Learning Domain Abstractions for Long Lived Robots

Benjamin Saul Rosman

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh
2014

Abstract

Recent trends in robotics have seen more general purpose robots being deployed in unstructured environments for prolonged periods of time. Such robots are expected to adapt to different environmental conditions, and ultimately take on a broader range of responsibilities, the specifications of which may change online after the robot has been deployed.

We propose that in order for a robot to be generally capable in an online sense when it encounters a range of unknown tasks, it must have the ability to continually learn from a lifetime of experience. Key to this is the ability to generalise from experiences and form representations which facilitate faster learning of new tasks, as well as the transfer of knowledge between different situations. However, experience cannot be managed naïvely: one does not want constantly expanding tables of data, but instead continually refined abstractions of the data – much like humans seem to abstract and organise knowledge. If this agent is active in the same, or similar, classes of environments for a prolonged period of time, it is provided with the opportunity to build abstract representations in order to simplify the learning of future tasks. The domain is a common structure underlying large families of tasks, and exploiting this affords the agent the potential to not only minimise relearning from scratch, but over time to build better models of the environment. We propose to learn such regularities from the environment, and extract the commonalities between tasks.

This thesis aims to address the major question: what are the domain invariances which should be learnt by a long lived agent which encounters a range of different tasks? This question can be decomposed into three dimensions for learning invariances, based on perception, action and interaction. We present novel algorithms for dealing with each of these three factors.

Firstly, how does the agent learn to represent the structure of the world? We focus here on learning inter-object relationships from depth information as a concise representation of the structure of the domain. To this end we introduce *contact point networks* as a topological abstraction of a scene, and present an algorithm based on support vector machine decision boundaries for extracting these from three dimensional point clouds obtained from the agent's experience of a domain. By reducing the specific geometry of an environment into general skeletons based on contact between different objects, we can autonomously learn predicates describing spatial relationships.

Secondly, how does the agent learn to acquire general domain knowledge? While the agent attempts new tasks, it requires a mechanism to control exploration, particularly when it has many courses of action available to it. To this end we draw on the fact that many local behaviours are common to different tasks. Identifying these amounts to learning "common sense" behavioural invariances across multiple tasks. This principle leads to our concept of *action priors*, which are defined as Dirichlet distributions over the action set of the agent. These are learnt from previous behaviours, and expressed as the prior probability of selecting each action in a state, and are used to guide the learning of novel tasks as an exploration policy within a reinforcement learning framework.

Finally, how can the agent react online with sparse information? There are times when an agent is required to respond fast to some interactive setting, when it may have encountered similar tasks previously. To address this problem, we introduce the notion of *types*, being a latent class variable describing related problem instances. The agent is required to learn, identify and respond to these different types in online interactive scenarios. We then introduce *Bayesian policy reuse* as an algorithm that involves maintaining beliefs over the current task instance, updating these from sparse signals, and selecting and instantiating an optimal response from a behaviour library.

This thesis therefore makes the following contributions. We provide the first algorithm for autonomously learning spatial relationships between objects from point cloud data. We then provide an algorithm for extracting action priors from a set of policies, and show that considerable gains in speed can be achieved in learning subsequent tasks over learning from scratch, particularly in reducing the initial losses associated with unguided exploration. Additionally, we demonstrate how these action priors allow for safe exploration, feature selection, and a method for analysing and advising other agents' movement through a domain. Finally, we introduce Bayesian policy reuse which allows an agent to quickly draw on a library of policies and instantiate the correct one, enabling rapid online responses to adversarial conditions.

Acknowledgements

There are times in life to leave your comfort zone, and try something new. For me, that was leaving South Africa to embark on an adventure to the distant and enchanting land of Scotland for my M.Sc. It had long been a dream of mine to head "overseas" for post graduate studies, and following the enthusiastic advice of so many people, I opted for the seemingly mythical city of Edinburgh. Exciting as it was, leaving an amazing web of friends and family to venture off into the unknown was a daunting challenge.

I had been developing a passion for artificial intelligence over the previous few years, and I was amazed at the idea of having so many modules from which to choose. I ended up making all my choices based on what sounded coolest, and even that was difficult. It wasn't long before I found robotics to be the perfect match for me, where in one lecture the lecturer mentioned ideas from almost every module I had ever taken in Computer Science and Applied Mathematics (I'd never thought I'd have an opportunity to use these so well together).

That lecturer was Subramanian Ramamoorthy. From the beginning, I felt inspired by his enthusiasm and deep knowledge of a broad range of subjects, so much so that he supervised my M.Sc. thesis, and I then changed my original plans of moving elsewhere to do a Ph.D. to continue to work with him. I'm so glad I did. He encouraged me to explore different avenues of research, and was always open to me dropping in to chat about my results, giving me good advice about how to tackle problems, or even discussing philosophy in general. I have learnt so much from you Ram, about so many things, and I thank you so sincerely for all of that. You have truly made this Ph.D. experience worthwhile and set the example of the researcher I would someday like to become.

I would also like to express my sincere gratitude to the other members of my Ph.D. panel: Sethu Vijayakumar and Alan Bundy. Their advice and suggestions throughout my studies have been invaluable, and it has been a very useful exercise for me to think about my work from the different angles they represent.

I am deeply grateful to my examiners, Jeremy Wyatt and Mark Steedman, for taking the time to work through my thesis and provide their incredibly helpful comments for improving my work, as well as the interesting discussions during and after my *viva*.

This work also would not have been possible without the financial and moral support of Simukai Utete and the rest of the Mobile Intelligent Autonomous Systems (MIAS) group at the Council for Scientific and Industrial Research (CSIR) in South

Africa. I am truly grateful to them for providing me with the means to pursue my dreams.

Ever since collaborating with him remotely on a paper in the second year of my undergraduate studies at the University of the Witwatersrand, I'd heard many stories about George Konidaris. I ended up inadvertently following him to Edinburgh (after he'd already moved on) and we then discovered we had similar interests. I remember finally meeting him after a long bus ride to Amherst feeling like I was meeting a long-lost friend. Thank you so much George, for inspiring me with your success, taking me under your wing, encouraging me when I was down about my work, introducing me to so many influential people, and making me feel a part of the community.

My time working in the Informatics Forum has been greatly enhanced by the company of some inspiring office mates, who made a day in the office all the more enjoyable. Thank you to Ioannis Havoutis, Aris Valtazanos, Majd Hawasly, Zhe Liu, Stathis Vafeias, Stavros Gerakaris, Alesis Novik, Stefano Albrecht, and Paul Andreadis, as well as Alex Bordallo and Tom Larkworthy. I would especially like to thank Majd Hawasly, Hassan Mahmud and Pushmeet Kohli, with whom I worked closely, and who played a critical part in my thesis work. I learnt so much working with you, and it was an absolute pleasure!

The great people were not limited to my office. So many others in IPAB chipped in to make it an amazing experience. I will always fondly remember the 6.00pm call to IPUB at Teviot, and the many great chats and laughs (and drinks and karaoke) over the years with Adam Barnett, Vlad Ivan, Joe Henry, Pete Sandilands, Georgios Petrou, Jun Nakanishi, Andreea Radulescu, Steve McDonagh, Chris Towell, He Wang, Hsiu-Chin Lin, David Braun, Bas Boom, Luis Horna Carranza, Cigdem Beyan, Xi Zhao, Michael Mangan, Matteo Leonetti, Matt Howard, Djordje Mitrovic, Helen Ramsden, Hannes Saal, Ian Saunders, Sebastian Bitzer and Stefan Klanke.

I am lucky to have the most wonderful group of friends anyone could wish for. Thanks to my former flatmates Jared Golden and Eric Lynch for the countless good times. A huge thank you to Aciel Eshky (and Ben, Teema and Adam) for always having such a wonderful, warm and inviting home. Thank you so much to Jelena Tomanikova for organising and hosting so many great holidays. Thank you to Amy Champion for making sure each day was even more "punderful" than the last. A grateful thank you to Deanne Goldberg for continuously checking in on me to make sure I hadn't done anything too stupid, and always being so supportive, encouraging, inspiring, and there to listen to my issues. Thank you to Peter Orchard for the technical advice,

and the regular lunches and fascinating and motivating discussions on everything from business plans to the fate of the universe and mankind. Thank you so much to each and every one of the wonderful people in my life, in Edinburgh, throughout South Africa, and all over the world. You make life awesome!

I would not be where I am today if not for so many other critical people. An enormous thank you to Gloria Spambo for looking after me all those years, teaching me about so many facets of life, culture and people, and ultimately putting me on the road to self-sufficiency (I hope I haven't done too badly). Thank you so much also to Morag Rees for encouraging and helping me to set forth on my adventures abroad and always reminding me to keep painting, to Sarah Rauchas for giving me my first taste of research and putting Edinburgh on my map, and to George Christelis for welcoming me to Edinburgh so warmly.

My memories of Scotland will forever be dominated by my "Marchmont Crew": Helle Hang, Philipp Petrenz, Claire Giry, David Braude, Kat McNabb and Whiski McGiry. You guys were my partners in crime, my Edinburgh family, and made it really feel like home. I will always fondly remember our group adventures, camping and boating trips, travels, parties, and just hanging out and living together. I'd never realised that real life could feel like an award-winning sitcom until I met you all. Thank you for the best laughs of my life.

Philipp and Helle, I will never think of boardgames, vegetarians, mismatched socks, bicycles or Christmas the same way ever again. Dave and Claire, thank you for putting up with such a bizarre flatmate, and making sure I would end each day with a smile. Whiski, thank you so much for always reminding me to stop worrying and love the cat, and for being a furry pillow of fun and claws when I needed it most.

Kat, you have touched my life in ways I cannot adequately describe. Without your love, support, patience and encouragement none of this would have been possible at all. Thank you so much for always looking after me, making sure I had enough brain food, and ensuring I had appropriately themed underwear. I really appreciate everything, Kitteh!

Finally, I want to extend my eternal thanks to my family, without whom literally none of this would have been possible. Their unwavering love and support (for everything except my long hair) have really been a pillar of strength and encouragement.

I remember as an eight-year old kid sitting with my Dad one summer evening in our lounge in Johannesburg in front of our old BBC Microcomputer with a manual in front of us and typing in BASIC commands. I was instantly amazed at how we

could make the computer print things out to the screen, and respond to user inputs. It wasn't long before I was spending hours designing games and applications of various sorts, and sometime soon thereafter computers and programming usurped dinosaurs and palaeontology as my number one passion. Mom and Dad, without your constant care and support for everything I do, and the encouragement to always take that extra step, I certainly wouldn't have made it this far!

Thank you Dad for always inspiring me in my pursuit of knowledge, and for always reassuring me that all my results will work out if I just "draw a graph". Thank you for the long discussions on everything from science to philosophy over the years – they have been foundational to my way of thinking and my approach to my work. Thank you Mom for always checking in on me and making sure I was happy and healthy, having all my vitamins and wearing jerseys, and even offering to fly out to look after me whenever I was upset or ill. Thank you also for all the emotional support throughout the years, for showing me I was never alone and always in your thoughts, and for always remaining a voice of optimism through the difficult times – you inspire me so much in the way you positively touch the lives of all those around you, and I've always aspired to follow the example you set. I really cherish being able to turn to the two of you for absolutely any advice I need, confident that you will always have sound rational guidance for every situation, even if it's that you "don't understand a word but think it sounds great anyway". You have always both been such incredible role models to me, and I really cannot express how much I appreciate everything you have both done for me!

Also, thank you both so much for taking the considerable time to proof-read so many of my documents, most notably this thesis! I know it was certainly not the most enjoyable way to spend a few days, but as usual my work and grammar is considerably better for it.

Lastly, a huge thank you to my little brother Adam – the practical one in the family. You are the one who truly taught me the value of breaking things (and then fixing them) in order to understand how they work. Thanks, too, for all your constant support, for fixing all my things whenever you visit, and generally being such a source of pride and inspiration. And what would I have done without the constant supply of cool robots you kept finding and sending me?

Thank you to every single person I have met along the way: every interaction, no matter how small, has helped shape me into who I am today!

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 Saul Rosman*)

to my family

"I learned that courage was not the absence of fear, but the triumph over it. The brave man is not he who does not feel afraid, but he who conquers that fear."

– Nelson Rolihlahla 'Madiba' Mandela

(1918/07/18 – 2013/12/05)

Table of Contents

Chapter 1

Introduction

Humans, and indeed other animals, have many remarkable cognitive abilities. One of the most fundamental of these is the ability to adapt to changing conditions, and so too to perform well under completely new conditions. Key to this seems to be our ability to form abstract mental models of different environments, and use a combination of previous experiences and observations of how the current environment responds to our prompts. We use these models to aid in tuning and selecting behaviours that are appropriate to our current conditions.

Being able to generalise in this way is a very powerful ability. Learning from scratch is necessarily a slower process than reusing knowledge, and being able to bias which knowledge is reused based on environmental cues reduces the combinatorial search that is required to formulate plans over some time horizon.

A core component of this ability lies in being able to understand general principles underlying the structure and evolution of the world with which we interact. In a general sense, the brain seems to possess intuitive principles of physics, biology and psychology, which guides the way it selects behaviours, by being able to form predictions of their outcomes (Friston et al., 2009). As an example, consider a hunter-gatherer who is able to infer both the species of an animal and its behaviour from its tracks, and in so doing be able to intercept it at a water hole, or alternatively throw rocks and spears at intended targets, perhaps coated in poisons extracted from plants (Pinker, 1999). All these behaviours require some understanding of the dynamics of many general systems, be it that animals need regular food and water, or an intuitive understanding of the effects of gravity on an object leaving a hand at speed.

Basic principles and abstractions such as these enable humans with limited computational abilities, i.e. *bounded rationality* (Simon, 1955), to employ fast responses

to the infinite range of situations they may encounter, by transferring ideas between similar situations. The speed of these responses comes with the drawback of a loss of optimality, but being able to invoke such *satisficing responses* implies a great versatility for our species (Simon, 1956).

Drawing inspiration from these cognitive traits of humans, we are interested in investigating how an autonomous artificial agent such as a robot can learn and act in a similar way in an environment over a prolonged period of time. The longer the robot works in the same or similar environments the more proficient it should become, through having had to solve multiple tasks. We seek to address this idea by examining the kinds of information which can be extracted and reused in an environment with which the robot develops some familiarity.

