the algorithm could be deemed successful.

The two stages of the algorithm, the acquisition of elementary skills and the synthesising of these skills on different instances of a problem task, are discussed at length in Section 3.3 and Section 3.4 respectively.

3.3 Primitive Skill Acquisition

A task to be learned by the agent is posed as a Markov Decision Process (MDP) consisting of multiple states, and actions to transition between them. Actions are chosen and implemented by the agent (as primitive actions available to the agent). Additionally, the transitions between states are also subject to environmental events, where the environment is considered to be everything external to the agent. These disturbances are manifested by the fact that state transitions are probabilistic, ie. selecting a particular action in a particular state could result in one of several possible outcomes. The domain of the problem can be changed by varying the rewards on the transitions, making some routes more favourable than others.

The objective of the agent is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$ which maps states to a probability distribution over the actions that may be considered in each state. The problem of learning such a policy as it is posed here is a problem from the field of reinforcement learning, and could be solved using a number of techniques such as policy iteration or value iteration. However, consider now that the true reward function $R^*(s)$ is unknown. Instead, assume that the learning agent has available a vector of features $\phi : \mathcal{S} \mapsto [-1,1]^k$ which describes the desirability of particular features of a state. Generally, if the MDP represents the position of the agent on a manifold in some configuration space, the features represent properties of the manifold at different locations. Now let the true reward function be a linear combination of these features such that $R^*(s) = w^* \cdot \phi(s)$ where $w^* \in \Re^k$ is an unknown weight vector. The apprentice is required to learn this weighting from observations of the expert.

This weight vector can be learned together with the policy by the Multiplicative Weights for Apprenticeship Learning (MWAL) algorithm [Syed and Schapire 2008]. This approach defines a feature expectations vector as the expected, cumulative, discounted feature values of a policy π as

$$\mu(\pi) \quad = \quad E\Big[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi, \Theta, D\Big] \tag{3.1}$$

Given an expert's policy π_E, the goal is to find a policy π for that agent, such that $V(\pi) \geq V(\pi_E) - \varepsilon$, where $V(\pi)$ is the value of a policy π on the MDP M,

$$V(\pi) \quad = \quad E\Big[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) | \pi, \Theta, D\Big] \tag{3.2}$$

and therefore $V(\pi) = w^* \cdot \mu(\pi)$.

The goal policy may be either a stationary policy π or a mixed policy ψ, being a distribution over the set of all stationary policies. The approach taken by the MWAL algorithm views the decision making process as a two player zero-sum game, where the maximising player is the learning agent selecting a distribution over the policies, and the minimising player is the environment which selects the reward function by changing the values of w. The maximising player wishes to find a mixed policy ψ to maximise $V(\psi) - V(\pi_E)$ with unknown and potentially worst-case choice of w. The fact that the weights of the constituent elements of the reward function are unknown and this reward function could therefore be selected in an adversarial manner by the environment, suggests that this problem be posed as a game. This game can then be represented as

$$v^* = \max_{\psi \in \Psi} \min_{w} [w \cdot \mu(\psi) - w \cdot \mu(\pi_E)] \tag{3.3}$$

$$= \max_{\psi \in \Psi} \min_{w} w^T \mathbf{G} \psi \tag{3.4}$$

with

$$\mathbf{G}(i,j) = \mu^j(i) - \mu_E(i) \tag{3.5}$$

where $\mu_E = \mu(\pi_E)$, $\mu^j = \mu(\pi^j)$ for π^j the j^{th} stationary policy, and $\mu(i)$ is the i^{th} component of μ.

The algorithm solves this game using the regret minimisation procedure of iteratively adjusting the weights w, using a multiplicative weights algorithm for solving large games [Freund and Schapire 1999] and then finding an optimal policy ψ for the current weights [Sutton and Barto 1998]. This approach has been shown to result in successfully acquired skills. Furthermore, because the apprentice learns exclusively from the feature expectations of the expert μ_E, the skill acquisition of the apprentice is independent of not only the control policy used by the expert, but also its state and action space. The apprentice may therefore be equipped with only a crude approximation of the actions available to the expert, yet still be able to learn the demonstrated skill.

Algorithm 2 presents the pseudocode for this process of learning base policies, reproduced from Syed and Schapire [2008]. The algorithm iteratively computes the optimal policy for the current reward function $R(s) = \mathbf{w}^{(t)} \cdot \phi(s)$ using a method such as an on-policy Monte Carlo control algorithm (see Algorithm 1), and then adjusts the reward function, by changing the weights w to minimise the regret incurred from sub-optimal playing of this game.

Algorithm 2: The MWAL Algorithm

Input: An MDP M, and an estimate of the feature expectations of the expert, $\hat{\mu}_E$

Output: A mixed policy $\bar{\psi} = \{\hat{\pi}^{(t)}\}$ for all $t \in \{1,\ldots,T\}$

begin

$\beta \longleftarrow \left(1+\sqrt{\frac{2\ln k}{T}}\right)^{-1}$

$\mathbf{W}^{(1)}(i) \longleftarrow 1$ for all $i = 1,\ldots,k$

$\mathbf{G}(i,\mu) \longleftarrow ((1-\gamma)(\mu(i)-\hat{\mu}_E(i))+2)/4$

for $t = 1,\ldots,T$ **do**

$\mathbf{w}^{(t)}(i) \longleftarrow \frac{\mathbf{W}^{(t)}(i)}{\Sigma_j \mathbf{W}^{(t)}(j)}$ for all $i = 1,\ldots,k$

Compute optimal policy $\hat{\pi}^{(t)}$ w.r.t. $R(s) = \mathbf{w}^{(t)} \cdot \phi(s)$

Compute an estimate $\hat{\mu}^{(t)} \approx \mu(\hat{\pi}^{(t)})$

$\mathbf{W}^{(t+1)}(i) \longleftarrow \mathbf{W}^{(t)}(i)\beta^{\mathbf{G}(i,\hat{\mu}^{(t)})}$ for all $i = 1,\ldots,k$

end

$\bar{\psi} \longleftarrow \{\hat{\pi}^{(t)}\}$ for all $t \in \{1,\ldots,T\}$ with probability $\frac{1}{T}$ for each

return *mixed policy* $\bar{\psi}$

end

The output of this algorithm is a mixed policy, with the use of which the agent can perform almost as well as, if not better than, the expert on the given task and domain, in terms of the reward function composed of the feature vectors.

3.4 Composite Skill Acquisition

Valuable questions to ask at this point would be the following: assuming an agent had acquired the ability to perform some skill in various instances, what would happen if a new, previously unseen scenario was encountered and the agent could not rely on an expert for a demonstration in this setting? Is there a way in which the agent could draw on the previously learned cases to synthesise a control policy for dealing with this new scenario?

These questions are equivalent to considering the incorporation of a curriculum into the learning schedule of the apprentice. This addition involves an expert demonstrating the use of a skill in a limited setting as before, and once the apprentice had mastered that skill, the expert would advance to a more challenging setting. In this way, the apprentice would be learning strategies under varying conditions. The result would be the acquisition of a complicated skill – one which the apprentice may not have been

able to learn from a single training case, but which could be applied in a new scenario.

This section presents this extension to the aforementioned algorithm through the use of such a curriculum. In this way, elementary strategies learned from observation of the expert must be composed to form composite strategies for different cases of the problem. The policies learned in the first phase are used as the input to the next stage of the learning task, where the apprentice combines them in order to learn a slightly different variant of the skill. A difficulty to consider at this point is that both the state space and the action space is likely to be different from one scenario to the next. The assumption is made that these are however kept constant across all instances of the learning problem, but the relaxation of this assumption could be the subject of future work.

The method whereby this is accomplished is to pose the problem as a game. This is an intuitive approach, as the agent wishes to synthesise the learned policies so as to maximise the reward gained from the MDP of the new task instance in the face of nature (the environment) which is acting as an adversary. As the behaviour of the environment is unknown, it makes sense to optimise against the worst-case scenario. An appropriate language for such adversarial optimisation can be found in game theory.

Consider that in the first stage of the algorithm (Algorithm 2), the agent was presented with a set of problem instances $\{B_1, B_2, \ldots, B_N\}$ in which the agent learned policies $\{\psi_1, \psi_2, \ldots, \psi_N\}$ from the expert for each instance. The agent is now presented with a new instance $\tilde{B}$ for the same task, and is required to learn a new policy $\tilde{\psi} = f(\psi_1, \psi_2, \ldots, \psi_N)$ to successfully complete this task.

Now consider a two-player repeated matrix game $\mathbf{G}$ defined as follows: the agent is the row player which hopes to maximise its reward in using its skill on $\tilde{B}$ in the presence of the column player – nature. The actions which may be played by the agent are the set of N primitive policies $\Psi = \{\psi_1, \psi_2, \ldots, \psi_N\}$, and the M actions which may be played by nature to present adversarial conditions are denoted $\mathbf{Q} = \{Q_1, Q_2, \ldots, Q_M\}$. The value of the game matrix for using the policies (ψ_i, Q_j) is $\mathbf{G}(i, j)$ and is a function of $\tilde{B}$. The solution to this game would be to find a Nash equilibrium (ψ^*, Q^*) such that unilateral deviation from this policy can not result in a better outcome.

The method used to find this mixed equilibrium is a regret minimisation technique, and is in fact based on the MWAL algorithm (Algorithm 2) of Syed and Schapire [2008] as well as the multiplicative weights algorithm of Freund and Schapire [1999]. The Strategy Composition Algorithm is shown in Algorithm 3. $E[R(P,Q)]$ is the expected reward of playing policy P on the MDP $\tilde{B}$ in the presence of nature's adversarial

Algorithm 3: The Strategy Composition Algorithm

Input: An MDP $\tilde{B}$, and a set of primitive strategies $\Psi = \{\psi_1, \psi_2, \ldots, \psi_N\}$

Output: A mixed policy $\tilde{\psi} = \mathbf{w} \cdot \Psi$

begin

$\beta \longleftarrow \left(1 + \sqrt{\frac{2 \ln N}{T}}\right)^{-1}$

$\mathbf{w}^{(1)}(i) \longleftarrow \frac{1}{N}$ for all $i = 1, \ldots, N$

for $t = 1, \ldots, T$ **do**

$\tilde{\psi} = \mathbf{w}^{(t)} \cdot \Psi$

$\mathbf{G}(\Psi, \tilde{\psi}, \mathbf{Q}) \longleftarrow ((E[R(\tilde{\psi}, \mathbf{Q})] - E[R(\Psi, \mathbf{Q})]) + 1)/2$

$\mathbf{W}(i) \longleftarrow \mathbf{w}^{(t)}(i)\beta^{\mathbf{G}(\psi_i, \tilde{\psi}, \mathbf{Q})}$ for all $i = 1, \ldots, N$

$\mathbf{w}^{(t+1)}(i) \longleftarrow \frac{\mathbf{W}(i)}{\sum_j \mathbf{W}(j)}$ for all $i = 1, \ldots, N$

end

return *mixed policy* $\tilde{\psi} = \boldsymbol{w}^{(T+1)} \cdot \Psi$

end

policy Q.

The algorithm starts with a mixed policy consisting of a uniform distribution over the primitive policies, and relies on an iterative procedure to update this distribution by multiplicatively allocating more weight to the policies which perform better on $\tilde{B}$. The output is a mixed policy consisting of a distribution of the primitive policy set Φ giving an equilibrium strategy for $\tilde{B}$.

This composed strategy $\tilde{\psi}$ is a linear combination of the policies in Ψ. Assume strategy ψ_i selects an action a_j in state s_k with probability p^i_{kj}. Then, given a final mixed strategy weighting $\mathbf{w}^{(T+1)} = \{w_1, w_2, \ldots, w_N\}$ with $\sum_i w_i = 1$, then policy $\tilde{\psi}$ would choose action a_j in state s_k with probability $\tilde{p}_{kj} = \sum_i w_i p^i_{kj}$. The weights are biased towards the policies that achieve a higher expected reward on $\tilde{B}$.

Using regret minimisation, Algorithm 3 thus learns to rely more heavily on the strategies which generate higher rewards on this unseen MDP.

It is important to note that this is not simply a case of state estimation, and applying the appropriate policy. The agent does not have knowledge of the conditions (rewards) of this unseen MDP and so cannot directly apply a policy. An alternative approach would be to employ a form of vision and identification system to recognise the MDP and choose the best policy. However, the premise of this work is that there are too many possible conditions to anticipate each scenario and thus this game-theoretic approach is used to synthesise the best policies.

3.5 Complete Learning System

The full system is composed of the two algorithms detailed in Sections 3.3 and 3.4 respectively. Repeated learning of a skill in different scenarios with Algorithm 2 would establish a basis of strategies. A combination can be learned from these to give a control policy for performing that skill in a new setting, with arbitrary disturbance processes. Consequently, the output from the MWAL Algorithm is used as input to the Strategy Composition Algorithm.

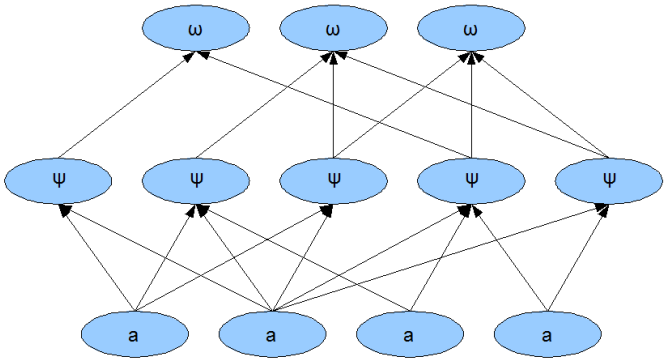

Figure 3.2: Policy composition example. Here four basic actions (a) are used to create five elementary policies (ψ), which in turn are synthesised into three composite policies (ω).

The same set of basis strategies could be used to learn synthesised strategies for many different scenarios. An example of the complete learning process is illustrated in Figure 3.2. The actions of the MDP (denoted a) are composed into primitive strategies, denoted in the image as ϕ. This is the extent of previous work such as that of Syed and Schapire [2008], and represents supervised learning guided by the expert. These primitive strategies are in turn synthesised to form policies for more complicated scenarios, as shown in the image by the ellipses labeled ω. This is an unsupervised learning task. It can thus be seen that this solution technique is hierarchical in nature, and consists of two different forms of learning task. The result is a rich set of synthesised policies, which are more complicated than would have been possible to learn from the expert

generating basis strategies alone.

The two phases of this combined algorithm provide a mechanism whereby the learning agent could be guided through the acquisition of a basic set of skills by a teaching agent, and then extrapolate these skills to more sophisticated settings with an unknown adversarial environment.

As an extension to this, one could envision additional steps in the hierarchical process, whereby not only are these synthesised strategies further composed to handle even more elaborate scenarios, but this strategy composition process could also be guided by an expert, as was the case for the MWAL Algorithm. This remains the topic of future research.

3.6 Conclusion

The ability to learn different behaviours and skills is important for robots to play a useful role in society as it suggests adaptability and versatility. One intuitive method for accomplishing this is through the use of learning by demonstration. As a result of this, a robot should be able to acquire new abilities by watching an expert. Ignoring the vision problem of mapping agent configurations to states, the problem becomes one of learning optimal trajectories on the surface of a manifold which represents the configuration space of the agent. Learning a policy in this setting is equivalent to formulating a probabilistic mapping from states to actions on an MDP.