1.1 Research Problem

As an example of the type of scenario we ultimately wish to address, consider an assistive robot in a hospital environment, as caricatured in Figure 1.1. In this setting, the robot interacts with a stream of patients, each with different symptoms, requirements and preferences, many of which cannot easily be pin-pointed through a simple discussion or diagnostic procedure. Similarly, the hospital staff which the robot assists each have different mannerisms, lists of tasks with which the robot should help, and particular ways in which they would like these tasks to be done. Furthermore, in order to complete the prescribed tasks, the robot needs to use tools and manipulate objects, some of which are in particular places such as hospital equipment, and others are patient specific, such as personal belongings. As a result, the robot must be able to identify and find required objects. The overall effect is that the robot is presented with a series of *partially specified* but related tasks, and as it gains experience over time, it should learn to perform these faster and more efficiently.

The primary over-arching question we address in this dissertation is thus *what should be learnt by a lifelong learning agent in a large world with many options, and how can this knowledge be reused?*

Building the autonomous assistive robot of Figure 1.1 would be a major undertaking, beyond the scope of this thesis. However, this illustration raises many problems to be solved, and in this thesis we focus on the core issue of generalising and reusing knowledge. To that end, we address three particular instances of knowledge acquisition and reuse:

Figure 1.1: An assistive robot in a hospital ward is required to provide regular and fast interaction with patients, nurses, doctors, and numerous objects of different varieties.

1. The assistive robot may often be required to carry and use a range of medical equipment and personal effects, be it retrieving scalpels lying on trays or blankets folded inside bags. A property which is common to many different physical manipulation tasks is the relative positioning of different objects. This is critical in planning, where a bed should only be moved once the patient is on it, but not when any other furniture is adjacent to it. As a result, we study the learning of *spatial relationships between objects* (discussed in Chapter 2).

2. Every action taken by the assistive robot is likely to be affected by situation specific constraints. With general navigation it must avoid collisions with objects, when moving between rooms it should be wary of the directions in which doors open, and its speed should always be modulated by the environment through which it is moving. When interacting with different objects, each object has a particular way in which it should be handled, and further, many behaviours are not appropriate in most situations. For example, surgery procedures should only be invoked in an operating theatre. The diversity and quantity of these constraints

means that not all can be explicitly pre-programmed, yet they are consistent between different tasks, and knowledge thereof greatly accelerates the planning and decision-making procedure. We address this and related concepts through the learning of *action priors* (discussed in Chapter 3).

3. For many particularly interactive tasks, the assistive robot may have a number of different behaviours or configurations which may be relevant. Examples of this include different personalities for dealing with patients, different instruction sets for interpreting patient gestures and commands, or even different ways of cleaning up a threatre after an operation. The choice of behaviour here depends on the patient, doctor, or other factors around the robot, and often the use of the wrong one at the wrong time may result in the task being poorly completed. The implication is that the robot should maintain models and estimates of which behaviours are the best to use in certain situations, and we deal with this in our presentation of *types* and *Bayesian policy reuse* (in Chapter 4).

What we ultimately desire is robots that are able to exist for some long duration in a certain environment which may change over time, and be able to perform many different tasks therein. This template of robot behaviour is needed for any general purpose robot, which could be useful in applications such as healthcare, where a carer robot may be required to assist people with a wide variety of tasks, the requirements of which may differ between individuals.

The acquisition of behaviours and skills is well-studied within the robotics and agents literature, particularly in the scope of optimal control theory and reinforcement learning. Some notable examples include different reinforcement learning approaches (Peters et al., 2003; Kober and Peters, 2011), learning motion primitives (Schaal et al., 2004), hierarchical reinforcement learning in discrete (Barto and Mahadevan, 2003) and continuous systems (Konidaris, 2011), and imitation learning and learning from demonstration (Argall et al., 2009). These take into consideration a number of different goals, settings and conditions for learning.

However, skill acquisition is only one side of the problem. Once a number of skills have been learnt, naïve temporal planning with those skills easily becomes an exponential search problem through a high-dimensional space. The challenge then is how to structure, manage, and use these skills efficiently.

In order to do this, we seek to learn the structure and invariants of domains in which agents are required to solve large families of tasks quickly, and in an ongoing manner.

The goal is thus to learn about the general manifolds governing large classes of tasks and behaviours. This provides additional information which can be used to prioritise and select between different behaviours when confronted with new tasks drawn from the same family, and so control the size of the behaviour space being searched.

We are motivated by the principles of bootstrap and developmental learning (Pierce and Kuipers, 1997; Kuipers et al., 2006), through autonomously discovering different forms of statistical regularity in the experiences of the agent, and using this as the basis of abstractions and knowledge representation. Additionally, we draw inspiration from the methods of transfer learning (Taylor and Stone, 2009), as we wish to accelerate learning through the application of knowledge acquired in a set of previous tasks (Thrun, 1996a). This general paradigm is often referred to as lifelong learning (Thrun, 1996b).

Our ultimate goal is to be able to build capability models (Rosman and Ramamoorthy, 2012a) while acting online, which involves learning an association of behaviours to observational elements of the environment. In this way, we aim to use environment signals as a behaviour guide, inspired by the ideas of reactive control (Brooks, 1986; Braitenberg, 1986), but using these signals to prioritise behaviours. This allows an agent to act with a coarse optimal behaviour in situations with extended uncertainty (Bookstaber and Langsam, 1985). The core theme unifying these ideas is *abstraction*, as appropriate abstractions enable knowledge and behaviours to be learnt in a generalised manner, and thus reused in new situations.

1.2 Abstraction in Artificial Intelligence

The idea of abstraction has a long history, and is a commonly recurring theme throughout artificial intelligence (AI). Since the early days of AI, abstraction has been advocated as a useful tool in building intelligent systems (Minsky, 1961). Many modern machine learning and decision theoretic models rely on abstractions of some decision space. For example, simple decision trees classify data or make decisions based on partitioning an input space (Bishop, 2006). On the other hand, more sophisticated machine learning tools, such as deep belief networks (Bengio et al., 2009) and hierarchical probabilistic generative models (Tenenbaum et al., 2011), decompose difficult classification problems by learning hierarchies of abstractions so as to factorise a large space of concepts, which overlap in features at some level of abstraction.

Abstraction is a broad term, generally taken to be a “mapping between formalisms

that reduces the computational complexity of the task at stake" (Zucker, 2003), and in practical senses usually refers to either simplifying or removing unnecessary details from a problem so as to speed up its solution (Holte and Choueiry, 2003). For example, focusing on the important parts of a process, such as the points at which some *qualitative* change occurs, allows one to reason about a continuous system rather as a finite set of qualitatively meaningful states (Kuipers, 1994). This approach can be used to reason elegantly about complicated dynamical processes (Ramamoorthy, 2007).

In this thesis, we employ the idea of abstraction as a *grouping* or *aggregation* of similar situations in which an autonomous agent may find itself, such that behaviours which are useful in one of those situations can be successfully applied to any of the others, and in doing so accelerate the decision making and learning processes. In this way, we identify general concepts true to multiple instantiations of a problem domain, and thus take a step towards "common sense" knowledge of that problem.

Generalising from sparse data is critical to the way in which humans learn and acquire knowledge, as seen for example in the way concepts are formed in language (Tenenbaum et al., 2011). Humans also seem to do planning in abstract spaces, and often difficult problems can be solved by focusing on reformulations of the original problems, and thinking about them in different ways (Pólya, 1945).

In the case of learning and planning agents such as robots, abstractions typically appear in one of two primary forms: state abstractions, and action or behavioural abstractions.

State abstractions involve redescribing the state space of the agent. There are a number of reasons for doing this. One common reason is for discretising continuous state spaces, for example in reinforcement learning, so that policies can be learnt and described as mappings from states to actions. Another reason is to reduce the size of problems, typically by aggregating states (Horvitz and Klein, 1993; Li et al., 2006), to either generalise experience (e.g. (Leffler et al., 2007)), or to increase the speed of learning (e.g. (Jong and Stone, 2005)), possibly by having to explore fewer possibilities (Hester and Stone, 2009). All these cases are useful for transfer, as the effective representation of the domain of the agent has been simplified, and as a result, each state configuration in the simpler representation is more likely to be encountered in future tasks than a state of a larger (higher dimensional, or continuous) representation.

Alternatively, action abstractions are often based around the ideas of combining primitive actions. An example of this is the *options* framework in reinforcement learning (Precup et al., 1998), where options are defined as composite behaviours (usually

composed of primitive actions), with their own start and termination conditions. These can be executed by an agent to accomplish something that would otherwise have required an exact sequencing of several primitive actions, and in doing so the agent can reuse knowledge to speed up learning. This idea of defining more abstract actions dates back to early work, such as *macro actions* (or "macrops") in the STRIPS planner (Fikes et al., 1972). Actions can also be abstracted by redefining them to apply to more general contexts, such as ABSTRIPS which defined abstraction hierarchies through different levels of details in the preconditions of actions (Sacerdoti, 1974), or *deictic references* which identify objects relative to the agent (Pasula et al., 2007).

1.3 Thesis Overview

In order to learn and plan, an agent is required to have some understanding of the structure of the domain within which it acts. Given the identity of objects in an arbitrary environment, arguably the most important feature of that environment for formulating plans is the set of relationships between those various entities within that domain. Typically these are hand-crafted and provided to the agent, but in order to achieve greater flexibility and autonomy, an agent should be able to learn important relationships autonomously.

Chapter 2 presents a study of learning such spatial relationships between objects in a three-dimensional scene. We present an algorithm which, by examining many pairs of objects, can learn these relationships as well as the topological structure of objects as they are used in the scene, by considering their points of contact with other objects. This provides a functional abstraction of the scene, and an important step towards scene understanding in robot vision.

The method presented in this chapter is based on learning geometric separators between objects, and this is done through support vector machines. The graph structure between the support vectors of a single object defines a topological abstraction of that object, and that between the support vectors of different objects describes their spatial relationships.

One of the core challenges of lifelong learning is being able to use past experiences to guide solutions to novel problems. Chapter 3 introduces *action priors* as a mechanism for biasing online action selection towards behaviours that were more useful in the past in similar situations. Through repeatedly solving different problems in the same general environment, we show that the agent is able to learn a model of "com-

mon sense" behaviour in that environment, and consequently greatly increase solution speed of new tasks.

We focus on reinforcement learning domains in this chapter, and posit the learning of context specific distributions over action space from multiple tasks as an effective strategy for locally pruning the action space (and thus the branching factor in temporal planning), as well as action prioritisation.

Action priors are accumulated as state-based Dirichlet distributions over the action space, from multiple optimal policies, to provide a multi-task notion of utility maximisation. We additionally present an extended notion of these priors which are instead conditioned on observable variables, and show that this formulation is useful for observation feature selection.

In many kinds of interactions, an agent may have a number of solution policies which all seem viable, owing to the fact that some aspect of the problem is unknown to the agent. If the agent is expected to have multiple interactions under the same conditions, it should be able to infer those latent aspects of the problem to some extent, and use this information to better its own performance. To this end, Chapter 4 introduces *Bayesian policy reuse* as a paradigm for selecting between behavioural choices in an efficient manner, so as to both gain information about the current task, and improve performance therein.

This decision making method is posed in a multi-armed bandit setting, where the policies are a number of options from which to choose. However, by learning a correlation between the performance expected from the different options, which occurs as a result of properties of the space of tasks faced by the agent, we draw on ideas from Bayesian optimisation to allow for a family of fast, sample efficient Bayesian algorithms.

1.4 Experimental Domains

Aspects of the general problem considered in this thesis, being abstractions and reuse in lifelong learning, have implications for a range of other problems. We now briefly discuss the experimental domains used throughout this thesis, and relate them to the overall picture.

In Chapter 2, when evaluating our algorithm for learning spatial relationships between objects, we use stereo-image pairs, which provide depth information for different configurations of physical objects. This is in line with how these relationships

would be learnt on a lifelong robot.

Action priors are presented in Chapter 3, where they are discussed as being useful in situations where an agent is required to solve a number of tasks in sequence within the same, or similar, environments. We note that one particular example of this requirement is seen in navigation tasks within a single building. We thus illustrate and evaluate action priors on several grid-world domains, as are typically used within the reinforcement learning literature. To add richness to the tasks, we allow the agent to execute simple 'manipulation' actions in some of the tasks, to demonstrate the effect of larger action spaces and show that our principles apply more generally.

Finally, Chapter 4 presents Bayesian policy reuse as applying to problems where the agent has the opportunity for thorough offline training under a number of different conditions, but online is required to make rapid decisions about new scenarios, and perform well with only a few available trials. These specifications are common in interactive tasks (such as computer-human or robot-human interaction) and surveillance, tracking and monitoring tasks. For this reason, our two primary experiments in this chapter involve an interactive caller system, and a wildlife monitoring system. Some smaller examples in video game scenarios are also presented for illustrative purposes.

1.5 Major Contributions

This thesis presents an advancement to the learning of abstractions for long-lived agents to successfully solve tasks in their environments better over time, along three major dimensions. We present five primary contributions: three of which directly address to the issues discussed above, as well as two additional contributions based on applications of our core algorithms.

1. *Learning spatial relationships*. We present a novel algorithm for learning both the spatial relationships of a scene represented as a three-dimensional point-cloud, and a topological description of the scene structure. This has direct applications in robot vision and planning.

2. *Action priors*. We introduce this novel concept for boosting the learning speed of new tasks, given previous experiences. Additionally, we provide algorithms for learning and using the action priors, and show that they provide a transfer approach to domain reduction. This is a major contribution to multi-task and lifelong learning decision making.

3. *Advice giving*. In a novel application of action priors, we show how an agent studying the movement and behaviour of multiple other agents in a particular environment such as a building, can use that experience to advise new agents who may be lost, providing large speed-ups in their task solution times.

4. *Bayesian policy reuse*. We introduce Bayesian policy reuse as a general algorithm for selecting and instantiating pre-learnt policies from a policy library, quickly in an online manner, so as to maintain low regret and sample complexity in new instances of tasks drawn from a family with which the agent has prior experience. This contributes a novel family of decision making algorithms for constrained time (in sample complexity) problems, where offline training is available.

5. *Online interface adaptation*. A specific case of Bayesian policy reuse, the BEAT algorithm, is introduced for solving the important problem of online adaptation of interfaces to different users.

Chapter 2

Spatial Relationships

The work presented in this chapter also appears in: Rosman and Ramamoorthy (2011).

2.1 Introduction

As robots become more capable of performing a wide range of tasks, and move from carefully engineered to open and unknown environments, the need for a concise representation of a widely varying world is becoming more pertinent. These robots must deal with immense variability in the structure of the world, for example a manipulation robot may find objects in a wide variety of configurations. In order for a robot to be able to manipulate these objects, it needs to understand the qualitative structure of the relationships between them independent of other quantitative variation.

Furthermore, with the move into more natural environments, robots are more likely to encounter objects with which they have had minimal, if any, previous experience. An increase in the number of these poorly known objects in the vicinity of an active robot suggests that new strategies for exploring and interacting with these objects would be required. It is infeasible to enumeratively represent all aspects of some real-world scene if we want an agent to use that information subsequently in real-time decision making. Another situation where enumeration is difficult is when manipulating articulated and flexible objects: complicated objects yielding seemingly disorganised point clouds, but with much qualitative structure.

As a result, we are particularly interested in the way in which different objects can be represented, in terms of their own structure as well as the way in which they relate to each other spatially in a scene. This would provide a redescription of the scene using a layered representation, including a qualitative level which provides interesting

topological features of objects that could be used for disambiguating actions, such as that holes are candidate contexts for the action of inserting a finger to pick up an object, or that points of contact between two objects constrain the relative motion of those objects.

In this chapter we thus address the issue of how to redescribe a scene in terms of abstractions, given a three-dimensional point cloud representation of that scene containing several objects. We use a layered abstraction, consisting of a skeletonised description of the objects themselves, as well as a *symbolic* description of the spatial relationships between these objects. These relationships are important in manipulation and decision making, allowing for a simplification of task specifications, as well as vagueness and robustness in planning.

In this work we are less concerned with detailed object identification, but rather are interested in separating a scene into potential objects that can be manipulated, and examining the way in which these objects are used as structural components of the environment. In this way, a wooden box and an intricate overturned vase could be considered to be topologically equivalent, in that they are both single connected component structures on which other objects can be placed. This view of what constitutes an object is inspired by emerging notions regarding topological invariance and qualitative structure in a wide variety of data sets, as in work by Carlsson (2009), and illustrates the level of description we are aiming towards with our representation scheme.

In order for a robot to efficiently manipulate objects in some environment, it needs to know something about how these objects relate to each other, e.g., how an object is supported by other objects and constrained in motion by yet other objects – a class of qualitative relationships defined by specific features such as contact points. We seek to redescribe a scene in terms of these types of relationships. The most important of these are arguably $\texttt{on}(\cdot,\cdot)$ and $\texttt{adjacent}(\cdot,\cdot)$, and while these are clearly a subset of the wide variety of qualitative relationships possible, we restrict our attention to them. By learning and identifying these relationships in the surrounding environment, a robot can use these concepts in the context of motion synthesis, particularly for performing tasks which require the use of several objects.

We choose to work with point clouds here as they are gaining prominence as a representation of the world in a number of new mobile manipulation platforms. This is further driven by the move towards unified and generalised operating systems for robotic platforms, such as the open-source ROS[1], which comes complete with a Point

[1]Robot Operating System – www.ros.org

Cloud Library (PCL). Point clouds provide a representation for the three-dimensional information that a robotic platform may extract from the world around it, in a number of ways ranging from the registration of stereoscopic cameras, to reconstructions from laser range scanners. As such, building a point cloud is a convenient[2] method for representing raw depth information about the world around a robot.

This chapter describes our proposal for a layered description of a scene, derived by a novel algorithm for extracting the structure of objects and then classifying the spatial relationships between the point clouds of these pre-segmented objects. Our method creates a contact point network for each object as an abstraction of the object into a graph-like skeleton which identifies points where the object either touches other objects, or comes close to touching them. This structure is a potentially useful abstraction for manipulation, as it identifies regions where the object has an interface with the external world. These may be useful candidate points for functional components and surfaces of the objects, and the edges between these points indicate constraints on the objects that could be factored into motion synthesis and planning. More concretely, these edges provide seeding regions, around which the search for grasp points can begin. In a sense, these skeletons can be considered as being loosely related to the concept of affordances (Gibson, 1986), to the extent that contact points and intra/inter-object holes indicate constraints and possibilities for manipulation motion synthesis. These constraints thus allow for the disambiguation between families of motion strategies at a global level for a particular environment, and an agent may make use of this knowledge to provide a viable first pass at synthesising behaviours.

2.1.1 Contributions

This chapter presents a supervised method (and outlines an unsupervised method) for learning spatial relationships between objects, from point cloud data. The approach is based on using support vector machines to skeletonise the representations of the objects, so as to allow for their mutual topological structure to be classified. The effect of this is the acquisition of operators, in particular defining the concepts of “on” and “adjacent”, which can subsequently be used for planning.

By finding a topological abstraction of inter-object relationships, we provide a context for policy transfer between similar situations, which may arise between geometrically different object configurations. Furthermore, we show that some structures con-

[2]Point clouds can now be easily constructed by low-cost sensors such as Microsoft’s Kinect for the Xbox 360.

tain simpler elements as substructures, which can also be used to seed plan or policy instantiation.

2.1.2 Chapter Structure

This chapter is organised as follows. First we present related work in Section 2.2. Our algorithm to extract structure from a scene and learn spatial relationships from point cloud data is described in Section 2.3, with experimental results in Section 2.4. A discussion of these results appears in Section 2.5, and a summary of the conclusions, contributions and future work is in Section 2.6.

2.2 Related Work

The question of how to learn the spatial relationships between different objects has received relatively little attention in the literature. This is an important question, as what is often difficult about motion planning is disambiguating between the global structure of candidate plans, and so finding good initial guesses at the appropriate course of action for some new scenario. Many researchers have examined the problem of identifying objects in a scene (Jain and Dorai, 2000; Leibe and Schiele, 2004; Desai et al., 2009; Alexe et al., 2010), but instead of focusing on object identification, we wish to extract coarser types of functional information which may aid a robot manipulating in that environment, by describing the structure and constraints of the scene.

A notable body of work on learning spatial relationships appears in the thesis of Sjöö (2011). This presents two lines of attack towards the problem of learning spatial relationships.

The first of these is a model for two spatial relationships: *on* and *in*. In this work *on* is defined as a predicate describing one object providing geometric support to another against gravity. The model of *on* defines three criteria, based on the separation between the objects, the horizontal distance between the centre of mass and contact point, and the inclination of the normal force between the objects. These criteria are evaluated and combined, to give a score describing how well the predicate *on* is encapsulated by the scene. Similarly, *in* is described as containment, and this is measured by a combination of the extent to which one object falls within the convex hull of the other as well as a penalty on the apparent object interpenetration.

By choosing appropriate parameters for these criteria, accurate descriptions of

physical scenes can be generated in terms of the inter-object relationships. The problem with this method is that it requires that each relationship be modelled in detail, to the level of forces and parameters, by a human designer.

This is not a requirement of our algorithm, which determines the relationships autonomously from labelled data. The second approach employed by Sjöö (2011) achieves this as well. This approach defines some functionally distinctive relationships between objects: support, location control, protection and constraint. For each of these relationships, different probing actions are designed to test for the corresponding condition. These tests are run multiple times in simulated environments with randomly generated scenes, and a sparse regression algorithm determines which of 93 features are needed to predict these relationships. Human intervention is still required to design these tests, and for general relationships the use of a physics simulator may not adequately model real-world object interactions.

Galleguillos et al. (2008) examine the relative spatial arrangements of patches in two-dimensional images, and use this to disambiguate object labels. These relative spatial arrangements are defined in terms of bounding boxes around patches, as well as their relative centroid locations. This provides an abstraction of a complex scene, in a similar manner to what we desire in determining abstractions which capture the relationships between irregularly-shaped point clouds corresponding to regions of objects in a three-dimensional environment. We seek a version for extracting manipulation specific information from a scene. A similar approach, based on bounding boxes, is successfully used by Dee et al. (2009) to learn relationships between parts of frames in videos which correspond to regions experiencing similar motion. The relative spatial relationships between regions of an image are often used to improve interpretation and recognition in the scene (Galleguillos and Belongie, 2010), working on the assumption that certain objects are more likely to be found in the context of, or in particular positions relative to, certain other objects.

An alternative approach to learning relationships is to learn them from scenes combined with natural language utterances of people, in an attempt to identify correlating visual and auditory regularities. This is the approach taken by Oates (2001) who developed a system capable of, amongst other things, identifying relationships such as *on* and *above* between objects. Identifying these relationships required the use of specific features, and were identified based on sensors measuring the angle of the line segment between the two objects where they are closest, the length of that segment, and the angle of the line segment between their centres of mass.

An intermediate step between logical plans and low-level control is the reasoning about large-scale structure of motion plans such as in work by Hauser and Latombe (2010). These approaches require the ability to appropriately seed, at the level of topology, the search process through strategy space.

Relational predicates are also an important component of first-order logic, which allows for reasoning about entities and the relationships between them. As a result, first-order logic is the language of choice for many planning problem descriptions and algorithms (Ghallab et al., 2004), as the goal of planning is the discovery of plans of action which can cause certain relationships between various objects in the domain of the problem to become true, remain true, or cease to be true.

For instance, the relationships between different regions in space can be defined and reasoned about using an interval logic known as region connection calculus (RCC), similarly to the way in which temporal logics reason about time and events (Randell et al., 1992). This provides an axiomatised description of the ways in which two regions can relate, such as 'overlaps', 'contains' and 'equals'. This theory provides no mechanisms for extracting these relationships from visual data, but provides a language for reasoning about them at a higher level.

Spatial relationships between objects in the physical (or a simulated) world are thus useful elements for agents in reasoning about planning tasks in worlds with multiple objects, for example stacking blocks to build towers (Pasula et al., 2007). The focus of such work is typically on methods for making inferences about the relationships that hold in the world, rather than deducing these relationships themselves. As a result, a hard-coded approach is often taken, where rules for recognising relationships can be designed by the system developers. An example of the usefulness of the relationships between parts of a domain in solving a problem can be seen for example in the robotic towel folding of Maitin-Shepard et al. (2010). By finding the key structural features of towels, namely the corners and edges, as well as the relationships between them (hand-coded by the designers), a robot was able to successfully fold a towel. We are interested in extracting this type of structure and relationship in more general scenes.

A more general example of how such rules could be hard-coded would be: "if there are points in object A above points in object B, with a vertical displacement below some ε, then A is on B". These rules are specific to each particular problem, and rely on the fact that these systems often use simple geometric objects such as blocks. As a result, these rules easily break down for the kinds of general objects which one would expect a personal robot operating in a home environment to encounter. There is

thus a need for more flexible methods for detecting relationships between objects.

Relationships between pairs of tracked vehicles moving on a road have been considered previously by Galata et al. (2002). The relationships captured between vehicles are similar to the types of relationships we seek, but we focus somewhat more on achieving a robotic manipulation centric representation. Additionally a crucial component of our layered representation is that we abstract some notion of the internal structure of the objects which may not be relevant in the application considered in Galata et al. (2002).

One notable approach to addressing the related problem, of relationships between regions and elements of a scene, is that of Waltz (1975). This work examines the broad problem of reconstructing a three-dimensional description of a scene, given a line drawing of that scene. A scene in this case is restricted to a set of planar-faced objects, described by the edges of each plane, and shaded areas to indicate shadows. The crux of this method is in labelling the line segments which are object edges. The edge labels indicate whether the edge describes features such as the edge of a shadow, a concave bend or a convex bend. The labels are assigned based on the types of junctions at either end of the line segments, as well as local information such as shadows and region orientation. Knowledge of the types of edges between two different objects in a scene allows the system to deduce when one object is in front of, behind or supporting another. This work was limited in the strong requirements placed on the input data: that it be line drawings of planar-faced objects. We instead seek an approach which can handle point cloud data, of any objects.

Planar faces can be easily described by line drawings, but more complicated objects require a richer representation. The idea of representing an object as a graph, being some skeletal abstraction of the object, has found interesting uses in describing the topology and other symbolic properties of individual objects (Pinz et al., 2008). This is a sparse representation of what may be otherwise complicated objects. An example of this by Katz and Brock (2008) provides a method for discovering articulation points in objects, based on which parts of the object maintain a constant separation while a robot is moving that object. In this way, nodes of the graph correspond to distinctive features of the object, and edges exist between two features if the distance between those features has never exceeded some threshold.

We propose to build up from low-level point cloud data as acquired by sensing devices, into a layered representation for redescribing the scene to aid in reasoning and planning through the use of graph-like abstractions of objects. Instead of basing

these graphs on features of an individual object, they arise from the way in which the object structurally forms part of a scene or environment. Much previous work has been dedicated to detecting affordances of objects of various sorts, such as Saxena et al. (2008) showing how good grasp points can be identified on an object, Rusu et al. (2009) describing curvature and other geometric properties, and Barck-Holst et al. (2009) showing how such grasp affordances can be learned from an ontological reasoning system. Works such as these often focus on statistically finding useful local features. The next level up would be to look at how the scene is composed of these local features. To this end, we consider the relationship patterns between sets of objects, with the aim of learning high-level concepts relating to the topological structure of an entire scene. This provides a platform from which inter-object spatial relationships are inferred, giving a high-level representation which may be useful for motion synthesis by reducing the dimensionality of the search space.

We comment here that this work, first published as Rosman and Ramamoorthy (2011), has subsequently been studied by other authors. These include Dearden and Burbridge (2013), who learn probabilistic models of the relationships between objects from a number of geometric features, and Fichtl et al. (2013), who classify relationships by building 2D histograms of the distances between all subregions of the two objects.

2.3 Algorithm

2.3.1 Overview

Our algorithm builds a layered representation of a scene, by extracting spatial relationships which exist between objects, as well as a topological description of those objects, in order to redescribe that scene in terms of its structural properties. This description is based on easily observed features of the scene, such as object candidates and contact points between the objects. The assumptions made by the algorithm are described in Section 2.3.2. The object candidates used by the algorithm are derived from segmented point cloud data of the scene, and a discussion of a simple method for object segmentation is provided in Section 2.3.3.