By drawing on the prior knowledge of an expert, and observing the expert utilising a skill, a learning agent is able to acquire proficiency in that particular skill for a given set of conditions. Repeating this process in a variety of different conditions provides the agent with the set of skills. Using these as a basis, and playing a repeated game of strategy selection against nature, the agent can synthesise these skills and deduce a mixed strategy which would to enable it to respond to novel scenarios.

The algorithm presented in this chapter is composed of two phases. The first learns a policy from an expert on an MDP, and the second combines these policies to add robustness and the ability to generate different variants of the skill. The following chapter discusses the implementation and experiments that were conducted using this algorithm, and presents the results which were produced. These results are used to determine the extent to which this system could be considered successful.

Chapter 4

Results and Discussion

4.1 Introduction

Chapter 3 outlined the creation of an algorithm which was based on the principles of learning from an expert, reinforcement learning, game theory and curriculum learning, all of which are discussed in Chapter 2.

The algorithm uses a multiplicative weights process and guidance from the expert to learn both the reward function for a particular skill and an optimal policy for maximising the rewards. This is repeated for each training scenario presented to the apprentice agent in order to create a basis of learned skills in different settings. The second phase of the algorithm is then concerned with using this basis of strategies to synthesise new strategies for unseen scenarios. This is again implemented with a multiplicative weights regret minimisation approach, in order to determine an optimal contribution of each strategy in the basis, to the new strategy.

Implementation of these two phases of the algorithm was done in MATLAB, and the code for each section can be found in Appendix A and B respectively.

This chapter describes and presents the experiments and corresponding results which are used to demonstrate the algorithm in practice. Section 4.2 outlines the details of the experiments and domain used to test the algorithm. Section 4.3 provides details of the results of the first phase of the algorithm, which deals with the learning of basis policies from an expert. The second phase, that is the synthesis of new strategies in unseen environments, is demonstrated in Section 4.4 together with the results and discussion of this policy synthesis. Finally, an analysis of the overall performance of the algorithm is presented in Section 4.5.

4.2 Sample Learning Experiment

4.2.1 Problem Formulation

In order to test the algorithm empirically, a test problem was established. Inspired by the examples used by Abbeel and Ng [2004], Syed and Schapire [2008] and Syed *et al.* [2008] the skill to be learned was one of navigating through traffic. The agent consists of a single car on a multi-lane one-way road. Assume all traffic is moving at a fixed speed, and the agent travels faster than the rest of the traffic. The net effect is that all the other cars on the road are moving towards the agent. The other cars do not exhibit any complicated behaviour such as changing lanes themselves. The goal of the agent is to learn to navigate on such a road while minimising collisions with other vehicles. This situation is demonstrated in Figure 4.1.

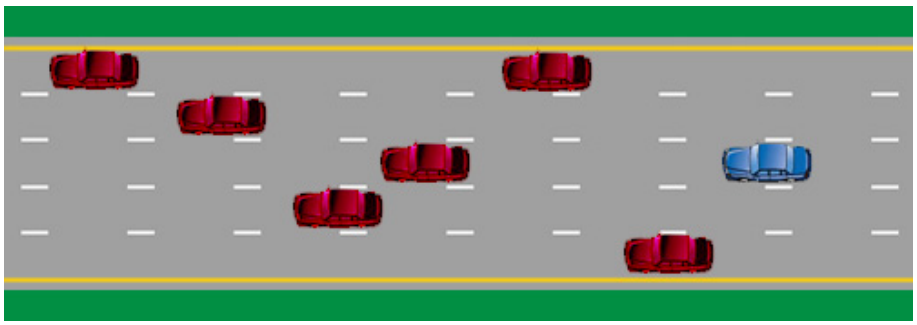

Figure 4.1: An illustration of the car navigation problem

As an extension to the example cited in the aforementioned three papers, consider that the navigability of the road may change at different points. For simplicity, two approximations are made. The first is that the road is discretised by car length. The result is that a stretch of the road can be viewed as a two-dimensional grid, where the length is the number of car lengths, and the width is the number of lanes. The second approximation is that the road is considered to be either paved or not at each point, with a high and low reward for using such a portion of the road respectively. This could be easily generalised so the road could have intermediate levels of navigability. As an example consider Figure 4.1 where there are seven lanes, but the entirety of the outer two lanes are unpaved (and so one would rather have the agent using these only if it cannot be avoided).

There are two different sources of randomness encountered by the agent in this setting. The first is the layout of the terrain over which the agent drives. This is a parameter of the problem, and corresponds to different conditions under which the

agent must learn to drive. The second is other cars on the road. These act as adversarial disturbances to the learner, and are generated by nature by means of some unknown process. The agent must therefore learn to drive optimally on a given terrain in the face of adversarial traffic conditions, where optimally refers to its attempts to maximise time spent on the paved sections of the road.

An important consideration is the extent to which this environment is observable. Consider that the agent learns a skill on a predefined stretch of road. The conditions of the surface are learned from experience and through observation of the expert. The rewards associated with different parts of the road do therefore not contribute towards the state space of the agent. If each section of road had a navigability reward $r \in [-1,1]$ then the resulting state space would be unwieldy in size.

Now consider the other cars on the road. The avoidance actions taken by the agent are dependent only on the cars in some nearby horizon. For example, if a collision with a car c_i is imminent, it is irrelevant to the agent whether or not there is a column of cars behind c_i. For that reason, only the distance to the closest car in each lane need be recorded. This has some correspondence to an actual driving scenario where distances to obscured vehicles would be inaccurate at best.

Furthermore, the state of the agent obviously includes its position on the road, in order for it to judge relative positions to other vehicles and receive the appropriate road navigability signals.

The complete state space of the agent for this example is then defined as

$$\mathcal{S} \quad = \quad agent\_position \times car\_positions$$

and the action space for the agent is

$$\mathcal{A} \quad = \quad \{move\_left, no\_move, move\_right\}.$$

The policy to be learned by the agent is then $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ which gives the probability of each action being chosen in each state.

The next aspect to be defined are the features which describe the performance of an agent, be it the learner or the expert. The important considerations in this problem are the lanes used, as well as the collisions. For this reason, there is one feature for each lane, and one for a collision. Use of a paved section of road yields a reward of 0.5, whilst an unpaved section has a payoff of -1. Similarly a collision contributes a payoff of -1 and all non-collisions 0.5.

This example domain, while tangible and easily visualised, is a sample of the kind of domain that would be encountered in a robotics example. Instead of the road, one

could imagine a multi-dimensional manifold representing a configuration space of a robot. The regions of paved or unpaved road could be interpreted as dynamics, conditions and constraints on the surface. Finally, the other cars represent adversarial disturbances which may be encountered by the robot and perhaps are not foreseen or predictable. In this way, learning to drive on this multi-lane road would be equivalent to learning any other skill on an arbitrary manifold for a robot.

4.2.2 Establishing a Basis

The learning agent must be presented with a full basis of different scenarios on which to learn behaviour from the expert, in order to be able to synthesise them to form new strategies for unseen terrains. The basis was selected by the author as a set of intuitive road shapes, but this is by no means meant to be a 'best' basis or cover all eventualities.

In fact, the converse should be true. By using an incomplete basis, the agent should still be able to synthesise some mixed strategy which would perform better in an unseen situation than any of the pure strategies. It is unrealistic to expect that a complete and comprehensive basis for all complex scenarios that could be encountered in a skill would be enumerated, hence the need for this learning mechanism.

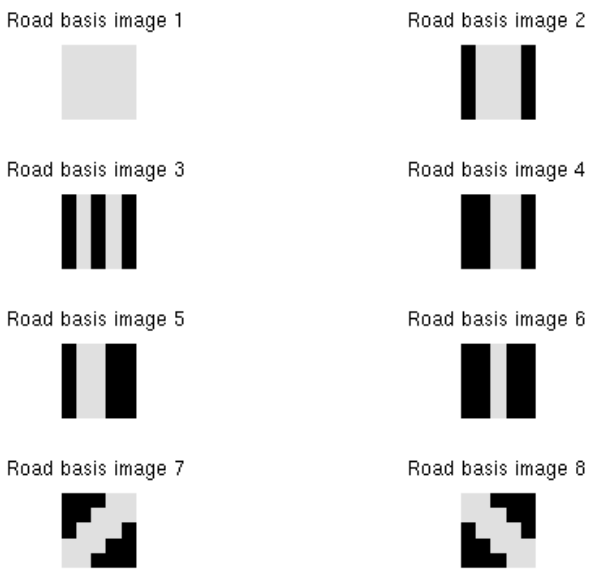

Figure 4.2: The set of basis roads used in the experiments. The dark areas depict unfavourable 'unpaved' regions, and the light areas show the 'paved' regions providing higher reward.

For these experiments, a basis of roads sized 5×5 were selected. That means that each road in the basis consisted of five lanes, with each lane being represented by five car lengths.

The set of basis roads is shown in Figure 4.2. As can be seen, this includes roads of different widths, different offsets, a split road and two diagonal roads. Each element of the basis which is not symmetric about the central lane has its symmetric equivalent included in the basis as well. Let these basis roads be denoted $R1$ through $R8$ respectively.

4.3 Primitive Skill Acquisition Results

The first stage of the algorithm involves learning optimal strategies for performing a skill in a known environment. This skill is learned from the observation of the performance of an expert agent in that setting.

For the experiments presented in this section, the sample roads were all sized 5×5, as mentioned in the previous section. The size of these environments is small, and was restricted by the time available for computations. Although the performance of the algorithm should ideally be examined in a larger setting, the results obtained are sufficient to demonstrate the success of the algorithm, as well as draw attention to several aspects of its performance.

The roads used for training in the basis are shown in Figure 4.2. These were chosen not specifically to form a complete basis, but in a more ad hoc manner to allow for different properties, such as varying directions of lanes and road widths. This is used to demonstrate that it is not necessary to have a perfectly crafted basis in order for a learning agent to devise a method for navigating a new terrain. For each iteration of each experiment, the starting location for the agent was drawn uniformly from the 5 lanes.

The reason that the number of collisions made by the apprentice is not completely reduced to zero is because the Monte Carlo policy iteration uses soft policies. That means that a small chance of choosing an alternate policy over the currently determined best one remains. This could be easily altered if a collision was considered worse than leaving the road for the agent. However, as it stands, both these actions are considered equally bad as far as the agent is concerned.

4.3.1 Learning a Basis

This section describes the results from having learned policies for each of the eight component roads of the basis. In each case, the algorithm was run for 100 iterations. For each iteration of every experiment, both agents were started at a position along the top of the road, selected uniformly from all five lanes. The following images each consist of three sub-images. The first of these, figure (a), depicts the road on which the apprentice is being trained. Light areas correspond to paved areas in the road, and dark areas are unpaved. Figure (b) then shows the six features for that policy ($\mu(\pi)$) where the first five attributes correspond to the usage of each lane, and the sixth is a lack of collisions. All are weighted by the appropriate rewards from the terrain. These are the features used to train the apprentice, and correspond to the total quantity of reward received by the agent for each attribute (spending time in that lane or avoiding collisions). Finally figure (c) shows the lane use statistics and collisions of the final trained policy when being executed for 100 iterations on that road. Note the difference that in (b), the sixth attribute refers to the rewards obtained from a lack of collisions, whereas the sixth attribute in (c) is a count of the collisions themselves. In both figures (b) and (c) the red line refers to the expert, and the green line the apprentice.

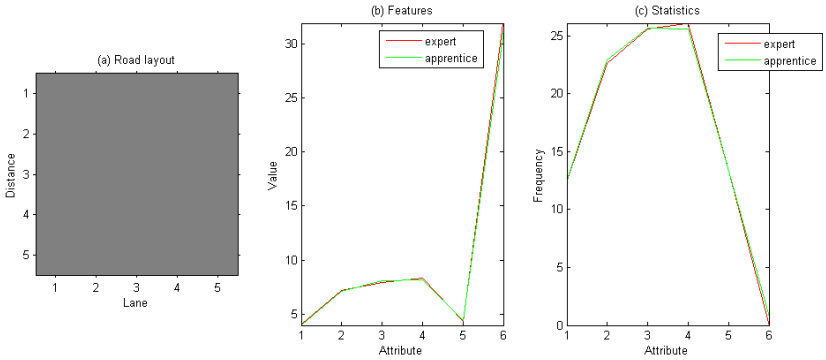

Figure 4.3: Primitive Policy 1

The ‘experts’ used in these cases consisted of simple heuristics, and so perhaps did not use the best imaginable policies. The heuristics involved rules such as:

- In case of impending collision, change lanes to a side that would not result in a collision.
- If either side is safe, choose the one with the highest terrain cost.
- If there are no impending collisions, change lanes if either adjacent lane is safe and would provide a higher reward.

The first element of the basis is shown in Figure 4.3. This shows that although the expert used all the lanes, the central three lanes were preferred, with a slight skewing to the right. The apprentice did however learn this distribution almost exactly from the expert, and attribute 6 shows that the discrepancy in the collisions made by the two agents was minimal – both avoided almost every other car.

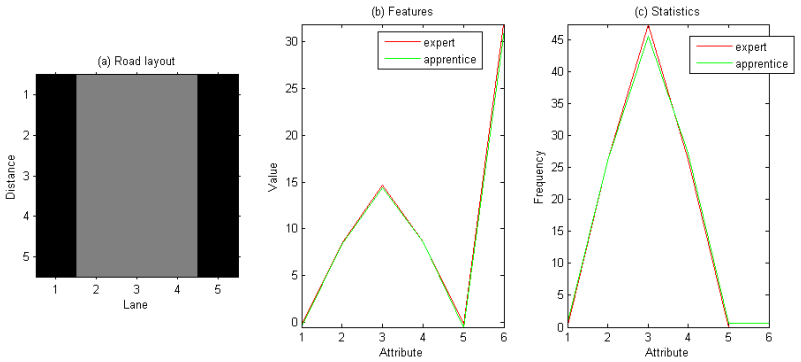

Figure 4.4: Primitive Policy 2

Figure 4.4 shows the learning of the road that uses the central three lanes. In this case, the expert completely avoided the outer two lanes, and spent the majority of its time in the central lane. This pattern was successfully learned by the apprentice, and again the number of collisions was kept to a minimum.

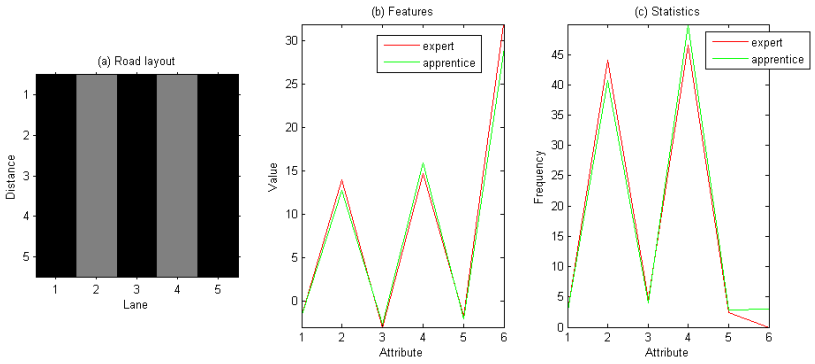

Figure 4.5: Primitive Policy 3

The split road case is shown in Figure 4.5. This is an interesting case, as it was not possible for even the expert to avoid both collisions and leaving the paved lanes as there were two separate single lanes. Thus, although the vast majority of time is spent in lanes 2 and 4, all of the other three lanes were marginally used. Collisions were thus completely avoided. The agent learned a very similar successfully from the expert, although the number of collisions was slightly higher in this more difficult case.