Contact points are another visual feature used for extracting the structural representation of the scene. To define these, the algorithm relies on the concept of geometric separability between two different objects, and in particular identifies regions where

the margin of separation between the two objects is small. These regions are most likely candidates for points of contact between the objects. The reason that geometric separability is important, rather than say the relative positions of the centres of mass of the objects, can be seen in Figure 2.1. In this case, the centres of mass of the two objects are far apart, both horizontally and vertically, and based on this property a relationship such as $\texttt{on}(\cdot,\cdot)$ could not be easily established.

To define this geometric separability, we identify the maximum margin hyperplane separating the pair of objects. This is the separating surface which has the maximum possible distance from points belonging to both objects, and as such runs through the centre of the regions where they are closest, being the regions of interaction between them. Finding this separator is efficiently solved as a quadratic programming problem, and is the same approach used by classification algorithms such as support vector machines (SVMs).

SVMs can compute this geometric separator efficiently, and so we use them as a tool in this sense, although we note that other randomised maximum margin algorithms could be used to perform the same task. The SVM is thus trained to identify elements in the point cloud which are critical for defining the object separation. These regions are known as contact points, and can be connected in a graph to give a contact point network. The details of extracting these networks are discussed in Section 2.3.4.

Relationships between objects are then identified by examining the displacements between the contact points of two different objects. This is elaborated in Section 2.3.5. Although this algorithm operates on only pairs of objects at a time, the extension to sets of objects is given in Section 2.3.6.

An illustrative example of the process is shown in Figure 2.1, with sample input and output of the algorithm.

2.3.2 Assumptions

A point cloud $P_M = \{p_i\} = \{(x_i, y_i, z_i, r_i, g_i, b_i)\} \subset \mathbb{R}^6$, $i = 1 \ldots M$, consists of M points, where each point i has three dimensions of spatial information (x_i, y_i, z_i), as well as three dimensions of colour information (r_i, g_i, b_i). We assume this point cloud has been captured by some device, such as stereo imaging or range scanning, but are indifferent to the hardware used for this purpose. What is required is that the pose of the imaging system is known. This work assumes the camera uses the same orientation as the scene, providing a known direction of "up". If this orientation is not known, a separate

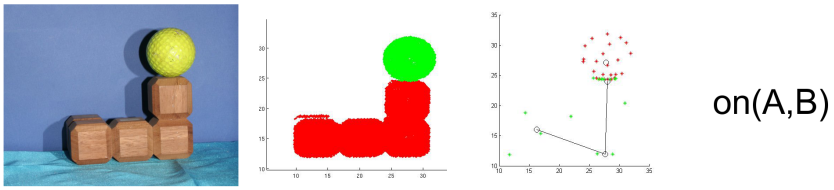

Figure 2.1: The relationship learning process. (a) The image captured by the left camera in a stereo camera pair. (b) The full point cloud, segmented into the two objects, after background subtraction. (c) The support vectors with the contact point networks of the two objects. The CPN of the ball is a single node, and the CPN for the L-shaped block is a three-node network. Note: in our experiments we used a user-defined threshold to discard contact points with less than 7 support vectors. This was not used in this image, to illustrate the concept of the network, but if it had, only the topmost contact point in the block would have remained. (d) The inferred relationship. A and B are arbitrary symbolic labels given to the two point clouds corresponding to the two objects.

routine may be required to estimate this.

We assume that the background has already been segmented from the data, leaving a point cloud $P_N \subseteq P_M$, with $N \leq M$. P_N is thus the subset of P_M, where every point belongs to one of the objects in the scene.

Finally, we require that the various objects in the scene be segmented. This is the process of assigning a single class identity c_i to each point in the cloud, depending on the object to which that point belongs. This has been the subject of much previous work, using a number of different techniques based on various properties of the point clouds (e.g. Jiang et al. (2000)), and so will not be discussed here in depth. However, as it is an important part of the preprocessing of our algorithm, one simple procedure is given in Section 2.3.3. This procedure determines class identity as a function of point colour: $c_i = f(r_i, g_i, b_i)$, and is sufficient for our purposes, although it is somewhat unsophisticated.

A potential difficulty would arise if the objects are incorrectly segmented. However, as we discuss further in Section 2.5, it may be reasonable for segmentation to fail in this way, as object identities in a static image may not even be correctly determined by a human, without the ability to perturb the scene. Nevertheless, the algorithm returns a potential interpretation of the scene that could actually be useful, for example to seed a motion plan.

The colour information is not required by our algorithm, and so is discarded. Hence

the input to our algorithm is the combination of spatial information and object identities of each point in the cloud: $\tilde{P}_N = \{(x_i, y_i, z_i, c_i)\}$, for $i = 1 \ldots N$. The final assumption we make is that this scene contains only two objects, i.e. $\forall i, c_i \in \{0, 1\}$. This simplifies the demonstrations in this chapter, but is not a strong assumption. This assumption is relaxed in Section 2.3.6, by considering objects in the scene in a pairwise manner.

2.3.3 Object Segmentation

We present one simple method for object segmentation, based on colour information in the point cloud. This is possible if the objects in the scene are of different colours. This strong assumption could be weakened by using a different segmentation method, relying on factors such as discontinuities in the curvature of surfaces fit to the point clouds, or assumptions based on the connectivity of regions of an object.

We normalise the RGB values of each point to reduce the effects of specularities, discretise the RGB space using a three-dimensional colour histogram, and then cluster the bins using the k-means algorithm. Setting $k = 3$ will generate three clusters, where the largest two clusters correspond to the two objects being segmented, and the third cluster absorbs remaining pixels from the background and shadows which may not have been correctly segmented out by the background subtraction process. Note that a larger value of k could be used if the number of objects was unknown, and any clusters of size below some threshold could be discarded as noise. Misclassified points are cleaned up by means of the k-nearest neighbours algorithm.

This process assigns each pixel p_i in P_N an identity $c_i = \{0, 1\}$, as belonging to one of the two clusters each corresponding to an object, resulting in the segmented and labelled point cloud $\tilde{P}_N$.

2.3.4 Extracting a Contact Point Network

The abstracted representation of the objects in a scene is useful as a concise description of the scene, and for reasoning about the structure of the environment. Furthermore, this representation also aids in classifying the relationships between the objects, as shown in Section 2.3.5. This representation is known as a contact point network for each object. We now describe the process for constructing these networks.

Given the labelled point cloud $\tilde{P}_N = \{p_i\} = \{(x_i, y_i, z_i, c_i)\}$, divide this into two distinct point clouds representing the two objects depending on the class identities c_i, such that object $O^j = \{p_i | c_i = j\}$ with $j = \{0, 1\}$, giving that $\tilde{P}_N = O^0 \cup O^1$. We do not

require a completely clean segmentation of the point clouds into the two objects, and the boundary may be noisy and slightly misaligned from the actual object boundaries, provided the noise does not qualitatively change the relationship between the objects.

The first step is to identify the contact regions between the two objects. This is done by training a support vector machine (SVM) to classify the two (already separated) objects, by defining a decision boundary between the classes which maximises the margin, being the smallest distance between any of the training points and the decision boundary (Bishop, 2006).

A conventional use of an SVM as a classifier might use features of the scene to classify whether or not a given test scene represents a particular predicate. Instead, we are using the property that an SVM also efficiently computes the geometric separator between two point sets, by classifying points as belonging to one of these two sets, which in this case are objects. The support vectors are thus in our case the features which are extracted from the data by the SVM, giving the contact points as markers of the geometric separators.

The training data provided is the combined dataset $O^0 \cup O^1$, which is labelled by object identities. As the data to be classified is in the form of two objects, it can be assumed that they are likely to form two regions (although each object may actually consist of multiple disconnected regions if it is obscured), with some unknown relationship between them. For this reason, we assume the objects are non-linearly separable, and so use a radial basis function (RBF) as the kernel in the SVM. An RBF is used as it is a nonlinear kernel with a localised response, and so is well suited to real-world objects, where the bulk of their mass is situated in closed regions with nonlinear boundaries.

The support vectors define the boundaries of the two objects. Let the support vectors for object O^i be $\mathbf{v}^i$. They are dense at any points where the two objects touch (or come near to touching) as these are regions where the boundary definition requires a high level of precision. Support vectors are also sparsely scattered around the silhouette of the object, as the contours of the decision boundary are defined with low precision in these regions. This is particularly true when the objects have a highly nonlinear boundary between them. As a result, regions of the object with a high concentration of support vectors are regions of the object at which the object is potentially in contact with another object. Furthermore, if the objects are far away from each other in some direction (relative to their sizes), very few if any support vectors are identified.

Clustering the support vectors $\mathbf{v}^i$ within an object provides a cluster at each (poten-

tial) point of contact between O^i and another object, with additional clusters of outlier points from the boundary of O^i. The centroids of these clusters are known as the set of contact points $\{\chi_k^i\}$, $k = 1 \ldots K$ of the object. In practice, clusters are discarded if the number of support vectors in the cluster is below some user-defined threshold (a threshold of 7 was used in our experiments, but this parameter is somewhat dependent on the resolution of the data used).

The *contact point network* of an object O^i is defined as a graph CPN^i, where the nodes are the K contact points $\{\chi_k^i\}$, $k = 1 \ldots K$, and the edges $e(\chi_m^i, \chi_n^i)$, $m \neq n$ are the edges of the minimum weighted spanning tree (MWST) covering $\{\chi_k^i\}$, with the weights given by the Euclidean distances between the nodes.

An object's contact point network provides an abstracted representation of the object as a skeletal structure, based on the regions of the object where it comes into contact with another object in the scene. Although the contact points themselves convey information on the relative positioning of objects in a scene, the edges of the graph are useful firstly for identifying regions of an object which could be searched for grasp points, and secondly provide more structure to the representation. These graphs are useful for maintaining coherency of an individual object, as well as for determining when one composition of objects is a substructure of another. This is discussed further in Section 2.5. As a result, any of a number of algorithms could be used to construct these graphs, provided they are consistent across scenes.

2.3.5 Learning Relationships

Having extracted a contact point network from a pair of objects, we next establish the spatial relationship between these two objects. This relationship describes, in symbolic terms, the physical positioning of the one object relative to the other in the scene. We may wish to infer that "A is *on* B", or that "C is *under* as well as *adjacent* to D".

Given the contact point networks for two objects, CPN^i and CPN^j, consider all pairs of contact points (χ_m^i, χ_n^j). Let the displacement between these contact points be $d_{mn} = \chi_m^i - \chi_n^j$. This gives the oriented displacements of the two regions of the objects corresponding to those contact points. By definition, if the objects touch along some interface, then both objects must have a representative contact point for that region. Those contact points will then have a small separation displacement d_{mn}, with the distance and orientation defining the spatial relationship between the objects. The problem of classifying this spatial relationship has then been reduced to the problem

of classifying $d_{mn} \in \mathbb{R}^3$. Although these displacement vectors may not be sufficient to capture every spatial relationship that exists in an environment as a result of, for example, large occlusions, the use of these contact points and the displacements between them provides us with a useful representation for abstracting a scene.

Taking a supervised learning approach to this classification requires a set of training data, that is, a set of point cloud images of pairs of objects, where the relationships between the objects have been provided. This input data is thus a set of triples, consisting of object pairs and relations, $\{(O^i, O^j, r^{ij})\}$, where r^{ij} is a symbolic label for a relationship that exists between the two objects. By the same procedure described above, the separation displacements can be extracted from the training images, and then labelled by the provided relationships. Call this set of labelled displacement vectors from the training data $\{(d_{ij}, r^{ij})\} = \mathcal{D}$.

To then classify the new separation displacement d_{query} a method such as k-nearest neighbours can be used to assign a label to d_{query} based on the most commonly occurring labels of the closest training points to d_{query} from $\mathcal{D}$. These labels are selected as the result of a voting scheme from the labels of the nearest neighbours. The variance of the labels provided by the nearest neighbours can then be used as a measure of confidence: the system is more confident of a solution if every neighbour proposes the same particular label, than if only 50% of them proposed that label. This acts as a quality measure of the output of the system in the form of predicted relationships.

Consider a training image with a pair of objects, O^0 and O^1, where CPN^0 has M^0 contact points and CPN^1 has M^1 contact points. There will be a total of M^0M^1 separation displacements for the object pair, $\{d_{mn}\}$, $m = 1 \ldots M^0$, $n = 1 \ldots M^1$. Each of these will receive the same relationship label r^{01}, as this label is provided at the level of objects, rather than regions. There is however probably only a single separation displacement which conveys this information accurately describing the spatial relationship. As a result, $M^0M^1 - 1$ spurious separation displacements will also carry the label r^{01}, even though they do not describe that particular relationship. These could enable the erroneous classification of a new object pair through d_{query}.

To overcome this issue, consider a threshold $\omega \in \mathbb{R}$. A labelled separation distance d_{mn} extracted from the training data would only be added to $\mathcal{D}$ if $|d_{mn}| \leq \omega$. This prevents the incorporation of displacements which are far from each other in the scene, although this threshold is dependent on the data. An alternative method for overcoming this problem is to weight the contributions of the k-nearest neighbours in classifying a label for d_{query}, by the inverse of the distance from d_{query}. This separation threshold is

a context dependent, and therefore tunable, parameter.

A concise description of the entire spatial relationships and abstraction extraction algorithm is given in Algorithm 1.

Algorithm 1 The spatial relationships and topology extraction algorithm

Require: A 3D point cloud with pre-segmented background P, and the labelled training data $\mathcal{D}$

1: $[O^0, O^1] \longleftarrow$ `segment-objects`(P)
2: svm $\longleftarrow$ `train-svm`(O^0, O^1)
3: $[\mathbf{v}^0, \mathbf{v}^1] \longleftarrow$ `extract-support-vectors`(svm)
4: **for** each object k **do**
5: $\quad \chi^k \longleftarrow$ `cluster`$(\mathbf{v}^k)$
6: $\quad \mathbf{e}^k \longleftarrow$ `minimum-weighted-spanning-tree`(χ^k)
7: $\quad \text{CPN}^k \longleftarrow (\chi^k, \mathbf{e}^k)$
8: **end for**
9: `rel` $\longleftarrow \emptyset$
10: **for** (χ^i, χ^j), $\chi^i \in \text{CPN}^0$, $\chi^j \in \text{CPN}^1$ **do**
11: $\quad d_{ij} \longleftarrow \chi^i - \chi^j$
12: $\quad \mathbf{d} \longleftarrow$ `nearest-neighbours`$(d_{ij}, \mathcal{D})$
13: $\quad r \longleftarrow$ `voted-common-labels`$(\mathbf{d})$
14: $\quad$ `rel` $\longleftarrow$ `rel` $\cup r$
15: **end for**
16: **return** a contact point network for both objects in P, CPN^0 and CPN^1, as well as the symbolic relationship `rel` between the objects

As an alternative to taking a supervised approach to classifying the relationships, they could instead be learned in an unsupervised manner. A similar approach of the unsupervised learning of relationships from data is seen in work such as that of Galata et al. (2002). Under this approach, labels are not provided for the training data. Instead, a set of separation displacements $\mathcal{D}$ are clustered, using an algorithm such as x-means (Pelleg and Moore, 2000) to determine the number of clusters as well as the cluster locations. This approach does not suffer from the problem of spurious labelled data points. Each cluster is then interpreted as a unique binary relationship between two objects. These relationships can be assigned arbitrary names, as rel^k, $k = 1 \ldots K$ for K relations, such that $\text{rel}^k : O \times O \rightarrow \{\texttt{T}, \texttt{F}\}$.

The primary advantage of an unsupervised approach, apart from removing the need

to generate training labels, is that the system would acquire those relationships present in the data, meaning that different relationships could be discovered to those expected by people (with their own prior knowledge of space). On the other hand, the system may not as easily be able to disambiguate between similar, yet different, configurations (consider that "on" and "adjacent" could appear very similar if one object rests against another at an angle).

2.3.6 Extensions to Multiple Objects

The procedure outlined in the previous section describes ways in which objects can be abstracted into their contact point network skeletons, and further that the spatial relationships between the objects can be determined from these networks. This was all done by considering only a pair of objects at a time. Considering larger groups of objects is a straightforward extension.

For a set of objects in a scene, the algorithm is extended to generate contact point networks by considering all pairs of objects in the scene. The question is then how to merge the networks generated for a particular object when considered together with other objects. The solution is to collect the set of all contact points from every different network representation of a single object, and then build a new network from these points.

Let $\mathrm{CPN}^{i|j}$ denote a contact point network of object O^i extracted by the algorithm when considered together with object O^j. Now $\mathrm{CPN}^{i|j} \neq \mathrm{CPN}^{i|k}$, for $O^j \neq O^k$. This is because the two other objects cannot interface with O^i in the exact same way, and hence give rise to different skeleton representations of O^i.

Merge $\mathrm{CPN}^{i|j}$ and $\mathrm{CPN}^{i|k}$ to give $\mathrm{CPN}^{i|(j\cup k)}$ by combining the two sets of contact points. Formally, let $\mathrm{CPN}^{i|j}$ have $M^{i|j}$ contact points $\{\chi_m^{i|j}\}$, $m = 1 \ldots M^{i|j}$ and $\mathrm{CPN}^{i|k}$ have $M^{i|k}$ contact points $\{\chi_n^{i|k}\}$, $n = 1 \ldots M^{i|k}$. When merged, $\mathrm{CPN}^{i|(j\cup k)}$ then has $M^{i|(j\cup k)} = M^{i|j} + M^{i|k}$ contact points $\{\chi_p^{i|(j\cup k)}\}$, $p = 1 \ldots M^{i|(j\cup k)}$. The edges of this merged contact point network are created by constructing a minimum weighted spanning tree over the combined set of nodes, with the weights of the edges in the tree determined by Euclidean distances between the nodes.

A simple illustrative example of combining two contact point networks for a single object is shown in Figure 2.2. This demonstrates how a scene consisting of three objects will be processed by the algorithm. The scene is described as: a ball A rests on a block B, which in turn rests on the ground G. Processing objects A and B gives

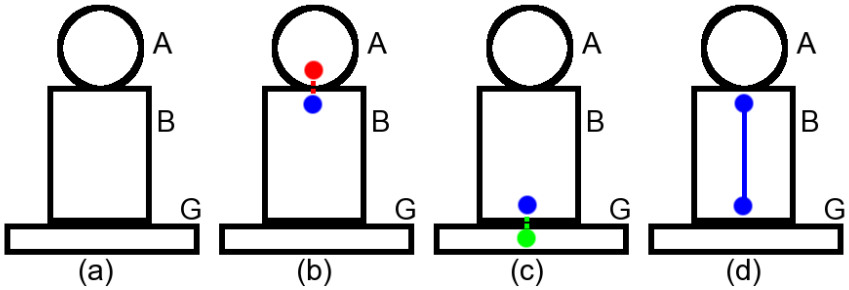

Figure 2.2: A simple illustrative example of combining contact point networks. Filled circles represent contact points. Solid lines are parts of the skeleton of an object, and dashed lines show relationships between two objects. (a) The base configuration of the scene. (b) Processing of objects A and B. (c) Processing of objects B and G. (d) Combining the information extracted from these two steps.

one contact point in each object, which is enough to infer $\texttt{on}(A,B)$. Similarly, processing objects B and G gives one contact point in each object, which is enough to infer $\texttt{on}(B,G)$. Combining the information extracted from these two steps, gives a symbolic description of the scene as $\{\texttt{on}(A,B),\texttt{on}(B,G)\}$, as well as a two-node topological description of B. This represents how the object B is used in this particular scene, rather than describing any distinctive properties of B itself.

2.4 Experiments

2.4.1 Classifying Relationships

In order to validate our algorithm, we present the following demonstration: the algorithm is run on the segmented point clouds generated from a set of stereo images captured in our lab. Firstly, we show that given a set of training data, the relationships between two new objects can be correctly classified by our algorithm. Secondly, we show that the contact point networks extracted from the objects convey interesting topological structure, which may aid manipulation. Finally, we demonstrate the effect of merging contact point networks, as described in Section 2.3.6.

For our experiments we used the following hardware configuration to acquire our images: a stereo capture rig was constructed of two 8.0 MPixel Canon EOS 350D cameras, calibrated and using the maximum level of magnification supported by the standard EF-S lens (0.28m closest focus distance). Dense registered range and colour data was collected using a Dimensional Imaging system. Intensity images have a spa-

tial resolution of 3456×2304 pixels, inter-pixel spacing is 25pixel/mm. RMS depth error is around 0.03mm. Although the resolution used in these images was high, the results appeared fairly robust when randomly discarding up to 70% of data points.

For the first experiment, we gathered a set of 128 stereo images, each consisting of a pair of objects. The objects varied greatly in shape, colour and texture, with the selected objects including a golf ball, blocks of various dimensions, a stone, a cloth, a piece of aluminium foil, and several plastic and furry childrens' toys. The range of objects used demonstrate that the algorithm is not particular to planar surfaces or convex volumes. In each case a blue material backdrop and table cover was used. This allowed for automated mask generation for background segmentation by the Dimensional Imaging software, as a part of the process of registering the stereo image pair from the two cameras and converting the output into a three-dimensional point cloud. A selection of the images which were photographed and used is shown in Figure 2.3.

This experiment aimed at examining two core example relations: $\texttt{adjacent}(\cdot,\cdot)$ and $\texttt{on}(\cdot,\cdot)$. The adjacent relation is considered in this case to mean that two objects are visually next to each other in the static scene, i.e. an object A is either to the left or the right of another object B, and so $\texttt{adjacent}(A,B) = \texttt{left-of}(A,B) \cup \texttt{right-of}(A,B)$. For each image I^j with objects O^{j0} and O^{j1}, a label $r^j \in \{0,1\}^2$ was prepared, where:

- $r^j(0) = 1$ if (O^{j0} on O^{j1})
- $r^j(0) = 0$ if (O^{j0} not on O^{j1})
- $r^j(1) = 1$ if (O^{j0} adjacent to O^{j1})
- $r^j(1) = 0$ if (O^{j0} not adjacent to O^{j1})

Representing the relations as a vector of labels allows for cases where one object is both on and adjacent to the other object (possibly at different places). Using this label representation scheme, there are thus four ways in which one object can relate to another: on, adjacent, on and adjacent, none. In practice, all of the images we used in our experiments had one of these four relationships existing between the two objects. Note that for any image where the distance between the two objects is great, there would be no contact points in either object, and as a result no displacement vectors would be generated.

The implementation of the algorithm was in MATLAB, using the external *SVM-light* implementation of Joachims (1999). In practice, this implementation was sufficient to process an entire image in well under a second on a dualcore desktop computer.

Figure 2.3: A subset of 12 of the object configurations used. A total of 128 such images were used in our experiments. In all cases the images shown are those which were photographed by the left camera of the stereo system. Some of the images are darker with less contrast as these photographs were taken in a second session with different lighting conditions.

Each image in the dataset was run through the algorithm, and the separation displacements d_{mn} were extracted. A plot of these displacements is shown in Figure 2.4. These carry the labels assigned to the spatial relationships between the objects in the images. Several features are clearly visible in this data. Firstly, there is a defined cluster describing the $\texttt{on}(\cdot,\cdot)$ relationship, centered at $(0,-0.5)$. Secondly, there is a broad band of points above $y=-0.1$ which corresponds to the $\texttt{adjacent}(\cdot,\cdot)$ relation. Note that this incorporates both $\texttt{left-of}(\cdot,\cdot)$ and $\texttt{right-of}(\cdot,\cdot)$, represented respectively by those displacements with a negative or a positive horizontal component. Object pairs which exhibit both $\texttt{on}$ and $\texttt{adjacent}$ at different points should contribute displacement vectors to both classes.

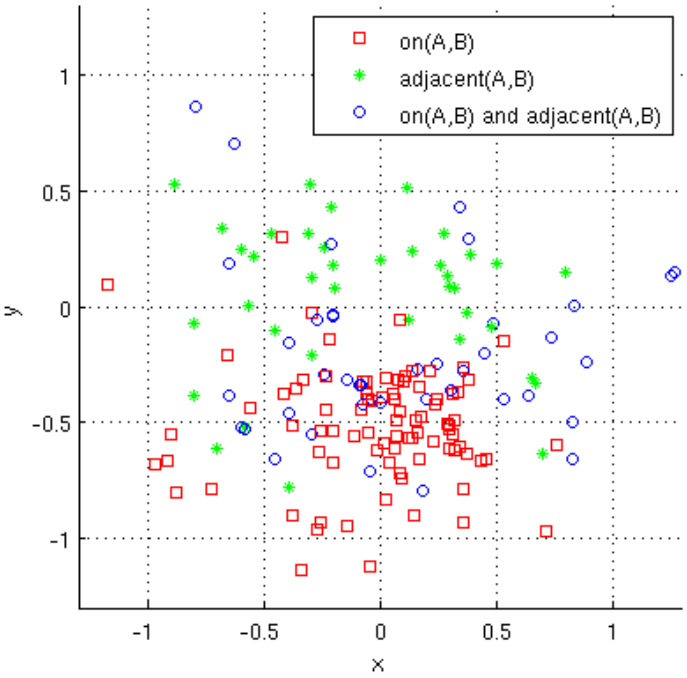

Figure 2.4: A plot of the x-y projection of the 3D separation displacements d_{mn} for the 128 images of object configurations used as training data. Each point corresponds to a vector starting at the origin and ending at that point, and is labelled according to the relationship exhibited in the corresponding image, being `on`$(\cdot,\cdot)$, `adjacent`$(\cdot,\cdot)$ or `on-and-adjacent`$(\cdot,\cdot)$.

We tested the classification ability of this method by using each image of the training set as a testing image, in a leave-one-out cross-validation test. The procedure used was as follows: for each image I^j, we removed that image from the dataset. Using the remaining training data, we classified the relationships between the objects in I^j by considering the nearest neighbours (the number of neighbours considered in our experiments was 4, but this parameter could be changed) to the separation displacements from that image. For the cases where there are two relationships between the objects in the image, indicated by the presence of multiple separation displacements, we considered the union of the predicted values. So if one separation displacement was classified as `on` and another as `adjacent`, the description of the relationship between the objects was (`on` and `adjacent`). Note that the label $r^j \in \{0,1\}^2$ in each case. As each test image was selected from the set of training images, we compared the predicted results to the true results. This is shown in Figure 2.5.

As may be expected, the method suffered most at the interface between the two

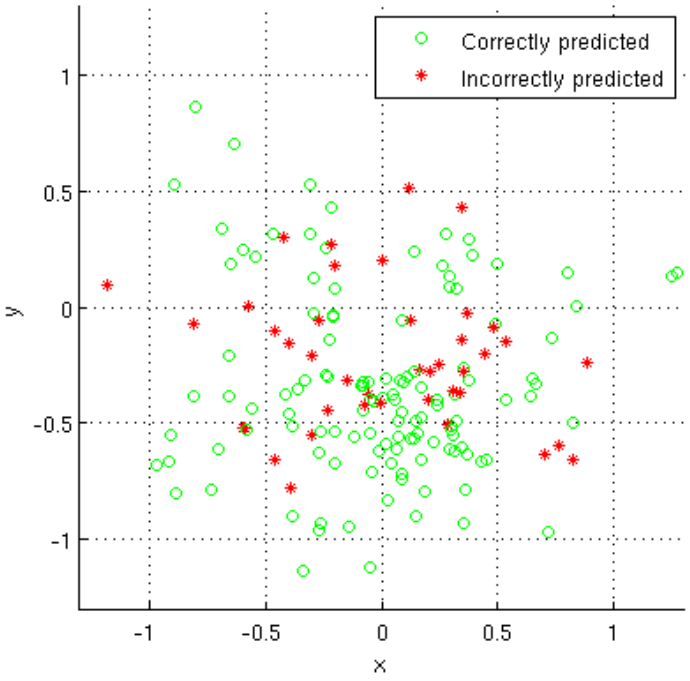

Figure 2.5: A comparison of the true and predicted object relationships

clusters. This is indicative of the fact that humans often struggle to cleanly differentiate between these relations. For example, two objects leaning against each other could be described as either on or adjacent. The confusion matrix of the predicted and true labels is:

		Actual			
		on	adjacent	both	none
Predicted	on	64	6	13	0
	adjacent	2	20	5	0
	both	4	5	12	0
	none	0	1	0	0

As can be seen, the cases with which the algorithm had the most difficulties were those when both relationships were present. These were often misclassified as being `on`. We can explain this problem by considering two cases for when one object may be both on and adjacent to another. The first is if these two relationships occur at disjoint regions in the object. This will give rise to a pair of separation displacements, one at the interface described as `on` and the other at the `adjacent` regions. On the other hand, one region in an object may be both on and adjacent to the same region in another object. In this case, the separation displacement would not be classified as

either relationship, but rather an intermediate one.