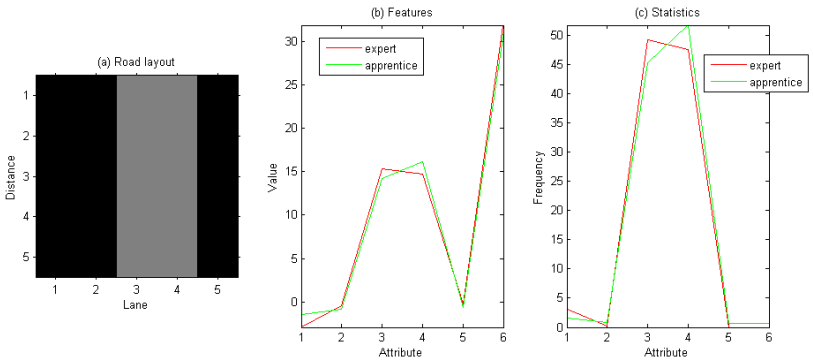

Figure 4.6: Primitive Policy 4

Figure 4.6 shows a two-lane road, consisting of the third and fourth lanes. The expert's policy has it using these lanes most of the time and interestingly the first lane occasionally as well. This is likely to be a result of the random start locations and car avoidance policy. Again the apprentice mimics the lane distribution of the expert very closely, although favours lane 4 slightly over the expert's choice to favour lane 3 slightly. As this would not affect the rewards, this is not a problem. Collisions are again maintained at a minimal level.

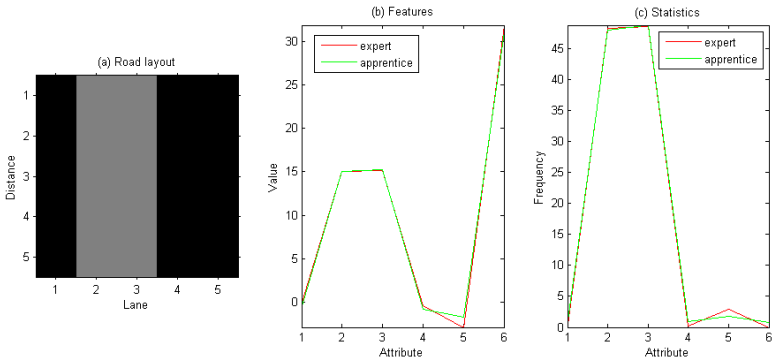

Figure 4.7: Primitive Policy 5

The mirror image of the previous experiment is shown in Figure 4.7 which features a two-lane road using the second and third lanes. In a similar manner to the way in which lane 1 was used in the previous case, lane 5 is used marginally by the expert in this case. However, the apprentice not only learns the use of lanes 2 and 3 well, but is also less reliant on lane 5 than the expert.

The road shown in Figure 4.8 consists of a single paved lane 3. The expert is able to avoid collisions with minimal deviations from this lane. The apprentice requires slightly more usage of the other lanes, yet still incurs a higher cost from an increased

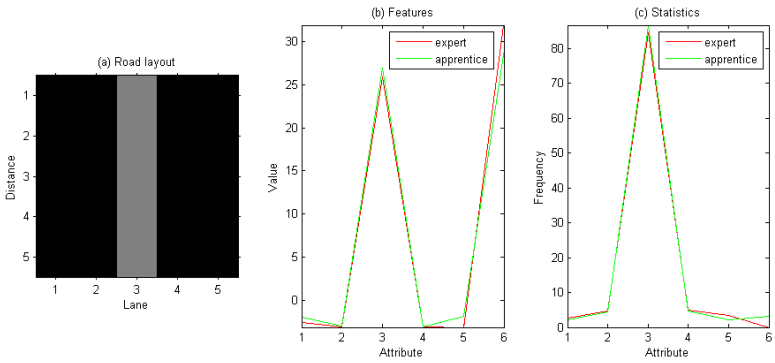

Figure 4.8: Primitive Policy 6

number of collisions. This case is difficult for the agent, as it has no room to maneuver around other cars while simultaneously staying on the paved road.

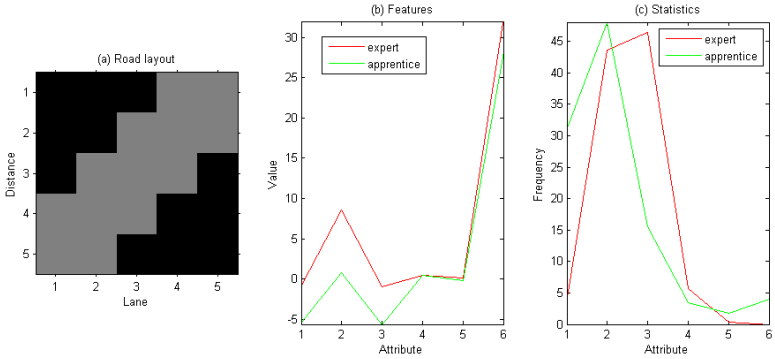

Figure 4.9: Primitive Policy 7

Figure 4.9 shows the first of the two diagonal roads in the basis. The lane use statistics show that the expert is focused mainly on lanes 2 and 3. This is somewhat less intuitive than the previous examples. Note that the starting location is selected at random from the five lanes. The ideal case scenario would be for the expert to be in lane 1 or 2 after five steps. If the expert starts in lanes 4 or 5, the policy should move the expert to the left into lane 2. Thus it is sufficient for this agent to spent most of the time in lanes 2 and 3 to maximise the reward. Similarly, starting in lanes 1 or 2 would require the agent to move into lanes 2 or 3 to connect with the path and proceed as before. Thus this lane distribution is logical.

Although the apprentice seems to exhibit dissimilar behaviour to the expert, it is actually performing similar route planning. Firstly, the learning agent spends a large percentage of its time in the second and third lanes, as did the expert. However, a large amount of time is spent in lane 1 as well. This is firstly because the agent receives the

same reward for ending in lane 1 or 2, and so does not have any incentive to leave lane 1 in the latter steps. Secondly, if this agent were to start in the leftmost lanes, it would choose to either stay there or only move as far as the second lane, as further movements to the right (in the presence of other cars) are not likely to increase the accumulated rewards. Again, being a more complicated case, the apprentice does not completely reduce collisions to zero.

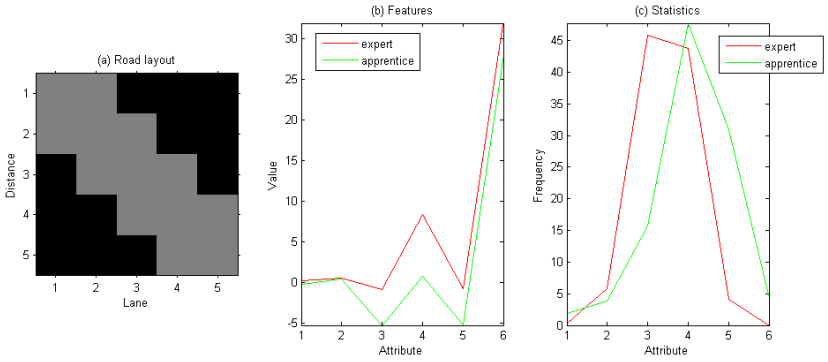

Figure 4.10: Primitive Policy 8

The example in Figure 4.10 shows the opposite diagonal road from the basis. The discussion on this case is largely similar to the previous one. The expert spends the majority of its time in lanes 3 and 4 as a result of reaching and following the diagonal path for five steps. The use of the other lanes is primarily a result of reaching these two lanes specifically. Again, the apprentice can be seen to use these two lanes much of the time, as well as the fifth lane. The reason for this is the same as the argument for it using the first lane in the previous example. Collisions are again not completely minimised but nevertheless impressive for an agent with no prior knowledge of the problem or task.

4.3.2 Performance of a Random Policy

The next experiment demonstrates the difference between the policy generated by the expert (and similarly by the learning agent) compared to a random policy which does not learn in any way.

Figure 4.11 provides this example of using a random policy where any of the three actions of either sticking in the current lane, or moving one lane to the left or the right are chosen with equal probabilities. The road used in this example is a three lane road, with an extra outer lane on either side acting as an unpaved road which should be

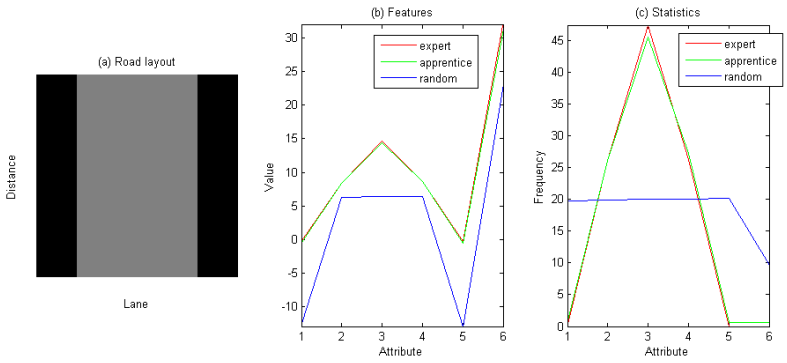

Figure 4.11: Results from a random policy

avoided if possible, as shown in Figure 4.11(a). This is the same example as depicted in Figure 4.4, and as such the results are repeated for the expert and the apprentice, where both achieve a distribution over the central three lanes peaked at the middle one, and very few collisions.

In contrast, the performance of the random policy is shown in blue. As can be seen in Figure 4.11(c), the five lanes are each used uniformly for 20 of the 100 steps each. Furthermore, the number of collisions is significantly higher than that of either of the other agents at about 10 in 100 steps. The poorer performance of this agent is clearly shown in Figure 4.11(b) as the rewards obtained for all six features are considerably lower than those of the other two agents, which are very similar to each other.

As can be seen from this example, the learning agent learns to perform approximately equivalently to the expert (although not always identically) and shows a considerable improvement over an agent with a random policy.

4.4 Composite Skill Acquisition Results

The terrains encountered and used for policy learning in Section 4.3 are all fairly simple in that they are all examples of configurations of straight line roads devoid of any complex features. However, for a complex task, one may expect the road to curve, weave or change direction. This is also what would be expected from a complicated configuration space for a robot, where there are regions which should be avoided corresponding to obstacles or impossible physical configurations.

This section demonstrates results of the second phase of the algorithm: synthesising primitive strategies into composite policies. These examples involve a game being played between the learning agent and nature. The actions available to the learning

agent are the policies learned in the previous section (or a subset of those policies in some cases). The actions available to nature are a choice of where and how cars appear on the road. The value of a game between these two players is the difference of the sums of their feature expectations on that terrain. That is, the sum of the total costs incurred by being in each lane at particular times (i.e. affected by the terrain layout) as well as the costs from any collisions. This provides a scalar value for the game as $\mathbf{G}(\psi, Q)$ for the learning agent using a strategy ψ and the environment using strategy Q.

The composite skill acquisition experiments were conducted by generating several new terrains and using these as input into the algorithm, together with the policies learned for the basis shown in Figure 4.2 in Section 4.3. For some of the experiments only a subset of this basis was used. The input roads consist of 5 lanes, but the length of the roads are longer (such as 10 or 15 car lengths). This allows for roads which vary in more interesting ways, and different weightings of the primitive policies are learned for each 5×5 subsection of the entire example. The agent thus traverses the full road length, but draws its actions from a different mixed policy after every 5 moves.

Each experiment was run for a total of 1,000 iterations. While this was not always enough to ensure complete convergence, the experiments were terminated at this point as a result of time constraints, and to ensure consistency between experiments. This length of time is however sufficient to show the asymptotic convergence of the game values to equilibria in each case.

A note on all the figures in this section. These figures all consist of four subfigures. Figure (a) shows the layout of the road presented in that experiment. As before, the paved section of roads allowing for easy travel are represented in gray, and the unpaved areas that are not to be used where possible appear black. Figure (b) shows the change in game values over the course of the experiment, with an increasing number of iterations up to 1,000. The blue line refers to the first region (5×5 patch) in the composite road, the green line the second region, and in some cases a red line for the third region. Figure (c) shows a distribution of the lane use statistics for the policies at the final ($1{,}000^{th}$) iteration. The five lanes are indicated by attributes 1 through 5 on the x-axis. Attribute 6 shows the number of collisions. As with Figure (b), the lines are coded as: blue for region 1, green for region 2 and red for region 3 (where applicable). Finally, Figure (d) shows the weights assigned to each element of the basis for constructing the final policy. Mathematically, these are the weights $\mathbf{w}$ such that if the composite policy $\tilde{\psi}$ is constructed from a combination of the basis policies Ψ, it is done according to

the formula $\tilde{\psi} = \mathbf{w} \cdot \Psi$. Each weight describes the extent to which the mixed policy is influenced by the corresponding component of the basis. The same colour conventions are used in this figure as in Figures (b) and (c).

4.4.1 Extracting Primitive Strategies

4.4.1.1 Simple Examples

The first experiment involves determining whether or not the algorithm is capable of fitting previously learned primitive strategies to exact replicas thereof in an unseen terrain. This situation is shown in Figure 4.12. As can be seen in Figure 4.12(a), this example is composed of three of the basis roads in succession: $R3 \rightarrow R4 \rightarrow R1$. One would expect the agent to change strategies for each of three regions, and draw primarily on the region from the basis matched to each sample region. Let the first region, being the first 5×5 patch, be referred to as region 1, the second as region 2, and the third as region 3.

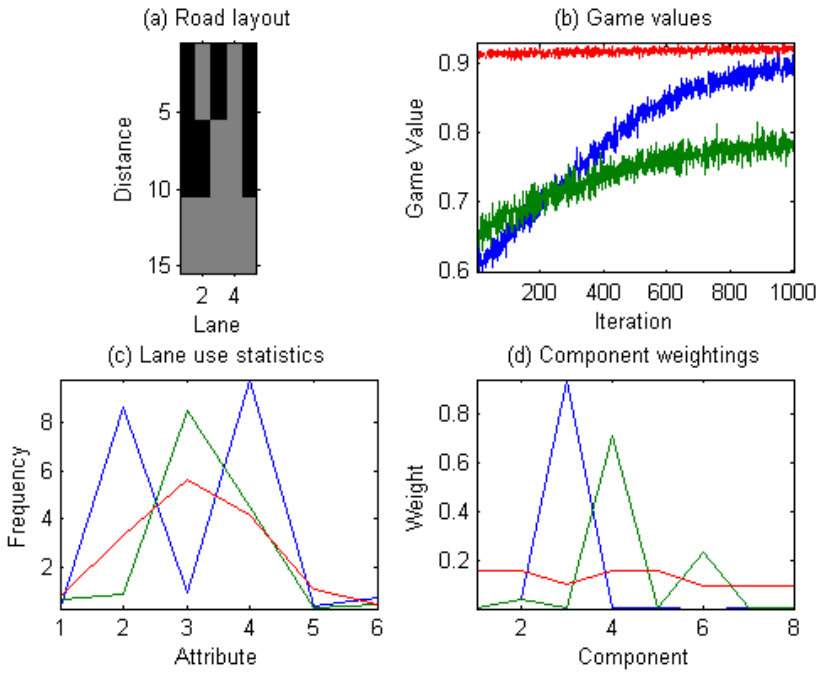

Figure 4.12: Extracting Primitive Strategies Experiment 1

The improvements in the game values for the three regions are shown in Figure 4.12(b) with the appropriate strategy weightings in Figure 4.12(d). In both cases, blue indicates results for region 1, green for region 2, and red for region 3. Marked improvements in the game value for regions 1 and 2 can be observed as the agent learns

to play Nash over the course of 1,000 iterations. The game value for region 3 is not seen to show any noticeable improvement. This is a result of the fact that this region is completely devoid of 'unpaved' road areas. As a result, the game value can not be improved by choice of lanes, but only by avoiding other cars.