The input data set of 128 images was then randomly divided into a training set of 95 training images, and 33 testing images, corresponding to 116 training relationships and 49 testing relationships respectively, as each object pair may contact multiple contact points. Again using k-nearest neighbours, with $k = 4$, it was found that 40 of the 49 test relationships had their `on` component correctly determined, and 39 of the 49 test relationships had their `adjacent` component predicted correctly. Using the naïve confidence measure of the percentage of the nearest neighbours with labels agreeing with the predicted values, 28 and 18 of the respective predicted relationships agreed with 75% of their nearest neighbours from the training set.

2.4.2 Contact Point Networks

In addition to classifying the relationships between objects, the algorithm abstracts the pair of objects into two contact point networks. The advantage of this is that these networks are representations of the objects in terms of how they interact with other objects in the scene. These skeletons allow an agent interacting with the objects to uncover the topological structure present in the individual objects, as well as the entire scene, which could then be used by the agent to guide further interaction.

As an example of this principle, consider the two scenes in Figure 2.6. Both scenes consist of an object resting on an L-shaped wooden block, but these objects (two wooden blocks connected by a rotary joint, and a plastic toy goat) are very different. However, in both scenes a hole is formed as a result of the interaction between the two objects. This is clearly visible in the contact point networks of the objects in the scene. In both cases, the algorithm has discovered a topological hole in the environment. This suggests that if, for example, an agent has a motion strategy to pick up the top object from underneath for the first scene, it could use a similar motion strategy as a candidate for performing this action in the second scene.

This demonstrates the ability of the algorithm to abstract complicated objects and shapes down into the fundamental topological structure which arises from the way in which these objects are used in this particular scene. Having extracted these structures, there is a large body of literature which addresses the topologies of graphs (see for example Cook and Holder (2007) and Carlsson (2009)), for operations such as detecting commonly occurring sub-structure.

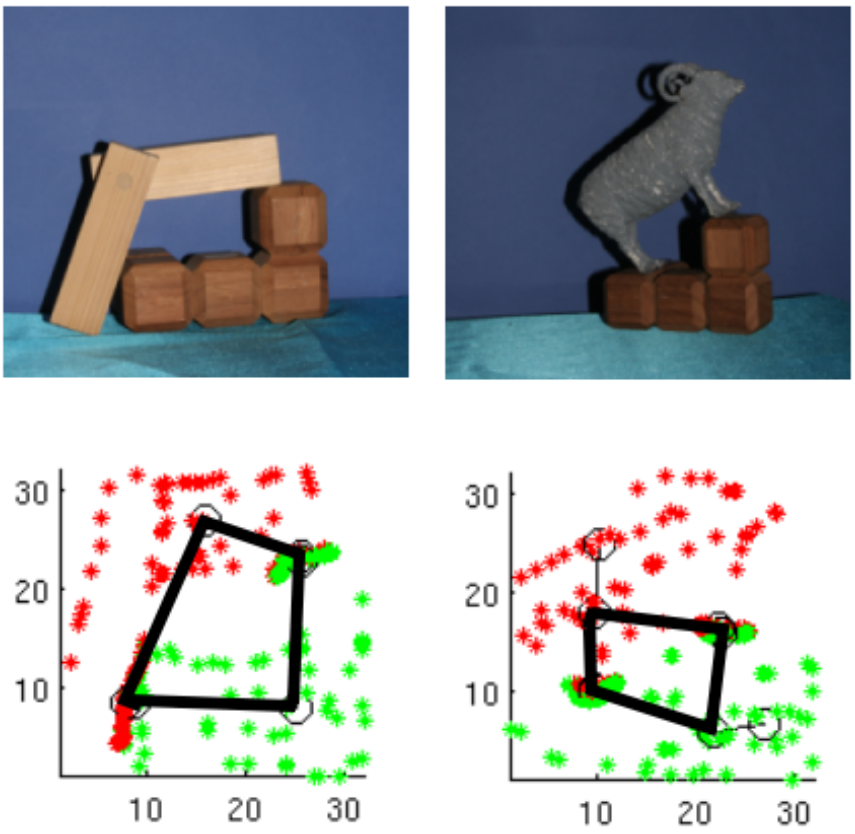

Figure 2.6: Using contact point networks to discover topological structure in two scenes. The algorithmic output is shown below each scene, where the points are the support vectors for each object, and the open circles are the contact points. The common topological feature – the hole – is emphasised.

2.4.3 Multiple Objects

Section 2.3.6 describes a method for extending the algorithm to cases where there are more than two objects in the scene, which is to be expected from any real environment. This extension involves examining the objects in a pairwise manner, and then joining the contact point networks of the same object, provided by its interfaces to various other objects in the scene.

Two examples of scenes with multiple objects are shown in Figure 2.7. The first instance shows three objects stacked above each other, with the result that instead of each skeleton being a single point, these two points are merged into one skeleton of two nodes for the central object. The second instance actually represents a failed segmentation, where two objects are incorrectly segmented into three. The same skeleton structure is observed in these two instances, showing that each object in the first scene

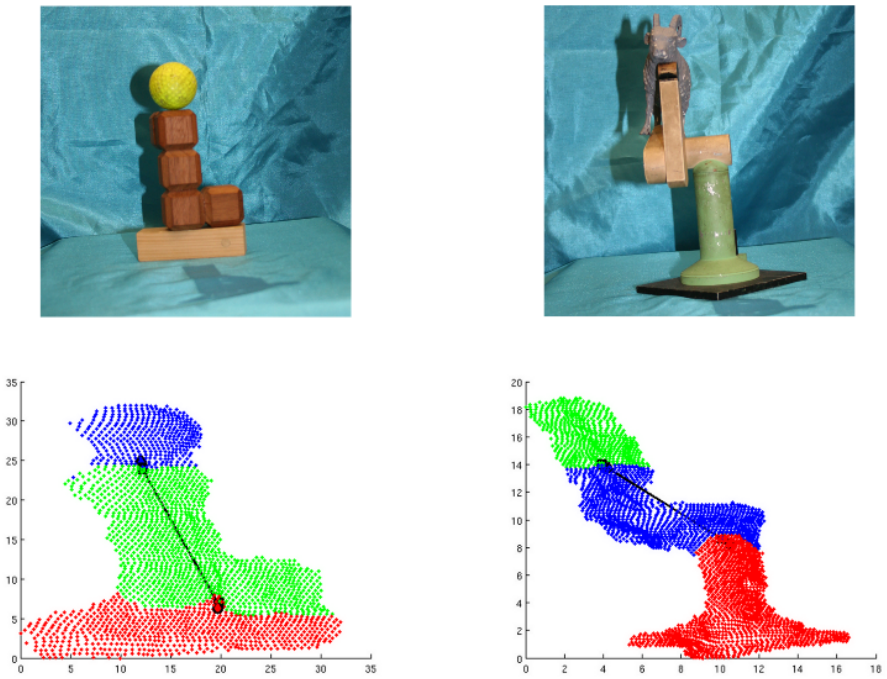

Figure 2.7: Examples of processing more than two objects in a scene. The point clouds are shown, with the skeletons of the objects.

has an equivalent object in the second scene, in terms of the structural use of that object in the scene. Note that even with the failed segmentation, a viable interpretation of the structure of the scene is extracted.

Clearly, in cases such as those in Figure 2.7, the objects as well as the contact points that are detected by the algorithm are only candidates, and may not actually correctly reflect the scene's structure. These could perhaps be disambiguated by motions, such as moving components of the scene and observing which objects move consistently together. Nonetheless, the structural abstraction in both cases mirrors the essence of the scene.

2.5 Discussion

The algorithm presented in Section 2.3 generates two different representations of a scene. Firstly, the spatial relationships between the different objects in the scene are extracted as symbolic predicates, and secondly the topological structure of the objects in the scene is constructed, as determined by these inter-object interactions.

The symbolic relationships between objects define the structure of an environment,

and as a result, learning these relationships and being able to identify them in a static scene has important ramifications for planning and learning the effects of actions, as done in the work of Pasula et al. (2007). All of this builds towards a better understanding of the overall structure and behaviour of the components of an environment, as these relationships provide a means for describing changes in the relative positioning of objects.

Similarly, these contact point networks are important for understanding the capabilities of an individual object, or set of objects, and provide insight into the topological structure of the environment. This enables reasoning and policy reuse at a coarser, more abstract level, which is important in practice. This is in contrast to the approach of first identifying an object from a database of known objects, as an agent may not need to know exactly what an object is in order to use it.

This algorithm generates a layered representation of a scene, shown in Figure 2.8. At the lowest level is the point cloud, which consists of the most information, and is the direct output of the perception system of the robot acting in that environment. This level is useful for predicting collisions between the robot and object, and other low level control functions.

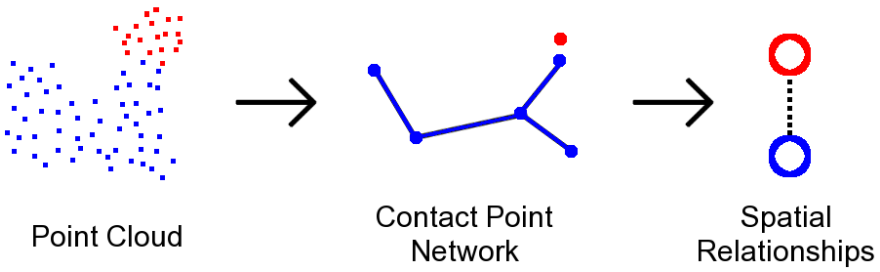

Figure 2.8: An illustration of the layered scene representation

The next level is the contact point network, which provides a manipulation robot with a set of candidate points and edges for interacting with the objects, as well as knowledge of similarities in structure of different parts of the scene. Finally, the relational predicates provide the robot with symbolic knowledge of the inter-object relationships, which can be easily used in plan formulation.

There are many useful mechanisms for reasoning about the relationships between objects and parts of objects, such as the spatial calculus of Randell et al. (1992). Logics such as this do not describe how the elementary concepts would arise from data, and our work attempts to bridge that divide by providing a semantic interpretation of a scene in symbolic terms. Our algorithm provides a mechanism for identifying some

of these relationships in a scene, and thus serves to ground this spatial logic calculus, thereby allowing the calculus to be utilised by a robot.

Reasoning can also be done at the level of the topological structures observed in the contact point networks in a scene. An example of this can be seen in the work of Calabar and Santos (2011). Our algorithm again bridges the gap, and would allow for the real-world versions of the spatial puzzles solved in this work, by identifying holes and other structures in the topology of the puzzles. We are not only interested in determining valid grasp points (Saxena et al., 2008), but are interested in the structure of a scene as a whole.

In addition to each layer providing an agent with different means for reasoning about a scene, another advantage to this layered representation is an increased robustness to segmentation failures. The most likely cause of failure of this algorithm is a result of its dependence on a relatively clean segmentation of the objects in the scene. The method is robust to small noise-related errors in the segmentation, but is prone to difficulties if, say, two parts of the same object are classified as being different objects, or conversely if two separate objects are classified as being the same object. However, as shown in Section 2.4.3, a plausible explanation for the structure of the scene will still be generated. In fact, given that any objects may be fixed together in any scene by glue or some other medium, only the incorporation of actions to perturb the scene such as poking and prodding would lead to a guaranteed correct object segmentation.

For the first of these cases, the segmentation of one object into multiple parts, the algorithm will still return the correct relationship between those parts, even if they do not correspond to whole objects. For a static scene, this is still a plausible description of the scene, even if not the most likely. A simple example of this is shown in Figure 2.9, where even though an object is incorrectly segmented, the relationships between these regions are still correctly identified. Using a similar methodology to the interactive perception proposed by Katz and Brock (2008), a robot manipulating in that environment may discover that even when subjected to various forces, the relationships between these parts remain constant. Using this observation, the parts could be conceptually merged into a single object. This remains the subject of future work.

Similarly, if two objects have not been separated and are instead considered as a single object, this is also a possible description of a single scene. When these objects are acted on, the relationship between them is likely to change, and thus the second object could be detected. In fact, it often happens that two objects only become distinguishable to a human under action, such as two adjacent papers on a desk, or a

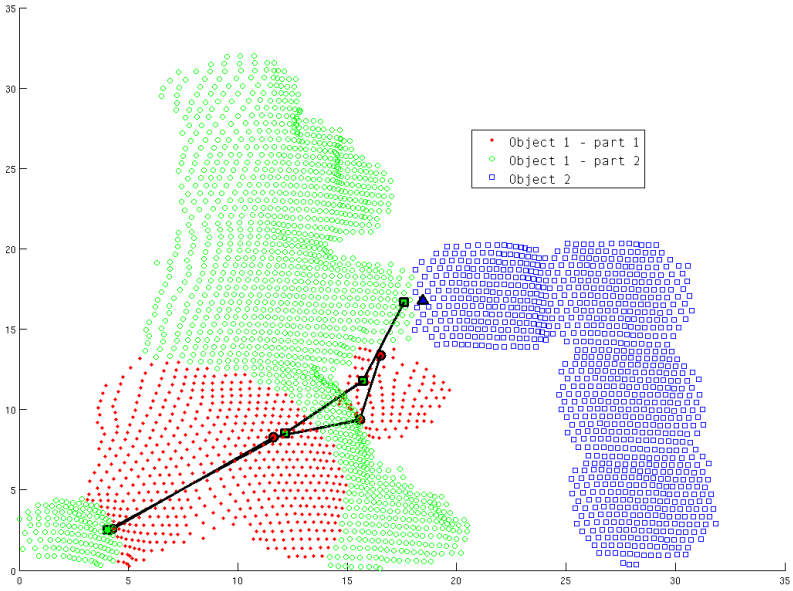

Figure 2.9: Relationships between incorrectly segmented objects. The object on the left was incorrectly subdivided into two different objects.

stick-insect in a tree.

This algorithm is somewhat related to the idea of bootstrap learning, as expounded by Kuipers et al. (2006). This paradigm aims at developing a set of methods whereby agents can learn common sense knowledge of the world, from its own observations and interactions with that world. There has already been much success in agents learning reliable primitive actions and sensor interpretations, as well as recognising places and objects. Our algorithm provides that same agent with a mechanism for learning about the different spatial relationships between those objects.