The weights used for the mixed policies show that the agent is exhibiting the desired behaviour. Region 1 which has the split lane appearance of $R3$ is indeed solved by the agent drawing primarily on the policy generated by learning from the expert in phase one on $R3$. Region 2 is a two-lane stretch of road as is $R4$. The agent learns to construct its new strategy for this region from the strategies learned for $R4$ as well as $R6$, where $R6$ consists of a single lane only. These are the two roads in the basis which coincide with region 2. On the other hand, region 3 draws fairly uniformly from all eight basis roads. This is a result of the fact that region 3 has no distinguishing features, and although it appears exactly as $R1$, the strategies learned for any of the primitive roads would suffice with equivalent performance on this surface.

Figure 4.12(c) shows that with the final mixed policy all three regions maintained a low collisions score. Furthermore, region 1 used primarily lanes 2 and 4 (the paved lanes). The policy for region 2 relied mainly on the central lane, as advocated by policies $R4$ and $R6$, but relied far more on lane 4 than lane 2 as a result of $R4$. The lane distribution for region 3 peaks about the central lane, rather than being uniform, as a result of drawing almost uniformly from the basis which consists of several policies which place an emphasis on the central lanes.

This experiment thus shows a successful retrieval of exact primitive strategies from a composite road.

4.4.1.2 More Challenging Examples with Basis Restrictions

The next three experiments are all concerned with a road consisting of two regions, and composed as $R7 \rightarrow R8$. These experiments were an extension of the first, and attempted to determine the way in which the algorithm would recreate slightly more challenging examples, with various restrictions on the basis. These are considered more challenging as the basis consisted primarily of straight roads, and it would provide useful insights observing the response of the algorithm to the diagonal roads.

The first of these, shown in Figure 4.13, attempts to solve $R7 \rightarrow R8$ (shown in Figure 4.13(a)) using only the basis $\{R7, R8\}$. Figure 4.13(c) shows from attributes 1 to 5 that an approximately uniform distribution of the five lanes was used, slightly pitched at the centre, as would be expected from the basis strategies $R7$ and $R8$ which

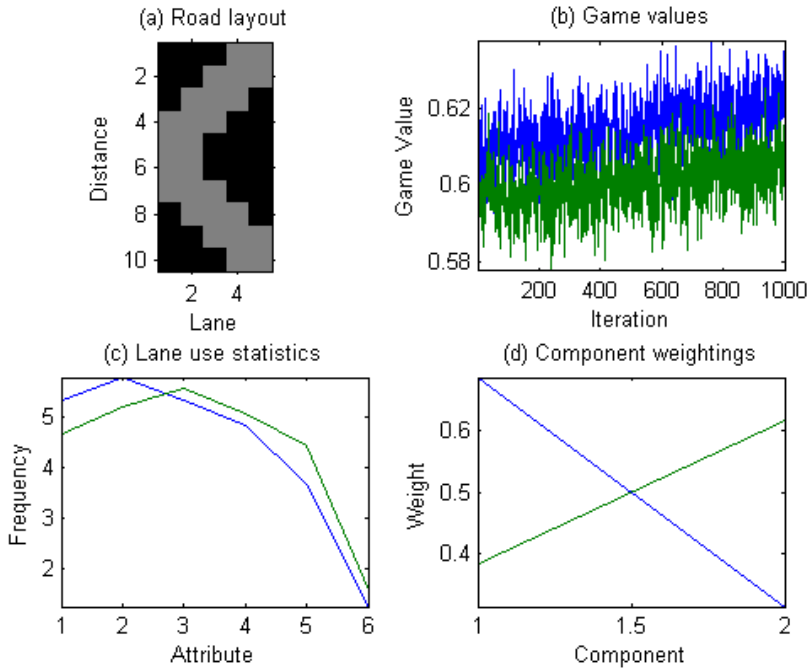

Figure 4.13: Extracting Primitive Strategies Experiment 2

use all the lanes at various points. The left side of Figure 4.13(d) corresponds to $R7$ and the right side to $R8$. This figure thus shows in blue that the policy for $R7$ was correctly used as the dominant policy for the first region, and the weights in green show that the policy $R8$ was considerably more heavily relied upon in constructing a policy for the second region. Interestingly, the game values do not improve considerably in either region after many iterations. Some possible reasons for this are discussed below.

The next experiment, seen in Figure 4.14, shows the results of the same road $R7 \rightarrow R8$ with the correct elements of the basis removed, and instead the basis consists of $\{R1, \ldots, R6\}$. This experiment was to establish whether or not the same results could be established from a different basis. As can be seen in Figure 4.14(d), the policies for both region 1 (blue) and region 2 (green) were constructed from those used for $R2$, $R4$ and $R5$. This seems a logical choice for the composition (from the options available) as all three basis roads are based on the central lane with some deviation to one or both sides. They also draw on the strategy for the split lane $R3$. Together these strategies allow the agent to navigate the diagonal paths. The fact that these basis components focus on the centre three lanes is reflected in the lane use statistics focusing primarily on these lanes in Figure 4.14(c).

Finally the agent was presented with the same road $R7 \rightarrow R8$ and allowed to draw

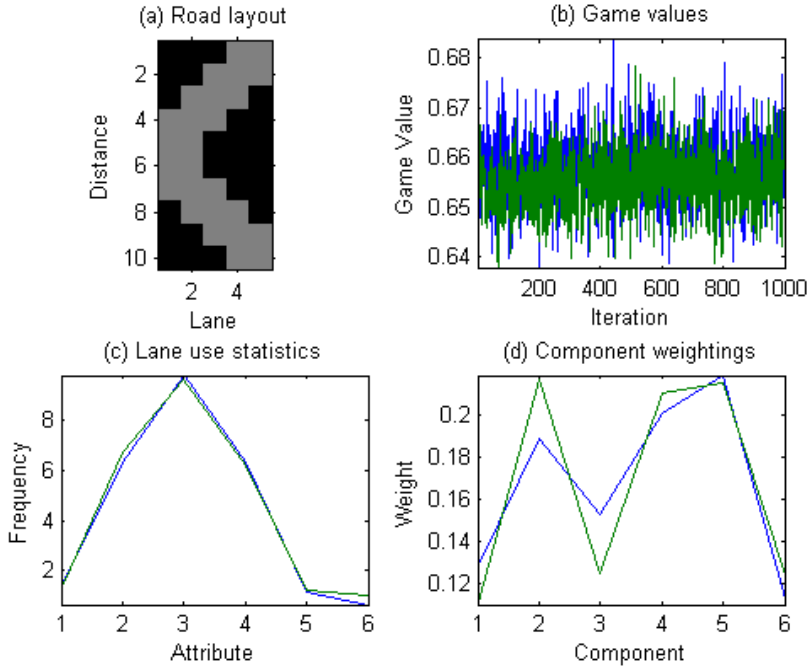

Figure 4.14: Extracting Primitive Strategies Experiment 3

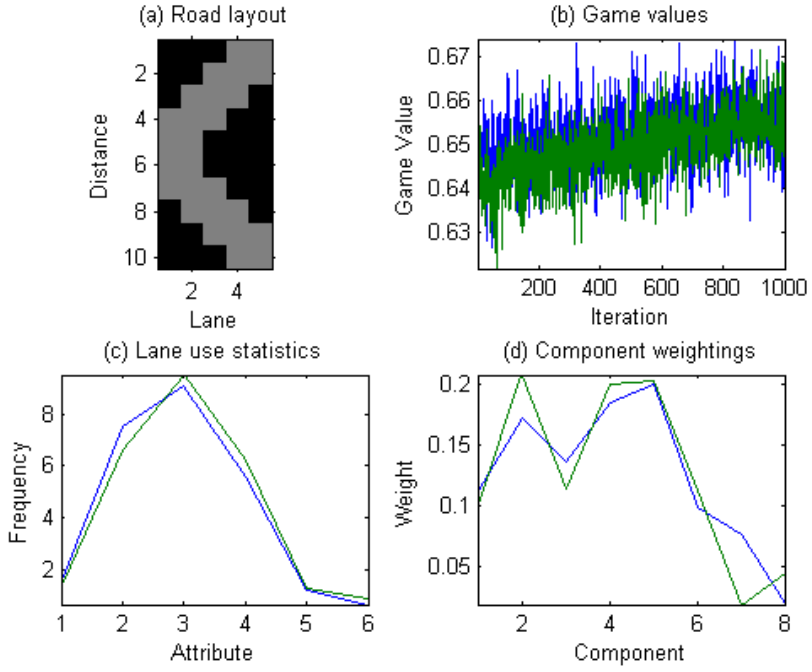

Figure 4.15: Extracting Primitive Strategies Experiment 4

from any of the strategies from $\{R1,\ldots,R8\}$. This was to observe the effects when the two candidates for suitable policies from the previous two experiments were combined. The results of this experiment were rather interesting. The agent reconstructed the

same relative weightings between the first six basis components as was seen in Figure 4.14(d), and between the last two components as seen in Figure 4.13(d). The surprise here was that the first six components were counter-intuitively emphasised much more in the mixed policy than the last two, which perfectly describe the example.

There are several reasons that could explain this. Firstly, the start location of the agent was selected randomly from the 5 lanes in each iteration. As a result, the agent was likely to begin on an unpaved section of road. It could thus take the agent several moves to reach the paved area, by which point any strategy would be useful. Furthermore, this was confounded by the fact that the regions are rather small (5×5). If the agent were to stick completely to the central lane, it would be driving on paved roads $6/10 = 60\%$ of the time. By invoking minimal lane changes, which could be provided by most of the strategies in the basis, the agent could be driving primarily on paved road. As a result, not only would this make $R7$ and $R8$ redundant, but the fact that they may not be useful for certain start locations may result in alternate strategies being favoured.

4.4.2 Strategies on New Roads

4.4.2.1 Unseen Roads

Having ascertained that the algorithm is capable of extracting exact primitive strategies from composite roads, the next set of tests involve observing performance of the algorithm on a composite road which has a different topology to all of those roads present in the basis. This is a test of the robustness of the algorithm, and whether or not it is capable of finding novel strategies for new terrains.

The first of these experiments involves a road which narrows from 5 lanes down to a single lane, and then widens again, as seen in Figure 4.16(a). An important observation to make is that in this experiment, region 1 and region 3 are very similar. Both are essentially $R1$ with a slight narrowing at either the beginning or the end. On the other hand, region 2 is completely different, with significant narrowing in the centre so as to resemble $R6$. This similarity between regions 1 and 3 is identified by the algorithm, which constructs very similar policies, as is seen by the blue and red plots in Figure 4.16(d). As was the case with the third region in the first experiment, these regions have policies which are composed almost uniformly from all the strategies in the basis. An exception to this is $R1$. This is because the areas to be avoided are minimal, and so different strategy selection would not make a considerable difference to performance,

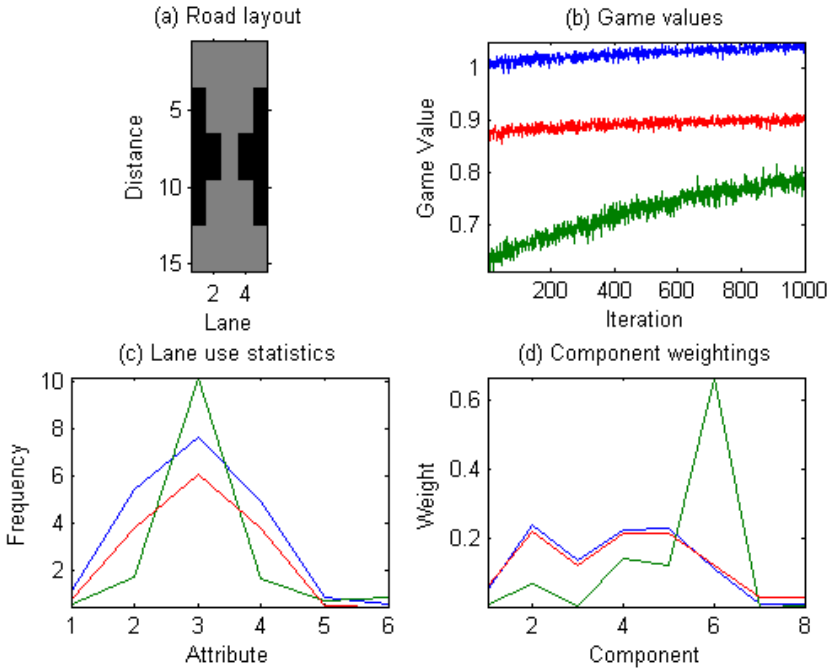

Figure 4.16: New Scenario Experiment 1

except for $R1$ which would use those areas. Also, for the same reasons, similarly to the first experiment, the game values for these two (blue and red) regions do not improve with more iterations. The two diagonal policies are not found to bear much resemblance to these regions either, as they use the corners and either draw the agent to or from undesirable areas.

There is more to be learned by the agent in region two, as the narrowing implies that the mixed policy has less freedom in choosing lanes. This learning is reflected in the fact that the game value for this region increases with additional training. The mixed policy for this region rejects the use of the strategies from the 'wider' road of $R1$ as it is insufficient for dealing with the narrowing. Instead, the 'two lane' policies from $R4$ and $R5$ are used, but the largest weighting is contributed by the single lane policy of $R6$. This is to be expected, as it is the case which most closely resembles the bottlenecked region.

The distributions of the lane usage in Figure 4.16(c) reflects the choice of policies. The blue and red lines corresponding to regions 1 and 3 respectively have a distribution primarily over the central three lanes, whilst the green distribution of region 2 spikes mainly on the one centre lane.

The next example is the inverse of the previous one. This, as seen in Figure 4.17(a),

consists of a single lane road which broadens into a five lane road before narrowing again to just a single lane. As was the case with the previous example, regions 1 and 3 are fairly similar, and this is recognised by the algorithm which devises a very similar mixture of policies for these two regions.

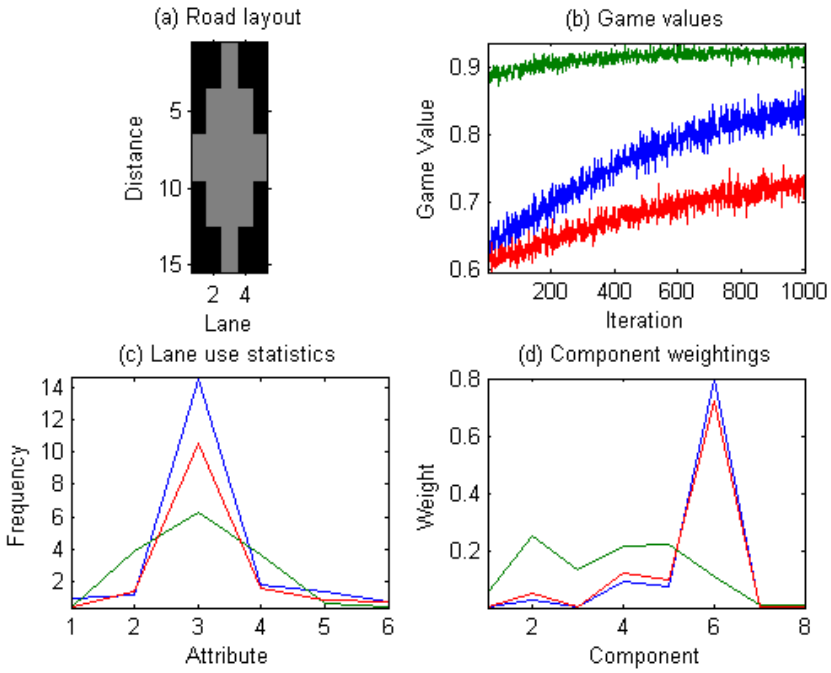

Figure 4.17: New Scenario Experiment 2

The fact that region 2 has unpaved areas at all four corners means there is still some, however limited, scope for learning. This is reflected in the fact that the green line in Figure 4.17(b) shows a very gradual increase in the game value. This increase is more marked in the blue and red graphs for regions 1 and 3 – the areas with considerable narrowing. As seen in Figure 4.17(d) the policies for these regions are drawn mainly from the single lane policy of $R6$. The policy for region 2 is composed almost uniformly from the first six components of the basis (excluding the diagonals). In fact, the weights assigned to the various policies in the basis for region 1 and 3 are very similar to those for region 2 in the previous example, and similarly those for regions 1 and 3 in the previous example are very similar to region 2 in this example.

The choice of policy obviously has a strong impact on the lane usage statistics. Regions 1 and 3 show, in blue and red respectively in Figure 4.17(c), large spikes about the central lane – the focus of the narrowing of the road. Conversely, the green of region 2 is more evenly distributed about the five lanes.

Again, it can be seen from the rightmost endpoint in Figure 4.17(c) (attribute 6) that the number of collisions in all three regions is low. They are however marginally higher for both regions 1 and 3 than for region 2. This is a result of the fact that the road narrows to a single lane, and so the policy is more reluctant to leave the road for an unpaved and unfavourable area.

Where the previous examples were symmetric about the central lane, the next example in Figure 4.18 has the road following a more skewed shape over the three regions.

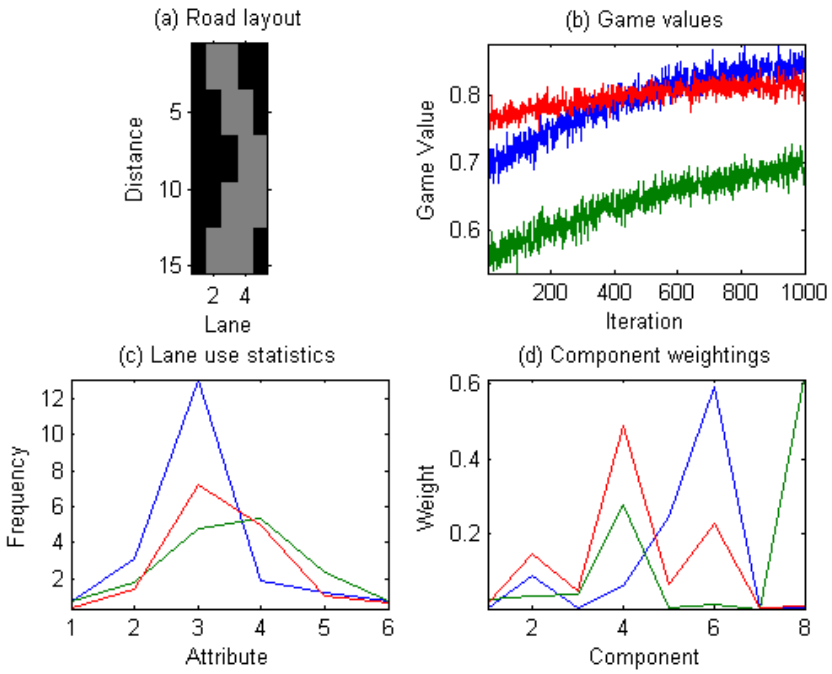

Figure 4.18: New Scenario Experiment 3

The first region in this example, seen in Figure 4.18(a) shows a slightly skewed two lane road. The basis policies used to form a strategy for this example are $R2$ which uses all three lanes present in the region, $R4$ and $R5$ which use two of the three lanes, but the second lane is used more than the fourth and so $R5$ is weighted more heavily, and the single lane $R6$ which is logical as the entire single lane is available in this region. The results of using this distribution is shown in Figure 4.18(c) as a spike about the central three lanes. As this distribution is learned, the game value increases considerably towards equilibrium in Figure 4.18(b).

Region 2 in this example has the valid area of road curved slightly and to the right of the centre. The green weights in Figure 4.18(d) show that the solution policy is

based largely on $R4$ and $R8$. This seems the best choice from the options available, as $R4$ is the two lane road to the right, and $R8$ is the diagonal road moving towards the right. Learning this distribution does move the game value towards equilibrium (green), but it is not as high as that for the first region, as a result of the basis not being particularly well suited to this example. However, it still manages to capture the topology well, and as shown in Figure 4.18(c), learns to use lanes 3 and 4 most of the time.

The third region has the road widening to three lanes and returning to the centre. As mentioned before, when the road is wider there is less of a steep increase in game value (red line), as the agent has less to gain by altering strategies. The main strategies used in this case are from $R2$, $R4$ and $R6$, corresponding to three lanes, two lanes to the right, and a single lane respectively. This gives the required distribution focusing on lanes 3 and 4.

4.4.2.2 Symmetric Cases of Diagonal Regions

The following four experiments are designed to ensure that the results being generated by the algorithm are consistent between examples. To show this, the roads in the experiment consist of the four permutations of having either a diagonal line from the top left to the bottom right, or from the top right to the bottom left, and then either the left or right side of the diagonal is paved, while the other half is unpaved. One would then expect the results arising from the four policies to have certain similarities and symmetries.

The first two cases are a right-to-left diagonal with the rightmost portion paved (Figure 4.19(a)) and a left-to-right diagonal with the leftmost portion paved (Figure 4.20(a)). These two instances correspond to a situation where the usable portion of the road starts off narrow as just a single lane to one side and then broadens to fill four of the five lanes. In both cases the game values show improvement towards equilibria, but the values for the second regions (green) are both considerably higher than those for the first regions (blue). This is a result of the fact that the first regions are far narrower, and allow the agent less room for maneuvering to avoid collisions. The agent subsequently does not perform as well in these regions.

The lane usages in Figure 4.19(c) shows a skewed distribution peaked around the fourth and fifth lanes for the first region (and to a lesser extent the third lane, for car avoidance), with a less skewed distribution on the third, fourth and fifth lanes for the second region – approximately as would be expected given the distribution of paved

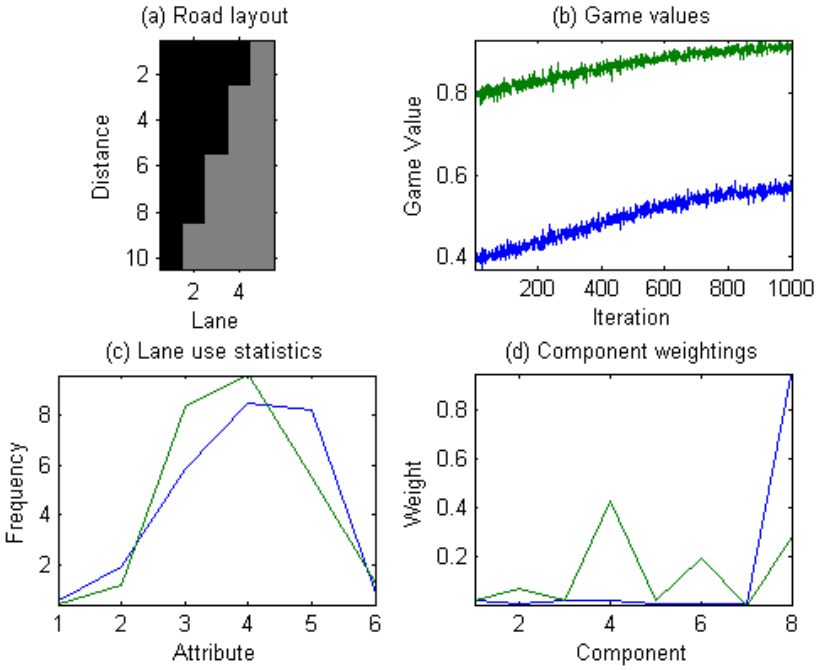

Figure 4.19: Diagonal Experiment 1

road seen in this example. Similarly, Figure 4.20(c) shows the skewed distribution peaked at lanes 1 and 2 in the first region and lanes 1, 2 and 3 in the second.

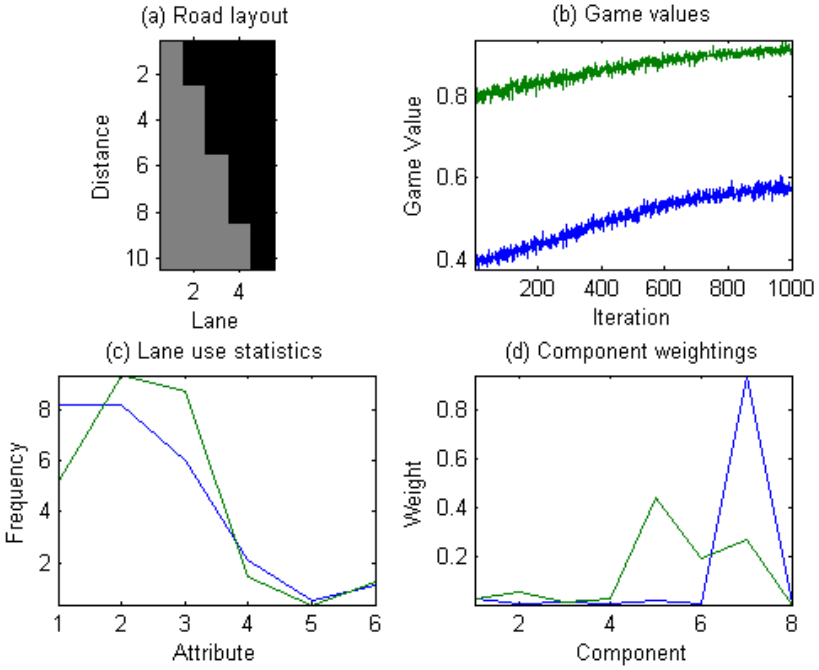

Figure 4.20: Diagonal Experiment 2

Figure 4.19(d) shows that the policy synthesised in that experiment is based predominantly on the one for $R8$. At a first glance, this is a rather surprising and counter-intuitive result. However, consider that the basis is rather limited and not complete, and the only other element of the basis which uses the fifth lane is $R1$ – which would incorrectly use the rest of the region as well, thereby incurring great costs. Also, the starting location of the agent is selected randomly. If it starts to the left, in the unpaved region, then using $R8$ would draw it towards the paved region on the right. As a result, the choice of this policy does make sense. Similarly, by the exact symmetric argument, the policy used for the first region in the experiment shown in Figure 4.20(d) is $R7$.

Now consider the policy for region 2 in Figure 4.19(d). The green weights provide a jigsaw pattern, showing a policy composition from $R2$, $R4$, $R6$ and $R8$. The use of the first three of these components is logical, in that they are the policies which deal with narrower regions with an emphasis to the right from $R4$. $R8$ has the same function as in the first region, in that it would draw the agent from the unpaved area on the left towards the right. Again, a similar policy can be seen for the second example in Figure 4.20(d). The symmetric basis policies, $R2$ and $R6$ are used here too, as well as the mirror images $R5$ and $R7$ of the other two.

The final two experiments consist of a left-to-right diagonal with the right side paved (Figure 4.21(a)) and a right-to-left diagonal with the left side paved (Figure 4.22(a)). These two cases correspond to a narrowing from four lanes to a single lane.

As with the previous pair of experiments, both regions in both these experiments showed an improvement in game value to equilibrium. This time, in both cases, the game value for the first region (shown in blue) is considerably better than that of the second region (shown in green). This is a result of the first region having a wider section of paved road, and therefore more maneuvering options to avoid collisions, and subsequently fewer collisions.

Figure 4.21(c) illustrates that lanes 3 and 4 are the lanes that are used the most by the mixed policy in the first region for the first of these experiments. In the second region, the fourth and fifth lanes are used more than any of the others, with the third lane receiving high usage as well. This is not ideal, but a result of the narrowing of the road, and thus a restricting of the options available for avoiding other cars. A symmetric case can be seen for the next experiment in Figure 4.22(c). This shows that in the first region lanes 2 and 3 are used primarily, whereas the distribution focuses on the first three lanes in the second region. The lane usage distributions thus match the paved road distributions fairly closely.