As an example of this, consider Figure 2.10. This shows two different scenes, each consisting of three objects stacked on each other. While stacking in the first scene involves each object having only one point of contact with the object below it, the second scene has two contact points between the topmost object and the one below it. The abstract skeletal structure of the first scene in the form of its contact point network can be seen here to be a subgraph of the skeleton of the second scene.

As a result of the first scene being a subgraph of the second, an agent operating in the space of these scenes can seed its behaviour in the case of the second scene with the behaviours it used in the first scene, e.g. grasp points. Furthermore, the additional components of this new structure over the previous one afford the agent new regions

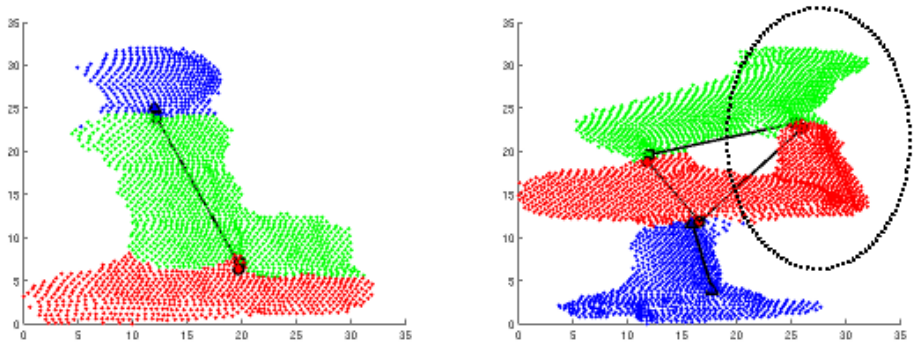

Figure 2.10: Example of one structure as a substructure contained in another. Without the inclusion of the objects under the dotted ellipse, the structure of the second image is topologically equivalent to that of the first. The skeleton of the first image is thus a subgraph of the second.

on the pile of objects to manipulate through exploratory actions.

2.6 Conclusion

In order for a robot to be able to manipulate objects in a meaningful way within an environment containing spatially interacting objects, it must possess knowledge of how the different objects in that environment are used in that environment, as well as how they relate to one another. The set of spatial relationships between objects is the glue which holds a scene together, and allows for differentiation between a set of items, and a structured environment.

We propose an algorithm for learning a layered representation of an environment, including a structural abstraction as well as these spatial relationships, and being able to classify them in either a supervised or an unsupervised manner. The algorithm finds regions of contact between pairs of objects, known as contact points, by locating areas where there is a small margin of geometric separability between the objects. Constructing a graph from these contact points results in a contact point network which is a topological description of the role of that object in the current scene. These contact point networks provide the agent with the ability to abstract objects into simpler topological structures. Inter-object relationships are classified based on the separation between contact points from the skeletons of two different objects.

The networks and the relationships provide a layered representation of a scene consisting of multiple objects. This representation facilitates reasoning about the objects

at different levels, through the use of planning mechanisms, as well as searching for common structure in different environments to seed behaviours and thus significantly reduce the search space for motion synthesis. This could enable a robot to perform comprehensive tasks in a manipulation environment.

Chapter 3

Action Priors for Domain Reduction

Parts of the work presented in this chapter also appear in: Rosman and Ramamoorthy (2012a), Rosman and Ramamoorthy (2012b), and Rosman and Ramamoorthy (2014).

3.1 Introduction

Consider some learning agent, such as a robot, operating in a building for a prolonged period of time. This robot is required to perform multiple tasks in this domain, for example couriering different items to different people around the building, monitoring access to certain areas, etc. The difficulty with this setting is that not only are these tasks varied, but their specifications are also not completely known *a priori*, e.g. goal states may differ between tasks.

It is clear that this robot would need to learn to perform a number of different tasks. However, we wish to accelerate this learning via transferring knowledge. When approaching a new task, the robot should be able to reuse knowledge from having experienced other tasks. The challenge here is how to organise this experience so as not just to store vast numbers of behaviours which are tailor-learnt to specific tasks. Instead, we desire some mechanism for generalisation.

Although the agent may be learning different tasks, the fact that they exist within the same environment means there is some underlying structure common to all of them. It is this structure we wish to exploit to facilitate faster learning through forming better abstractions of the domain. Hence, we are interested in building agents capable of learning domain invariances. Invariances in this sense act as a form of common sense knowledge of the domain: these are elements which are generally common to a large class of behaviours. This form of knowledge can provide insights into learning what

not to do in particular situations, e.g. learning to avoid behaviours in a simulator which may not be realisable on a real system (Koos et al., 2013). In this way, we are interested in a form of model learning, where the model consists of the commonalities between a set of tasks, rather than the reward structure for any individual task.

Learning new behaviours necessarily requires extensive exploration of the space of possibilities. This is particularly the case when the specification of the task (given by the exact reward structure) is difficult to obtain, such as in the case of delayed reward reinforcement learning. We want to address this problem through the observation that different behaviours have commonalities at a local level. Our goal is thus to be able to inject weak knowledge into the problem, which is a prior of sensible behaviours in the domain, and can be learnt autonomously from previous tasks. To this end we introduce the notion of *action priors*, which are local distributions over the action sets of a learning agent, that can be used to bias learning of new behaviours.

Learning action priors equates to finding invariances in the domain, across policies. The invariances in which we are interested are the aspects of the domain that the agent treats in the same way, regardless of the task. For instance, when one is driving a car the "rules of the road", techniques for driving the car, and interaction protocols with other vehicles remain unchanged regardless of the destination. Learning these domain invariances is useful in a lifelong sense, as it factors out those elements which remain unchanged across task specifications. We thus regard this as a form of multitask learning (Thrun, 1996a; Caruana, 1997) where the agent is required to learn to generalise knowledge gained from solving some tasks, in discovering their built-in commonalities, to apply to others.

The key assumption we are leveraging in this work is that there is certain structure in a domain in which an agent is required to perform multiple tasks over a long period of time. This is in the form of many local situations in which, regardless of the task, certain actions may be commonly selected, while others should always be avoided as they are either detrimental, or at best do not contribute towards completing any task. This induces a form of local sparsity in the action selection process. By learning this structure throughout a domain, when posed with a new task an agent can focus exploratory behaviour away from actions which are seldom useful in the current situation, and so boost performance in expectation. We note that extracting this structure gives us weak constraints in the same way as manifold learning (Havoutis and Ramamoorthy, 2013).

Action priors allow a decision making agent to bias exploratory behaviour based

on actions that have been useful in the past in similar situations, but different tasks. This formalism additionally provides a platform which can be used to inject external information into the agent, in the form of teaching.

We pose this framework within a reinforcement learning context, but note that it is applicable to other decision making paradigms. Indeed, we argue in Section 3.5 that this mechanism resembles techniques which humans are believed to invoke in order to facilitate decision making under large sets of options (Simon and Chase, 1973).

3.1.1 Contributions

The main problem we address in this chapter is that an agent learning to perform a wide range of tasks in the same domain is essentially relearning everything about the domain for every new task, and so learning is slow. We thus seek a middle ground between model-based and model-free learning, where a model of the regularities in behaviours within a domain can be acquired.

Our approach to tackling this problem is to extract invariances from the set of tasks that the agent has already solved in the domain. The specific invariances we use are termed action priors, and they provide the agent with a form of prior knowledge which can be injected into the learning process for new tasks.

Additionally, we show

- that this approach corresponds to minimising, and thus simplifying, the domain,
- an alternative, but weaker, version of the action priors which is suitable for changing domains and partial observability,
- a method for selecting domain features so as to maximise the effect of these priors,
- that action priors are based on mechanisms which are cognitively plausible in humans,
- a different application of action priors, where they are used to provide advice to other agents.

3.1.2 Chapter Structure

This chapter is structured as follows. We introduce our core innovation, action priors, in Section 3.2. We then discuss how action priors can be used in scenarios where the

structure of the domain changes, and use this reformulation to perform feature selection in Section 3.3. We demonstrate our methods in experiments in Section 3.4, discuss the relation of our approach to psychological findings in human behaviour in Section 3.5, and present additional related work in Section 3.6. Finally, in Section 3.7 we provide an application of action priors being used to advise other agents navigating around a common environment.

3.2 Action Priors

3.2.1 Preliminaries

In keeping with the standard formalism of reinforcement learning, we assume that an environment is specified by a Markov Decision Process (MDP). An MDP is defined as a tuple (S,A,T,R,γ), where S is a finite set of states, A is a finite set of actions which can be taken by the agent, $T : S \times A \times S \to [0,1]$ is the state transition function where $T(s,a,s')$ gives the probability of transitioning from state s to state s' after taking action a, $R : S \times A \times S \to \mathbb{R}$ is the reward function, where $R(s,a)$ is the reward received by the agent when transitioning from state s with action a, and $\gamma \in [0,1)$ is a discount factor. As T is a probability function, $\sum_{s' \in S} T(s,a,s') = 1, \forall a \in A, \forall s \in S$.

A Markovian policy $\pi : S \times A \to [0,1]$ for an MDP is a mapping from states to actions. The return, generated from an episode of running the policy π is the accumulated discounted reward $\bar{R}^{\pi} = \sum_k \gamma^k r_k$, for r_k being the reward received at step k. The goal of a reinforcement learning agent is to learn an optimal policy $\pi^* = \arg\max_{\pi} \bar{R}^{\pi}$ which maximises the total expected return of an MDP, where typically T and R are unknown.

Many approaches to learning an optimal policy involve learning the value function $Q^{\pi}(s,a)$, giving the expected return from selecting action a in state s and thereafter following the policy π (Sutton and Barto, 1998). Q is typically learnt by iteratively updating values using the rewards obtained by simulating trajectories through the state space, e.g. Q-learning (Watkins and Dayan, 1992). A greedy policy can be obtained from a value function defined over $S \times A$, by selecting the action with the highest value for the given state.

3.2.2 State Based Action Priors

We define a domain by the tuple $D = (S,A,T,\gamma)$, and a task as the MDP $\tau = (D,R,S_0)$. In this way we factorise the environment such that the state set, action set and transition

functions are fixed for the whole domain, and each task varies only in the reward function, and the set of initial states $S_0 \subseteq S$. Given an arbitrary set of tasks $\mathcal{T} = \{\tau\}$ and their corresponding optimal policies $\Pi = \{\pi^*_\tau\}$, we wish to learn for each state $s \in S$ a distribution $\theta_s(A)$ over the action set, representing the probability of each action in A being used in an optimal policy in the state s, aggregated over tasks. This is then, in subsequent tasks, used to prioritise the actions in each state.

Basing these preference distributions on state information equates to a fully observable problem, wherein the action prior depends on the complete state description. We relax this and discuss the more general setting of conditioning the action priors on some subset of observational features in Section 3.3.

The idea here is that by considering a set of policies, each of which selects actions in the same state s according to different distributions, one can gain insights into properties of s. These insights are in terms of the "usefulness" of different actions in that state. For example, if one action is favoured by all policies when in s, then that action could be considered as very useful in s, and should be favoured during exploration. Conversely, if an action is not selected by any policy in s, then that action is likely to have negative consequences, and should be avoided. This notion thus informs weak constraints on the policy at s, in that an action not selected by previous policies should not be prioritised in solving future tasks.

To provide further intuition into how this distribution represents action "usefulness", we say that an action a_1 is preferable to an action a_2 in a state s, if a_1 is used in s by a larger number of optimal policies than a_2. The setting in which we study the phenomenon of accelerated learning in a lifelong sense is that the tasks seen so far are sampled from a distribution of all possible tasks, and are representative of that task space. By studying the optimal policies that arise from multiple tasks in the same domain, we aim to learn about the structure of the underlying domain, in terms of identifying local behaviours which are invariant across tasks.

3.2.2.1 Combining Policies to Learn Priors

Consider the setting in which the agent has prolonged experience in the domain D. This means the agent has had to solve a set of tasks in D, and we use the resulting set of optimal policies to extract the action priors as a form of structural information about the domain.

For each state $s \in S$ in the domain, we model the action priors $\theta_s(a)$, $\forall a \in A$ as a distribution over the action set A, describing the usefulness of each action in optimally

solving the tasks in $\mathcal{T}$ using the policies Π. To do so, we first define the utility of an action a in a state s under a policy π as

$$U_s^{\pi}(a) = \delta(\pi(s,a), \max_{a' \in A} \pi(s,a')), \tag{3.1}$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta function: $\delta(a,b) = 1$ if $a = b$, and $\delta(a,b) = 0$ otherwise. As a result $U_s^{\pi}(a) = 1$ if and only if a is the best (or tied best) action in s under π. The reason that we use this formulation of a utility function rather than $Q(s,a)$, is that the values stored in $Q(s,a)$ relate to the rewards allocated for the task for which π is a solution. These values are thus relative, unlike those of U_s^{π} which provide a point-estimate of the value of an action in a state, and as such are comparable across policies for very different tasks with different reward scales.

Now consider the utility $U_s^{\Pi}(a)$ of an action a in a state s under a policy randomly drawn from a *policy library* Π. This value is a weighted sum, given as

$$U_s^{\Pi}(a) = \sum_{\pi \in \Pi} w(\pi) U_s^{\pi}(a), \tag{3.2}$$

where $w(\pi) \in \mathbb{R}$ is a weight over the policy π. We can use this weight either to measure our confidence in the optimality of the policy π when the policies are provided by different agents, or can indicate a prior probability over the policies if they are not drawn uniformly. The inclusion of this weight factor allows us to include suboptimal policies in the formulation, by giving them lower weights. These weights could be normalised to reflect probabilities, but they need not be, as the process for converting these utilities to action priors accounts for normalisation of these terms (see discussion below). In our experiments we do not assume to have knowledge about this distribution, and so uniformly set $w(\pi) = 1, \forall \pi \in \Pi$.

The fact that $\theta_s(a)$ is constructed from the policies solving a set of tasks $\mathcal{T}$ opens up the possibility that $\mathcal{T}$ is not actually representative of the complete set of possible tasks in D. We counteract this by forming an augmented policy set $\hat{\Pi}$, defined as $\hat{\Pi} = \Pi \cup \pi_0$, where π_0 is the uniform policy: $\pi_0(s,a) = \frac{1}{\|A\|}, \forall s \in S, a \in A$. The utility of this policy is then $U_s^{\pi_0}(a) = 1, \forall s \in S, a \in A$. In this case, the weight $w(\pi_0)$ is representative of the likelihood of encountering a new task.