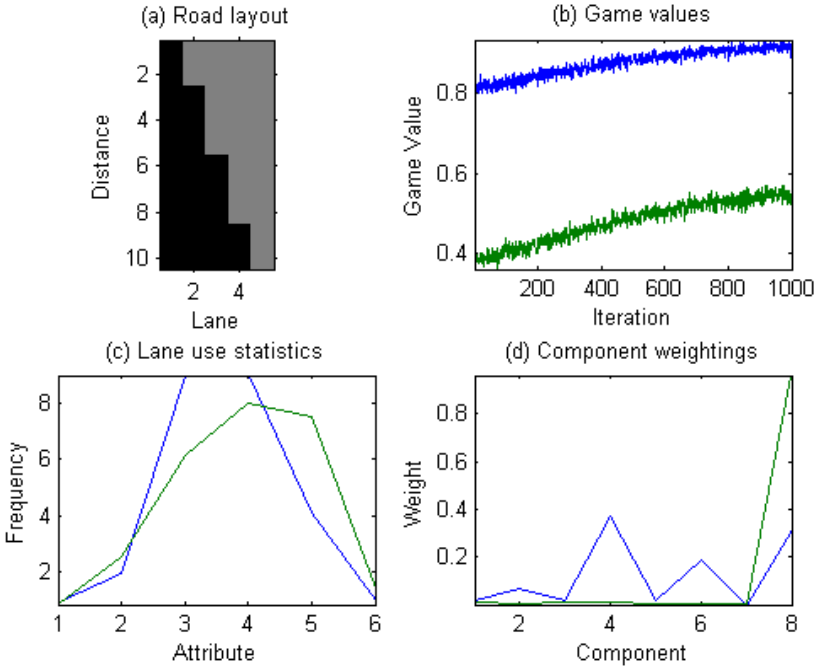

Figure 4.21: Diagonal Experiment 3

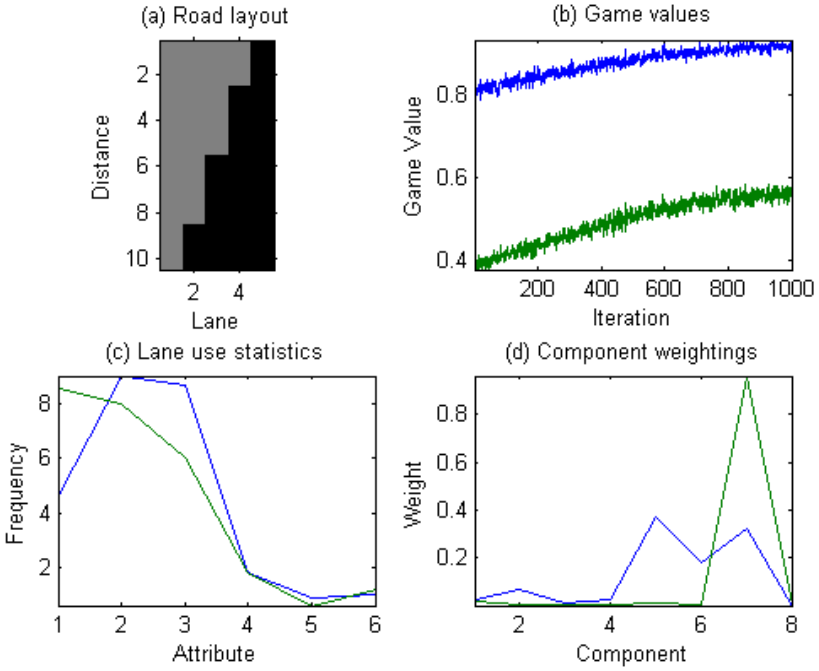

Figure 4.22: Diagonal Experiment 4

The weight distributions in Figure 4.21(d) are nearly identical to those of Figure 4.19(d), except with the regions reversed. Thus, for the first region, the weights favour the policies corresponding to $R2$, $R4$, $R6$ and $R8$. Again, the first three of these repre-

sent straight lane road layouts, with the bulk of the road in the centre or to the right, and not too wide. In this case $R8$ has a diagonal in the same direction as the road, and so is useful for drawing the agent from the left to the right. The weights for Figure 4.22(d) similarly use the symmetrical weights $R2$ and $R6$, as well as the mirror image basis components $R5$ and $R7$, and thus has a very similar policy to that in Figure 4.20(d) (with the regions reversed).

Now consider the weight distributions for the second regions. As was the case in Figures 4.19(d) and 4.20(d), the weights for region 2 in Figures 4.21(d) and 4.22(d) correspond to relying almost exclusively on $R8$ and $R7$ respectively. In Figure 4.21(d), the effect of the policy from $R8$ is to draw the agent from the left to the right, along the same diagonal as the edge of the paved road. The converse is true in Figure 4.22(d). In this case, the policy arising from $R7$ in the basis, draws the agent to the left, and away from the unpaved area.

The policies from the wider regions in Figures 4.19 and 4.21 are very similar, because these patches of road are similar. The same holds for the narrow regions in these examples, and consequently the lane use statistics. The same similarities exist between Figure 4.20 and Figure 4.22.

As can be seen, even with a potentially inadequate basis, the algorithm is able to construct policies which can be used for successfully navigating a variety of unseen terrains. A more sophisticated basis may have included roads at various other angles, but this set of experiments show that even a smaller basis can suffice for reconstructing complex strategies. The fact that mirror image policies were generated in pairs of examples which were mirror images of each other confirms that the strategies are consistent in the way they are constructed. Furthermore, in every experiment, the choices made by the algorithm for primary constituents of the mixed policies can be manually justified and confirmed to be logical choices. The algorithm could thus be said to be accomplishing its task of synthesising strategies successfully.

4.5 General Composite Policy Performance

The previous sections describe the results of the operation of the algorithm. Section 4.3 first provides examples of the algorithm learning from experts to create a basis of eight policies. Then Section 4.4 demonstrates the composition of these primitive policies into composite policies on particular complex examples of road. It remains to assess the performance of these synthesised strategies on the roads for which they

were generated, compared to the performance of the primitive strategies on the same roads.

There are two aspects of the performance of a policy which have been implicit in defining and improving the policies, and these are also useful for assessing the performance. When running a policy on a road, two averages were generated: the average number of times the policy deviated from paved road per time step, and the average number of collisions per time step.

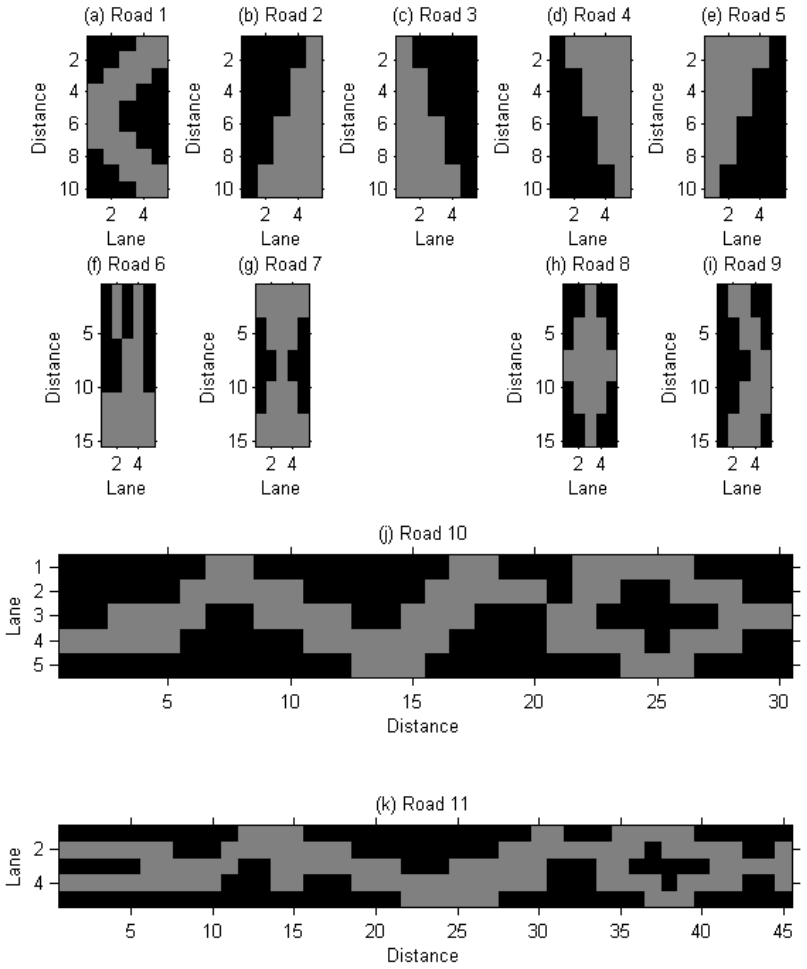

Figure 4.23: Experiment Road Set

For these experiments, a policy was trained on each of the eleven boards shown in Figure 4.23, all of which were five-lane roads. The first five of these roads, marked (a) to (e), are short templates, sized 10×5. The roads marked (f) to (i) are medium sized

at 15×5. The policies for all nine of these had already been generated in Section 4.4. Finally, an additional two policies were trained, one for each of road (j) and (k) – the long roads, of length 30 and 45 respectively.

Both the full contingent of eight primitive basis policies, and the composite policy trained specifically for that road were run repeatedly on each of these eleven roads for 4,500 time steps. The average number of times each policy deviated from the paved section of the road per time step (the accuracy of the policy), as well as the average number of collisions per time step were calculated and recorded for each of the nine policies on each of the eleven roads. The accuracies are presented in Table C.1 and the collisions in Table C.2, both in Appendix C.

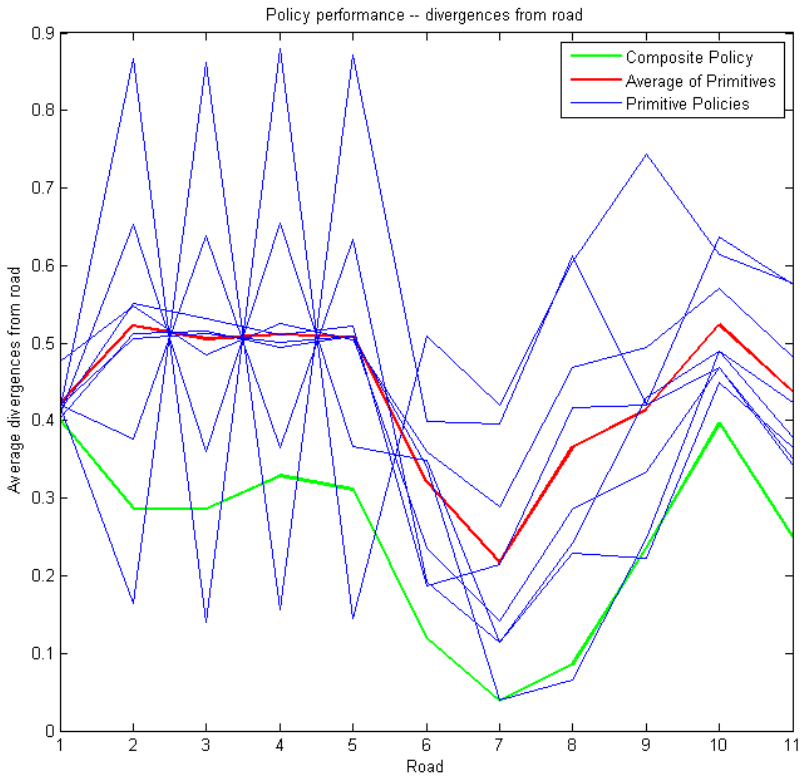

Figure 4.24: Performance – policy accuracy

Figure 4.24 plots the accuracy of the nine policies, as well as the average of the eight primitive policies. It is not necessary to differentiate between these primitive policies in this figure. As the figure records deviations from the paved section, the lower the value, the better. The x-axis, representing the different roads, has them sorted by increasing length.

As can be seen in this figure, the mixed composite policy acts as an approximate

loose lower bound on the performance of the primitive policies, and is considerably better in accuracy than the average of the primitives. This shows that the composite strategy is consistently among the candidates for most accurate policy.

First consider the performance of the policies on the first five roads: those of length 10. The first road has the accuracies of all policies tightly packed around 0.4125. As such, none of the basis policies are particularly accurate on this road. As this is the same road that was used in the experiments shown in Figures 4.13, 4.14 and 4.15, this would explain why the Strategy Composition Algorithm was unable to learn a strategy which was much of an improvement over the basis policies in those examples. The next four roads are those from the diagonal cases of Figures 4.19, 4.20, 4.21 and 4.22. In each of these, only one primitive policy can be seen to outperform the composite policy in Figure 4.24, and it performs considerably better than the rest of the policies.

The next set of four roads are the medium length roads which were used in experiments in Section 4.4, and are reproduced in Figure 4.23 (f) to (i). In the first of these, Road 6, the mixed policy performs better than any other policy. In the subsequent three experiments, Roads 7, 8 and 9, the mixed policy performs almost on a par with the best performing of the primitive strategies on those roads.

Now consider the two long roads, Roads 10 and 11. In both these cases, the composite policy outperforms all other policies, with the margin of accuracy between the composite and primitive policies slightly larger in the longer of the two roads.

The result of these observations, is that the shorter roads show the composite policy to perform better than the majority of, but not all, the primitive policies, the medium roads show a similar accuracy between the composite and primitive policies, and as the roads lengthen, the composite policies improve compared to the primitives and ultimately outperforms them. The reason for this is intuitive. The shorter a road template, the more likely it is that it will closely match one of the primitive roads. If one of these primitive policies were to perform better than the other primitives on both regions of a two-region road, it is likely that the composite policy would learn to be based predominantly on that policy, and may still be outperformed by that policy. However, with a longer road comes less of a chance of one of the basis policies performing well continuously, and that provides the composite policy with an opportunity to draw on elements of different components of the basis at different times for improved results.

The second important property which is used to measure the performance of policies is the average number of collisions with other cars per time step. These results for all nine policies on all eleven roads are displayed in Figure 4.25. This figure shows

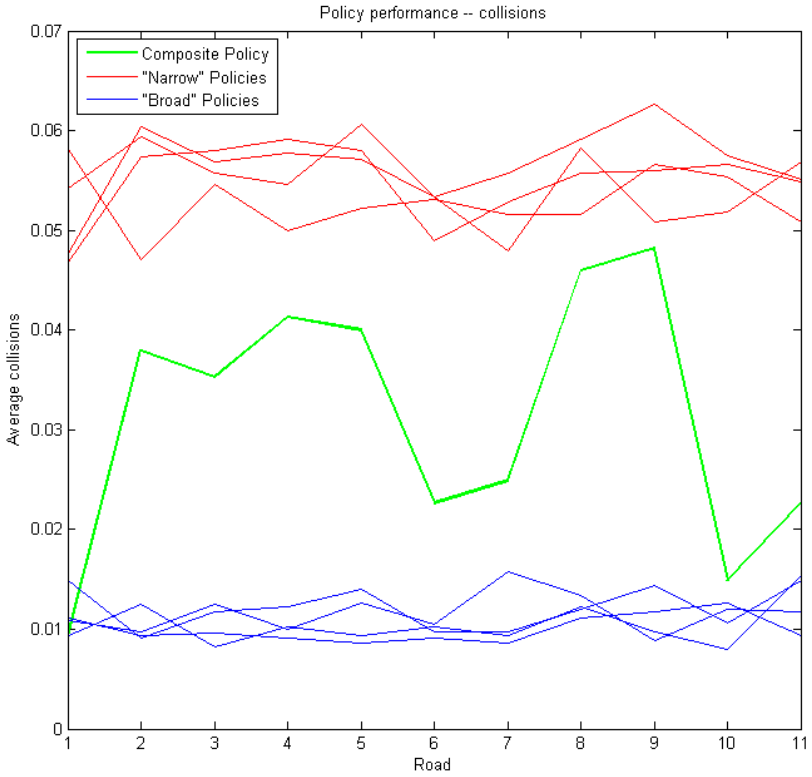

Figure 4.25: Performance – policy collisions

three ‘bands’ of results. The ‘lower’ band has values averaging around 0.0111 for all roads, and the ‘higher’ band averages 0.0548. The ‘central’ band corresponds to the mixed policy. Again, lower values are obviously better.

As it turns out, the primitive policies in the ‘lower’ band are policies corresponding to $R1$, $R2$, $R4$ and $R5$. These are the ‘broad’ policies, as they have two or more adjacent lanes that are paved. This means the policy is less likely to leave the road to avoid a collision and so, if collisions and deviating from the paved road are equally ‘bad’, these policies will tend to have less collisions as it is easier for them to avoid collisions without incurring terrain-based costs.

Conversely, the policies contributing to the ‘higher’ band are those corresponding to $R3$, $R6$, $R7$ and $R8$. These are the ‘narrow’ policies, because they do not consist of long segments of several adjacent lanes. As such, avoiding collisions is likely to involve leaving the paved areas of road and are penalised for doing so. Consequently, the number of collisions is higher for these policies.

The composite policy sits in a ‘central’ band between results of the ‘broad’ and the ‘narrow’ policies. It is clearly influenced almost equally by the two groups, with

the exact proportions thereof depending on the road. Although lower values would be preferred, the margin between the 'higher' and 'lower' band is small and the number of collisions are bounded by the primitive policies. Clearly there are trade-offs between accuracy and collisions in the creation of the policies, and is a consequence of the fact that neither property is indicated to the agent to be better or worse.

4.6 Conclusion

The algorithm described in Chapter 3 was implemented in MATLAB and a series of experiments were run to empirically demonstrate the performance of the algorithm, as documented in this chapter. The domain of the experiments used for this purpose was described in Section 4.2 as being a car navigating down a 5-lane road. The car has two different sets of conditions to endure. The first is that certain sections of the road are paved whilst others are not, yielding high and low rewards respectively. This corresponds to different surfaces or topologies of a manifold. The second is that there are other cars around which the agent should navigate, in order to avoid collisions. These cars are generated by nature and represent disturbance processes acting on a surface.

Section 4.3 was concerned with the first stage of the algorithm: learning primitive policies from an expert. A set of eight examples was used, and it was shown that the apprentice could achieve similar performance to the expert in terms of both lane distributions and collisions. The effectiveness of this learning policy was made even more apparent by the comparison to the performance of a random policy, which was significantly inferior to both the learning agent and the expert.

The second stage of the algorithm, dealing with the synthesis of primitive strategies was demonstrated in Section 4.4. The first experiments showed that the algorithm could recover the primitive strategies corresponding to roads in the basis, which were contained as sub-roads within a longer road. This was accomplished using various subsets of the basis. The algorithm then proceeded to generate novel strategies for unseen roads. The primary components of these mixed strategies were generally intuitive and logical choices. The algorithm also behaved as expected when presented with mirrored examples which correctly generated symmetrical results. Some difficulties were encountered, particularly with a unexpected use of the diagonal elements of the basis. This can be traced to the fact that the elements of the basis were in no way orthogonal and the basis was thus imperfect. However, this does demonstrate that the algorithm is

adaptable in that it will still find a solution to a presented problem.

The results in Section 4.5 show that the accuracy of the composite policies act as loose lower bounds on the accuracy of the primitive policies. Furthermore, as the length of the unseen road increases, the accuracy of the composite policy surpasses those of all the primitive policies, rendering it the superior policy. Additionally, although the composite policy can not boast the lowest number of collisions per unit time, it is tightly bounded by the same performance metric of the constituent primitive policies.

The following chapter summarises the important points contained in this document and provides insight into future work to be pursued by the author.

Chapter 5

Conclusion

Being able to learn new skills, and then adapt these skills to new situations, is one of the fundamental abilities of humans. This is thus a behaviour that would be useful to bestow on robots: increasing their versatility and scope for usefulness. Of the different ways to transfer the knowledge of different skills to a robotic agent, teaching by means of a demonstrating expert is a practical one. A human with some expert knowledge of a skill could demonstrate this skill in various settings to a robot, which would then learn to reproduce those behaviours and actions.

Apart from simply learning the trajectories required to perform a certain skill in particular scenarios, a robot should be able to apply that skill in new environments with unanticipated conditions. It is not practical to assume that an expert would be on hand to demonstrate the skill in every new scenario the robot encounters. To overcome this, the robot itself must be able to draw on previously acquired knowledge to establish a method for handling the new conditions.

The algorithm developed through the course of this research was inspired by the aforementioned ideals: to have a learning mechanism whereby a robot could learn a task in particular situations by observing an expert agent, and then compose this knowledge to deal with new conditions. The algorithm thus consists of two phases; one for learning primitive strategies and a second for composing them into composite strategies. This is inspired by the concept of learning from a curriculum and providing an agent with different tasks which can be hierarchically combined to handle more complex cases. This is all done whilst the agent is in the presence of adversarial processes in the form of the environment, which formulates the problem as a two-player game.

The method whereby this is accomplished is based on a combination of concepts from the fields of game theory and reinforcement learning. The reward function for

the MDP representing the state and action space of the skill is learned from the expert through a multiplicative weights technique, and then the optimal policy is approximated using Monte Carlo policy iteration. For the synthesis phase, the strategies are combined into a single mixed policy using a similar regret minimisation technique to adjust the contribution of each policy in the basis, depending on the success of that policy on the unseen scenario. The result is a single policy formed from the primitive policies, which best handles the new scenario and any random disturbances placed thereon.

Chapter 4 provides a sample domain for demonstrating the success of the algorithm. This takes the form of a traffic avoidance problem, where an agent is required to avoid collisions with other vehicles on a multi-lane road, and at the same time avoid leaving the paved areas of road for unpaved ones where possible. The topology of this road changes at various points along its length, in terms of the paved and unpaved areas. This is equivalent to a complicated topology on an abstract manifold in configuration space.

The results in this chapter demonstrate that in the first phase of the algorithm, the agent is able to learn optimal policies for navigating roads in the basis from the expert, such that both the apprentice and the expert use a similar distribution of the lanes throughout the course of one driving episode. The collisions made by the apprentice are also kept to a minimum.

In the second phase of the algorithm, the agent is able to successfully extract the appropriate primitive strategies from a composite one consisting of several primitive strategies connected together. It can also compose the primitive strategies successfully to navigate other completely unseen terrains and converge on an equilibrium which results in minimising both collisions and departures from the paved sections of the road. Even though the agent was not provided with a basis which may be considered complete in any sense, the agent is still able to draw on these strategies to increase its robustness and flexibility with respect to creating novel strategies for new situations.

It was also shown that the accuracy of the composite strategies are improvements over those of the primitive strategies on long roads, in terms of the number of deviations from the paved sections of the road. Furthermore, the composite policy accuracy is an approximate and loose lower bound on the accuracy of its constituent policies. The number of collisions of the composite policy is not as low as for some of the primitive policies, but this score remains bounded both above and below by the primitive policies. Considering these facts, the composite policy is an improvement over the

primitive policies for the unknown scenarios.

This work is merely a single step towards a complete system which can learn a skill in different settings from an expert, then synthesise these to handle any new scenario to ensure total robustness of the newly acquired skill, and finally draw on previously learned skills to ease the learning of different skills.

Within this framework, there is thus much scope for future work. The first challenge that will be considered in extending the work presented within this document will be to examine methods for reducing the state space dimensionality, both by means of techniques such as PCA and with different representations. This is because large tasks become exponential in the number of states required for their representation, which makes it infeasible to compute optimal policies. One approach to this may be to consider model predictive control (MPC) and, instead of learning weights for the primitive strategies at discrete patches of the problem domain, learning the weights as a continuous function over the state space.

Other extensions to this work would be to consider that different instances of the skill may have varying state and action spaces. An agent may start the learning process in a very simple setting with few actions available to it, but once it masters this setting, the number of available actions increase. Furthermore, in the synthesis phase, it may be worth investigating whether or not an expert could be employed in some situations to guide the mixing of these strategies, as an intermediate step in the hierarchy.

One last extension to consider would be a method for amalgamating all the acquired information. The system in its current form is merely a learning system. If this was implemented on a robot, that robot would need to be able to recognise the scenarios on which it has trained so that it knows when it would be appropriate to invoke the correct strategy to complete the required task. In other words, the system should be able to easily recognise a problem it has already solved.

The research described in this document presents a novel approach to learning from demonstrating by posing the learning problem as a game, and then drawing on curriculum learning to learn a structured strategy which is robust to changes in conditions and adversarial environmental disturbances. The result is an algorithm which is capable of learning a new skill, and adapting it from the ‘classroom examples’ taught by the expert to situations that are encountered in the unsupervised operation of the agent.

Bibliography

[Abbeel and Ng 2004] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings of the International Conference on Machine Learning*, 2004.

[Atkeson and Schaal 1997] Christopher Atkeson and Stefan Schaal. Robot learning from demonstration. *International Conference on Machine Learning*, pages 11–73, 1997.

[Bakker and Kuniyoshi 1996] Paul Bakker and Yasuo Kuniyoshi. Robot see, robot do: An overview of robot imitation. *AISB96 Workshop on Learning in Robots and Animals*, pages 3–11, 1996.

[Basar and Olsder 1999] T. Basar and G.J. Olsder. *Dynamic Noncooperative Game Theory*. Society for Industrial and Applied Mathematics, 2nd edition, 1999.

[Bengio *et al.* 2009] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[Billard *et al.* 2004] A. Billard, Y. Epars, S. Calinon, S. Schaal, and G. Cheng. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47:69–77, 2004.

[Breazeal and Scassellati 2002] C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in Cognitive Sciences*, 6(11):481–487, November 2002.

[Choset *et al.* 2005] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms and Implementations*. MIT Press, 1st edition, 2005.

[Demiris and Hayes 1996] John Demiris and Gillian Hayes. Imitative learning mechanisms in robots and humans. *Proceedings of the European Workshop on Learning Robotics*, 1996.

[Elman 1993] J.L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[Erez and Smart 2008] T. Erez and W.D. Smart. What does shaping mean for computational reinforcement learning? *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pages 215–219, 2008.

[Foster and Young 2006] D.P. Foster and H.P. Young. Regret testing: learning to play Nash equilibrium without knowing you have an opponent. *Theoretical Economics*, 1:341–367, 2006.

[Freund and Schapire 1999] Y. Freund and R.E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[Harmon and Harmon 1999] M.E. Harmon and S.S. Harmon. Reinforcement learning: A tutorial. 1999.

[Inamura *et al.* 2004] T. Inamura, Y. Nakamura, I. Toshima, and H. Tanie. Embodied symbol emergence based on mimesis theory. *The International Journal of Robotics Research*, 23(4-5):363–377, 2004.

[Lambson and Probst 2004] V.E. Lambson and D.A. Probst. Learning by matching patterns. *Games and Economic Behaviour*, 46:398–409, 2004.

[Nakanishi *et al.* 2004] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47:79–91, 2004.

[Nisan *et al.* 2007] N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 1st edition, 2007.

[Schaal *et al.* 2004] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. *International Symposium on Robotics Research*, 2004.

[Shamma and Arslan 2005] J.S. Shamma and G. Arslan. Dynamic fictitious play, dynamic gradient play, and distributed convergence to Nash equilibria. *IEEE Transactions on Automatic Control*, 50(3):312–326, March 2005.

[Singh 1992] S.P. Singh. Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[Sutton and Barto 1998] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1st edition, 1998.

[Syed and Schapire 2008] U. Syed and R.E. Schapire. A game-theoretic approach to apprenticeship learning. *Advances in Neural Information Processing Systems*, 2008.

[Syed *et al.* 2008] U. Syed, M. Bowling, and R.E. Schapire. Apprenticeship learning using linear programming. *Proceedings of the 25th International Conference on Machine Learning*, 307:1032–1039, 2008.

[Young 2009] H.P. Young. Learning by trial and error. *Games and Economic Behaviour*, 65:626–643, 2009.

Appendix A

Code for Algorithm Phase 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main Primitive Learning Function                                      %
% Input: terrain reward function                                        %
% Output: policy, weights, features (both agents), stats                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [policy, weight, mu, mu_e, stats] =
            learnprimitive(terrain)

% Initialise parameters
params = setParams(terrain);

gamma = params(1);
T = params(2);
M = params(3);
MAXSTATES = params(4);
MAXACT = params(5);
FEAT_SIZE =
    length(features(getStartState(params), params));
stats_e = zeros(1, FEAT_SIZE);
stats = stats_e;

% Simulate with the expert to create features
mu_e_temp = [];
for m = 1:(5*M)
    s0 = getStartState(params);
    [mu_t, state_t, act_t, stats_t, params] =
         simulate(s0, params, 0);
```

```
    mu_e_temp = [mu_e_temp; mu_t];
    stats_e = stats_e + stats_t;
end
mu_e = sum(mu_e_temp)/(5*M);
stats_e = stats_e / (5*M);

% Initialise apprentice policy
policy = ones(MAXSTATES, MAXACT)/MAXACT;
Q = zeros(MAXSTATES, MAXACT);
Rets = zeros(MAXSTATES, MAXACT, 2);

% MWAL algorithm
k = length(mu_e);
beta = 1/(1+sqrt(2*log(k)/T));
W = ones(1, k);
for t = 1:T
    weight(t,:) = W(t,:)/sum(W(t,:));
    % Update policy P(t) - Monte Carlo RL
    [policy, Q, Rets] =
        MC_RL(weight(t,:), policy, Q, Rets, params);
    mu_temp = [];
    s_t = zeros(1, FEAT_SIZE);
    % Estimate features mu(policy)
    for m = 1:M
        s0 = getStartState(params);
        [mu_t, state_t, act_t, stats_t, params] =
            simulate(s0, params, policy);
        mu_temp = [mu_temp; mu_t];
        s_t = s_t + stats_t;
    end
    stats = stats + s_t;
    mu(t,:) = sum(mu_temp)/M;
    % Update weights
    W(t+1,:) =
        W(t,:).*exp(log(beta)*game([1:k],mu(t,:),mu_e,gamma));
end
stats = stats/(T*M);
stats = [stats_e; stats];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Function to Return the Game Value                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function G = game(i, mu, mu_e, gamma)
G = ((1-gamma)*(mu(i) - mu_e(i))+2)/4;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Monte Carlo RL to optimise policy                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P, Q, Rets] = MC_RL(w, P, Q, Rets, params)

M = params(3);
MAXSTATES = params(4);
MAXACT = params(5);

for m = 1:M
    s0 = getStartState(params);
    % Generate episode
    [mu, state, actions, temp, params] =
            simulate(s0,params,P);
    for j = 1:length(actions)
        s = state(j);
        a = actions(j);
        R = reward(state(j+1), w, params);
        % Record returns
        Rets(s,a,1) = Rets(s,a,1) + R;
        Rets(s,a,2) = Rets(s,a,2) + 1;
        % Update Q values
        Q(s,a) = Q(s,a) + 1/Rets(s,a,2)*(R - Q(s,a));
    end

    e = 0.01;
    for q = 1:MAXSTATES
        % Select best actions
        [ig, best] = max(Q(q, :));
        if sum(Q(q,:) == ig) > 1
            ind = find(Q(q,:) == ig);
            r = ceil(rand*length(ind));
            best = ind(r);
        end
        % Soft policy
        P(q, :) = e/MAXACT;
```

```
112         P(q, best) = 1 - e + e/MAXACT;
113     end
114 end
115
116 end
117
118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119 % Function to Return the Reward for a State                               %
120 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121 function R = reward(state, weight, params)
122 R = weight * features(state, params)';
123 end
124
125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
126 % Function to Initialise Parameters for Road Example                      %
127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
128 function params = setParams(terrain)
129 gamma = 0.99;
130 NUMLANES = size(terrain,1);
131 DISTROAD = size(terrain,2);
132 T = 100;
133 M = 100;
134 MAXSTATES =
135       prod([NUMLANES, DISTROAD, (DISTROAD+1)^NUMLANES]);
136 MAXACT = 3;
137 params = [gamma T M MAXSTATES MAXACT NUMLANES DISTROAD
138     reshape(terrain', 1, numel(terrain))];
139 end
140
141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
142 % Function to Select Random Start Lane                                    %
143 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
144 function s0 = getStartState(params)
145 NUMLANES = params(6);
146 s0 = ceil(rand*(NUMLANES));
147 end
148
149 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
150 % Function to Simulate Road Example                                       %
151 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
152 function [mu, state, actions, stats, params] =
153                 simulate(s0, params, policy)
```

```
154
155 gamma = params(1);
156 T = params(2);
157 NUMLANES = params(6);
158 DISTROAD = params(7);
159
160 % Initialise road and cars
161 roadtemp = zeros(DISTROAD, NUMLANES);
162 road = roadtemp;
163 lane = s0;
164 dist = 1;
165 cars = zeros(1, NUMLANES);
166 tcars = num2cell(cars+1);
167 tslane = ones(1, NUMLANES)*(DISTROAD+1);
168 state = sub2ind([NUMLANES,DISTROAD,tslane],lane,dist,tcars{:});
169 actions = [];
170 stats = zeros(1, NUMLANES+1);
171
172 for i = 1:T
173     % Move along road
174     road = [zeros(1, NUMLANES); road(1:DISTROAD-1, :)];
175     % Generate new cars
176     o_cars = cars;
177     if (rand > 0.5)
178         k = ceil(rand*NUMLANES);
179         road(1, k) = 1;
180     end
181     [ig, cars] = max(flipud(road));
182     cars = DISTROAD + 1 - cars;
183     cars(ig == 0) = 0;
184     o_lane = lane;
185
186     if length(policy) == 1
187     % Expert policy
188         if DISTROAD - cars(lane) == 0
189             if lane == 1
190                 if DISTROAD - cars(2) ≥ 1
191                     lane = 2;
192                 else
193                     lane = 1;
194                 end
195             elseif lane == NUMLANES
```

```
196                 if DISTROAD - cars(NUMLANES-1) ≥ 1
197                     lane = NUMLANES-1;
198                 else
199                     lane = NUMLANES;
200                 end
201             else
202                 r = ceil(2*rand);
203                 k = (-1)^r;
204                 if DISTROAD - cars(lane+k) ≥ 1
205                     lane = lane+k;
206                 else
207                     lane = lane-k;
208                 end
209             end
210         end
211     else
212     % Apprentice policy
213         tcars = num2cell(o_cars+1);
214         tslane = ones(1, NUMLANES)*(DISTROAD+1);
215         st =
216           sub2ind([NUMLANES,DISTROAD,tslane],
217           lane,dist,tcars{:});
218         r = rand;
219         act = 0;
220         while r > 0
221             act = act + 1;
222             r = r - policy(st, act);
223         end
224         lane = lane + act - 2;
225         lane = min(max(lane, 1), NUMLANES);
226     end
227
228     % Record state
229     tcars = num2cell(cars+1);
230     tslane = ones(1, NUMLANES)*(DISTROAD+1);
231     st =
232         sub2ind([NUMLANES,DISTROAD,tslane],lane,dist,tcars{:});
233     state = [state; st];
234     actions = [actions; (lane-o_lane)+2];
235     dist = mod(dist, DISTROAD) + 1;
236
237     stats(lane) = stats(lane) + 1;
```

```
238     if cars(lane) == DISTROAD
239         stats(end) = stats(end) + 1;
240     end
241 end
242
243 % Compute feature vector
244 mu = zeros(size(features(state(1), params)));
245 for t = 0:length(state)-1
246     mu = mu + gamma^t*features(state(t+1), params);
247 end
248
249 end
250
251
252 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
253 % Function to Compute Feature Vector for Road Example                    %
254 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
255 function ft = features(state, params)
256
257 NUMLANES = params(6);
258 DISTROAD = params(7);
259 terrain = params(8:end);
260 terrain = reshape(terrain, DISTROAD, NUMLANES)';
261
262 tslane = ones(1, NUMLANES)*(DISTROAD+1);
263 cpos = num2cell(zeros(1,NUMLANES));
264 [lane,dist,cpos{:}] =
265             ind2sub([NUMLANES,DISTROAD,tslane],state);
266 % Reward lane use
267 ft = [zeros(1, NUMLANES) 0.5];
268 ft(lane) = terrain(dist, lane);
269 % Penalise collisions
270 if cpos{lane} == DISTROAD+1
271     ft(NUMLANES+1) = -1;
272 end
273
274 end
```

Appendix B

Code for Algorithm Phase 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main Composite Learning Function                                      %
% Input: terrain reward function                                        %
% Output: policy, game value, stats, weights                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [policy, res, stats, weights] =
                compose_policies(terrain)
% Load basis
load('basepolicies.mat');
[POLICIES, STATES, ACTIONS] = size(basis)

% Initialise parameters
params = setParams(terrain);
boards = params(end-1)/params(7);

T = 300;
k = POLICIES;
beta = 1/(1+sqrt(2*log(k)/T));
% Initialise weights
W = ones(boards, k);
for i = 1:boards
    weights(i,:) = W(i,:)/sum(W(i,:));
end
res = [];
stats = [];
for t = 1:T
    w_o = weights;
```

```
    % Update policy
    for j = 1:boards
        policy(j,:,:) = weights(j,1) * basis(1,:,:);
        for i = 2:POLICIES
            temp(:,:) = weights(j,i) * basis(i,:,:);
            policy(j,:,:) =
               policy(j,:,:) + reshape(temp, [1 size(temp)]);
        end
    end

    % Test policy performance
    [rv, rs] = test(policy, params);
    res = [res rv];
    stats = [stats; rs];

    % Determine game values
    gval = fgame(basis, policy, params)';
    tempv = beta.^gval;

    % Update weights
    for i = 1:boards
        W(i,:) = weights(i,:) .* tempv(i,:);
        weights(i,:) = W(i,:)/sum(W(i,:));
    end
end
res = reshape(res, [boards, T])';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to Test Current Policy Performance                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [V1, stats_ave] = test(policy, params)

P = policy;

M = 100;
for m = 1:M
    % Run simulator with current policy
    [val, stats] = simulate(getStartState(params), params, P);
    if m == 1
        stats_ave = stats;
        V1 = val;
```

```
    else
        stats_ave = stats_ave + stats;
        V1 = V1 + val;
    end
end
% Estimate game value and statistics
V1 = V1/M;
stats_ave = stats_ave/M;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to Determine Game Value                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function V = fgame(policy1, policy2, params)

% Play each basis policy
M = 100;
for i = 1:size(policy1,1)
    P(:,:) = policy1(i,:,:);
    for m = 1:M
        val = simulate(getStartState(params), params, P);
        if m == 1
            V1(i,:) = val';
        else
            V1(i,:) = V1(i,:) + val;
        end
    end
end
V1 = V1/M;

% Play mixed policy
P = policy2;
M = 100;
for m = 1:M
    val = simulate(getStartState(params), params, P);
    if m == 1
        V2 = val;
    else
        V2 = V2 + val;
    end
end
```

```
112 V2 = V2/M;
113
114 % Determine game value
115 V = ((repmat(V2, (size(V1)./size(V2)))-V1)+2)/4;
116 end
117
118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119 % Function to Initialise Parameters for Road Example           %
120 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121 function params = setParams(terrain)
122 gamma = 0.99;
123 NUMLANES = 5;
124 DISTROAD = 5;
125 T = 50;
126 M = 50;
127 MAXSTATES = prod([NUMLANES, DISTROAD, (DISTROAD+1)^NUMLANES]);
128 MAXACT = 3;
129 [FULL_L, FULL_W] = size(terrain);
130 params = [gamma T M MAXSTATES MAXACT NUMLANES DISTROAD
131       reshape(terrain', 1, numel(terrain)) FULL_L FULL_W];
132 end
133
134
135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
136 % Function to Select Random Start Lane                         %
137 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
138 function s0 = getStartState(params)
139 FULL_W = params(end);
140 s0 = ceil(rand*(FULL_W));
141 end
142
143 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
144 % Function to Simulate Road Example                            %
145 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
146 function [mu, state, actions, stats, params] =
147                 simulate(s0, params, policy)
148
149 gamma = params(1);
150 T = params(2);
151 NUMLANES = params(6);
152 DISTROAD = params(7);
153 FULL_L = params(end-1);
```

```
154
155 % Initialise road and cars
156 roadtemp = zeros(DISTROAD, NUMLANES);
157 road = roadtemp;
158 lane = s0;
159 dist = 1;
160 fpos = 1;
161 cars = zeros(1, NUMLANES);
162 tcars = num2cell(cars+1);
163 tslane = ones(1, NUMLANES)*(DISTROAD+1);
164 state = sub2ind([NUMLANES, DISTROAD, tslane],
165               lane, dist, tcars{:});
166 actions = [];
167 stats = zeros(FULL_L/DISTROAD, NUMLANES+1);
168
169 for i = 1:T
170     % Move along road
171     road = [zeros(1, NUMLANES); road(1:DISTROAD-1, :)];
172     % Generate new cars
173     o_cars = cars;
174     if (rand > 0.5)
175         k = ceil(rand*NUMLANES);
176         road(1, k) = 1;
177     end
178     [ig, cars] = max(flipud(road));
179     cars = DISTROAD + 1 - cars;
180     cars(ig == 0) = 0;
181     o_lane = lane;
182
183     % Calculate current region of terrain
184     frame = ceil(fpos / DISTROAD);
185
186     % Apprentice policy
187     tcars = num2cell(o_cars+1);
188     tslane = ones(1, NUMLANES)*(DISTROAD+1);
189     st = sub2ind([NUMLANES, DISTROAD, tslane],
190                  lane, dist, tcars{:});
191     r = rand;
192     act = 0;
193     while r > 0
194         act = act + 1;
195         if ndims(policy) == 3
```

```
196             r = r - policy(frame, st, act);
197         else
198             r = r - policy(st, act);
199         end
200     end
201     lane = lane + act - 2;
202     lane = min(max(lane, 1), NUMLANES);
203
204     % Record state
205     tcars = num2cell(cars+1);
206     tslane = ones(1, NUMLANES)*(DISTROAD+1);
207     st = sub2ind([NUMLANES, DISTROAD, tslane],
208                  lane, dist, tcars{:});
209     state = [state; st];
210     actions = [actions; (lane-o_lane)+2];
211     dist = mod(dist, DISTROAD) + 1;
212     fpos = mod(fpos, FULL_L) + 1;
213
214     stats(frame, lane) = stats(frame, lane) + 1;
215     if cars(lane) == DISTROAD
216         stats(frame, end) = stats(frame, end) + 1;
217     end
218 end
219
220 % Compute feature vector
221 mu = zeros(size(features(state(1), params, 0)));
222 for t = 0:length(state)-1
223     mu = mu + gamma^(mod(t, DISTROAD))
224                 *features(state(t+1), params, t);
225 end
226
227 % Determine value used to calculate game value
228 val = (sum(mu')+(T)/(FULL_L/DISTROAD))/(2*T/(FULL_L/DISTROAD));
229
230 end
231
232
233 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
234 % Function to Compute Feature Vector for Road Example          %
235 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
236 function ft = features(state, params, fpos1)
237
```

```
238 NUMLANES = params(6);
239 DISTROAD = params(7);
240 FULL_L = params(end-1);
241 FULL_W = params(end);
242 terrain = params(8:end-2);
243 terrain = reshape(terrain, FULL_W, FULL_L)';
244 fpos = ceil((mod(fpos1, FULL_L)+1)/DISTROAD)-1;
245
246 tslane = ones(1, NUMLANES)*(DISTROAD+1);
247 cpos = num2cell(zeros(1,NUMLANES));
248 [lane, dist, cpos{:}] =
249          ind2sub([NUMLANES, DISTROAD, tslane], state);
250 % Reward lane use
251 ft = [zeros(1, NUMLANES) 0.5];
252 ft(lane) = terrain((fpos*DISTROAD)+dist, lane);
253 % Penalise collisions
254 if cpos{lane} == DISTROAD+1
255     ft(NUMLANES+1) = -1;
256 end
257
258 out = zeros(FULL_L/DISTROAD, NUMLANES+1);
259 out(fpos+1,:) = ft;
260 ft = out;
261
262 end
```

Appendix C

Policy Performance Results

	Road 1	Road 2	Road 3	Road 4	Road 5	Road 6
Composite	0.4009	0.2860	0.2860	0.3289	0.3116	0.1200
Primitive 1	0.4767	0.5473	0.4844	0.5251	0.5044	0.3604
Primitive 2	0.4033	0.5118	0.5153	0.4942	0.5089	0.2349
Primitive 3	0.4122	0.5511	0.5311	0.5111	0.5227	0.1871
Primitive 4	0.4198	0.3771	0.6384	0.3647	0.6331	0.1913
Primitive 5	0.4089	0.6531	0.3609	0.6544	0.3660	0.3493
Primitive 6	0.4198	0.5056	0.5116	0.5011	0.5089	0.3402
Primitive 7	0.4153	0.8664	0.1396	0.8791	0.1451	0.5096
Primitive 8	0.4196	0.1647	0.8616	0.1564	0.8713	0.3989
	Road 7	Road 8	Road 9	Road 10	Road 11	
Composite	0.0389	0.0856	0.2351	0.3960	0.2493	
Primitive 1	0.2893	0.4689	0.4944	0.5702	0.4820	
Primitive 2	0.1418	0.2862	0.3347	0.4682	0.3504	
Primitive 3	0.2140	0.4176	0.4196	0.4687	0.3422	
Primitive 4	0.1142	0.2284	0.2229	0.4487	0.3647	
Primitive 5	0.1149	0.2427	0.4300	0.4900	0.3782	
Primitive 6	0.0396	0.0660	0.2487	0.4891	0.4231	
Primitive 7	0.4198	0.6038	0.7433	0.6149	0.5764	
Primitive 8	0.3958	0.6129	0.4224	0.6367	0.5756	

Table C.1: Policy Accuracy Results

	Road 1	Road 2	Road 3	Road 4	Road 5	Road 6
Composite	0.0093	0.0380	0.0353	0.0413	0.0400	0.0227
Primitive 1	0.0149	0.0091	0.0118	0.0122	0.0140	0.0098
Primitive 2	0.0109	0.0098	0.0124	0.0100	0.0127	0.0104
Primitive 3	0.0582	0.0471	0.0547	0.0500	0.0522	0.0531
Primitive 4	0.0111	0.0093	0.0096	0.0091	0.0087	0.0091
Primitive 5	0.0093	0.0124	0.0082	0.0102	0.0093	0.0102
Primitive 6	0.0476	0.0604	0.0569	0.0578	0.0571	0.0533
Primitive 7	0.0542	0.0593	0.0558	0.0547	0.0607	0.0533
Primitive 8	0.0467	0.0573	0.0580	0.0591	0.0580	0.0489
	Road 7	Road 8	Road 9	Road 10	Road 11	
Composite	0.0249	0.0460	0.0482	0.0149	0.0227	
Primitive 1	0.0098	0.0120	0.0144	0.0107	0.0149	
Primitive 2	0.0158	0.0133	0.0089	0.0120	0.0118	
Primitive 3	0.0516	0.0516	0.0567	0.0553	0.0509	
Primitive 4	0.0087	0.0111	0.0118	0.0127	0.0093	
Primitive 5	0.0093	0.0122	0.0098	0.0080	0.0153	
Primitive 6	0.0558	0.0591	0.0627	0.0576	0.0551	
Primitive 7	0.0480	0.0582	0.0509	0.0518	0.0569	
Primitive 8	0.0529	0.0558	0.0560	0.0567	0.0549	

Table C.2: Policy Collision Results