Given a state s, for each action a the utility of that action $U_s^{\hat{\Pi}}(a)$ provides an estimate of the value of the state-action pair (s,a) in D under the augmented policy set. We thus choose actions according to these values. To select an action, we sample from a probability distribution $\theta_s(A) = f(U_s^{\hat{\Pi}}(A))$, such that $a \sim \theta_s(A)$. There are many ways in which f may be defined, e.g.

1. as a proportion: $f(U_s^{\hat{\Pi}}(A)) = f_p = \frac{U_s^{\hat{\Pi}}(a)}{\sum_{a' \in A} U_s^{\hat{\Pi}}(a')}$,

2. as a Boltzmann softmax distribution: $f(U_s^{\hat{\Pi}}(A)) = f_s = \frac{\exp\{U_s^{\hat{\Pi}}(a)\}}{\sum_{a' \in A} \exp\{U_s^{\hat{\Pi}}(a')\}}$,

3. as a draw from a Dirichlet distribution: $f(U_s^{\hat{\Pi}}(A)) = f_d \sim Dir(U_s^{\hat{\Pi}}(a))$.

Throughout the remainder of this chapter, we choose to model f as a Dirichlet distribution, although these all remain valid choices. We base this choice on the fact that the mean of the Dirichlet distribution is its normalised parameters which, as we show in Section 3.2.2.2, is exactly the proportion f_p.

We prefer both f_p and f_d to the softmax f_s as we hypothesise that in situations where a large number of prior policies have been encountered, small discrepancies in action counts could explode under the exponential term, and as a result some viable options would be explored considerably less than the prior advocates. Additionally, any 'max' function is unlikely to provide the same exploration convergence guarantees. We provide empirical validation for this choice in Figure 3.15 in Section 3.4.7.

3.2.2.2 Action Priors as a Dirichlet Distribution

For each state $s \in S$, we choose to draw the action priors $\theta_s(a), \forall a \in A$ from a Dirichlet distribution conditioned on s. The Dirichlet distribution is parametrised by concentration parameters $(\alpha(a_1), \alpha(a_2), \ldots, \alpha(a_{\|A\|}))^T$ and so for each state s, we maintain a count $\alpha_s(a)$ for each action $a \in A$. The initial values of $\alpha_s(a) = \alpha_s^0(a)$ are known as the pseudocounts, and can be initialised to any value by the system designer to reflect prior knowledge. If these counts are the same for each action in a state, i.e. $\alpha_s(a) = k, \forall a \in A$ this returns a uniform prior, which results in each action being equally favourable in the absence of further information.

The pseudocounts $\alpha_s^0(a)$ are a hyperprior which models prior knowledge of the tasks being performed by the agent. If the variance in the tasks is expected to be small, or alternatively a large number of training tasks are provided, then this hyperprior is set to a smaller value. However, if there is great diversity in the tasks, and the agent will not be expected to sample them thoroughly, then a larger hyperprior will prevent the action priors from over-generalising from too little data.

We wish these counts to describe the number of times an action a was considered

optimal in a state s, across a set of policies Π. We thus set

$$\alpha_s(a) = U_s^{\hat{\Pi}}(a) = \sum_{\pi \in \hat{\Pi}} w(\pi) U_s^{\pi}(a) \tag{3.3}$$

$$= \sum_{\pi \in \Pi} w(\pi) U_s^{\pi}(a) + \alpha_s^0(a). \tag{3.4}$$

This provides a natural intuition for the counts as the weighted utility of Π, and the hyperprior is then $\alpha_s^0(a) = w(\pi_0)$.

Typically, one would not want to maintain a full library of policies. As a result, the α counts can alternatively be learnt by the agent in an online manner as it learns the solutions to new tasks. In this way, when the agent solves some task τ^{t+1}, the counts for each state-action pair can be updated by the values in π^{t+1}. This provides an online version of Equation (3.3) as

$$\alpha_s^{t+1}(a) \longleftarrow \begin{cases} \alpha_s^t(a) + w(\pi^{t+1}) & \text{if } \pi^{t+1}(s,a) = \max_{a' \in A} \pi^{t+1}(s,a') \\ \alpha_s^t(a) & \text{otherwise.} \end{cases} \tag{3.5}$$

To obtain the action priors $\theta_s(a)$, sample from the Dirichlet distribution: $\theta_s(a) \sim Dir(\alpha_s)$. Note that $\theta_s(a)$ is sampled as a probability distribution over A, and so $\sum_a \theta_s(a) = 1$, $\forall s \in S$.

This process could be viewed as a form of averaging the policies. However, naïve averaging is known to be problematic, and often gives detrimental results. Reward functions may, in principle, differ by orders of magnitude, and an averaged policy may not even be feasible. For example, given a state s in front of an obstacle, policy π_1 may suggest moving around the obstacle to the left, while policy π_2 indicates movement to the right. Averaging suggests moving forward, straight into the obstacle. We instead infer that both moving left and moving right are feasible choices to be later explored, whereas moving forward is never the correct choice. The action priors should consequently place more weight on 'left' and 'right' than on 'forward', reflecting the preferences elicited from π_1 and π_2. This knowledge is accumulated from the experience of different tasks.

Note that in Equation (3.5) we increment the α counts by $w(\pi^{t+1})$, rather than the probability stored in $\pi^{t+1}(s,a)$. This is because we are aiming to enumerate the actions used by the different policies, rather than simply averaging the individual policies. This form of averaging could result in the agent being drawn towards a local optimum in state space, by one policy dominating others, and subsequently being unable to escape from it.

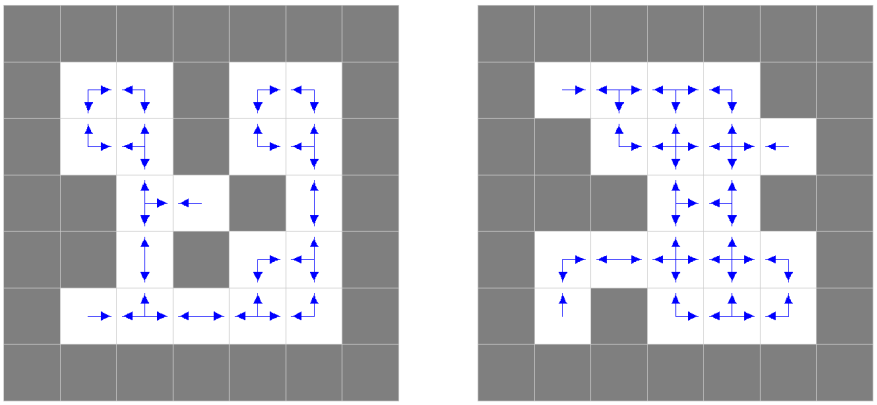

Figure 3.1: Example action priors learned for 7×7 maze worlds, from 50 random optimal Q-functions. The indicated directions in each cell are those with a non-negligible probability mass, but in every cell the agent has the choice of executing any of four directional movements. Grey cells are obstacles, and white cells are free space.

Instead we weight each action by the number of independent tasks which require the selection of that particular action, which is then used as a prior probability of that action choice in that state over all tasks in the domain.

To illustrate instances of action priors learned from optimal policies, Figure 3.1 demonstrates the result of using our method on 7×7 maze worlds, extracted from policies which were the optimal solutions to 50 random navigation tasks. An arrow in a cell is drawn in a direction only if any mass was allocated to that direction by any policy. Note that this results in the "useful" actions of the domain, being the actions that do not cause collisions with obstacles. The use of action priors effectively reduces the set of actions from four in each cell to the subset which were useful in the training tasks (55.26% and 63.16% of the full action sets respectively in the examples shown in Figure 3.1).

3.2.2.3 Using the Action Priors

An action prior provides the agent with knowledge about which actions are sensible in situations in which the agent has several choices to explore. As a result, they are useful for seeding search in a policy learning process. We demonstrate this modified exploration process with an adaptation of traditional Q-learning (Sutton and Barto, 1998), called ε-greedy Q-learning with State-based Action Priors (ε-QSAP, first introduced by Rosman and Ramamoorthy (2012b)), which is shown in Algorithm 2. Note, in this algorithm, $\alpha^Q \in [0, 1]$ denotes the learning rate, and should not be confused with the

Dirichlet distribution counts $\alpha_s(a)$. The parameter $\varepsilon \in [0,1]$ controls the trade-off between exploration and exploitation. Both α^Q and ε are typically annealed after each episode.

Algorithm 2 ε-greedy Q-learning with State-based Action Priors (ε-QSAP)

Require: action prior $\theta_s(a)$

1: Initialise $Q(s,a)$ arbitrarily
2: **for** every episode $k = 1 \ldots K$ **do**
3: Choose initial state s
4: **repeat**
5: $a \longleftarrow \begin{cases} \arg\max_a Q(s,a), & \text{w.p. } 1-\varepsilon \\ a \in \mathcal{A}, & \text{w.p. } \varepsilon\theta_s(a) \end{cases}$
6: Take action a, observe r, s'
7: $Q(s,a) \longleftarrow Q(s,a) + \alpha^Q[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
8: $s \longleftarrow s'$
9: **until** s is terminal
10: **end for**
11: **return** $Q(s,a)$

The difference between this and standard ε-greedy Q-learning can be seen on line 5. This is the action selection step, consisting of two cases. The first case deals with exploiting the current policy stored in $Q(s,a)$ with probability $1-\varepsilon$, and the second case with exploring other actions $a \in A$ with probability ε. The exploration case is typically handled by choosing the action uniformly from A, but instead we choose with probability based on the prior $\theta_s(a)$ to shape the action selection based on what were sensible choices in the past.

The effect is that the agent exploits the current estimate of the optimal policy with high probability, but also explores, and does so with each action proportional to the number of times that action was favoured in previous tasks. This highlights the assumption that there is inherent structure in the domain which can be identified across multiple tasks.

Exploration thus occurs by randomly sampling actions according to the probabilities that they are optimal, given the previously encountered tasks. Choosing actions randomly in this way is an established action selection mechanism (Wyatt, 1997; Dimitrakakis, 2006) known as Thompson sampling (Thompson, 1933) which, rather than being a heuristic, has been shown to be a principled approach to dealing with uncer-

tainty (Ortega and Braun, 2013). As these probabilities relate to the domain, rather than the current task, this is still traded off against exploiting the current Q-function.

3.2.3 Action Priors as Domain Reduction

We now show that extracting action priors from a set of tasks is equivalent to reducing the transition system of the domain. To do this, we first define futile states and actions in an MDP, as being the undesirable states and actions which do not further progress towards any goal.

Definition 3.2.1 (Task Futile Action)**.** For a domain $D = (S, A, T, \gamma)$, a task $\tau_i = (D, R, S_0)$ and the corresponding optimal policy π_i, let the *task futile actions* $A_{\mathcal{B}_i} \subset S \times A$ be the set of state-action pairs which are not selected as optimal under π_i. Therefore, $A_{\mathcal{B}_i} = \{(s,a) | \pi_i(s,a) \neq \max_{a' \in A} \pi_i(s,a')\}$.

Definition 3.2.2 (Task Futile State)**.** For a domain $D = (S, A, T, \gamma)$, a task $\tau_i = (D, R, S_0)$ and the corresponding optimal policy π_i, let the *task futile states* $S_{\mathcal{B}_i} \subset S$ be the set of states which, under π_i, appear on no paths starting in S_0. These states are computed as $S_{\mathcal{B}_i} = S \setminus S_{R_i}$, where S_{R_i} are the states reachable in the domain under policy π_i, as computed in Algorithm 3.

Algorithm 3 Computing reachable states

Require: domain $D = (S, A, T, \gamma)$, a task $\tau_i = (D, R, S_0)$ and a policy π_i

1: $S_{R_i} \longleftarrow S_0$
2: **repeat**
3: $\quad S^{\text{old}} \longleftarrow S_{R_i}$
4: $\quad S_{R_i} \longleftarrow S_{R_i} \cup \{\hat{s} \in S | T(S_{R_i}, \pi_i(S_{R_i}), \hat{s}) > 0\}$
5: **until** $S^{\text{old}} = S_{R_i}$
6: **return** S_{R_i}

The concept of a futile state encompasses all states which are harmful to the agent (e.g. trap states), as well as any states which are otherwise non-progressive towards the goal of the agent, or are completely unreachable. Similarly, a futile action is one which an optimal policy for that task would avoid taking, possibly owing to not being optimal for that task. We now define the futile actions (states) of a domain as the intersection of those for all tasks in the domain.

Definition 3.2.3 (Domain Futile Action)**.** For a domain $D = (S, A, T, \gamma)$, and a set of tasks $\mathcal{T} = \{\tau_i\}$ with corresponding task futile actions $A_{\mathcal{B}_i} \subset S \times A$, let the *domain futile actions* be $A_{\mathcal{B}} = \bigcap_i A_{\mathcal{B}_i}$.

Definition 3.2.4 (Domain Futile State)**.** For a domain $D = (S, A, T, \gamma)$, and a set of tasks $\mathcal{T} = \{\tau_i\}$ with corresponding task futile states $S_{\mathcal{B}_i} \subset S$, let the *domain futile states* be $S_{\mathcal{B}} = \bigcap_i S_{\mathcal{B}_i}$.

Given a domain D and a set of tasks $\mathcal{T}$, the domain futile states $S_{\mathcal{B}}$ and domain futile actions $A_{\mathcal{B}}$ are those which are consistently avoided by all optimal policies which complete the tasks in $\mathcal{T}$. We can then use these to define a task-set minimised domain.

Definition 3.2.5 (Task-Set Minimised Domain)**.** For a domain $D = (S, A, T, \gamma)$, a set of tasks $\mathcal{T} = \{\tau_i\}$ with corresponding domain futile states $S_{\mathcal{B}}$ and domain futile actions $A_{\mathcal{B}}$, the task-set minimised domain under task set $\mathcal{T}$ is given by $D_{\mathcal{T}} = (\hat{S} = S \setminus S_{\mathcal{B}}, A, \hat{T}, \gamma)$, with

$$\hat{T}(s,a,s') \longleftarrow \begin{cases} \eta(s,a)T(s,a,s') & \text{if } s \in \hat{S}, (s,a) \in \hat{A}, s' \in \hat{S} \\ 0 & \text{otherwise,} \end{cases}$$

where $\hat{A} = (S, A) \setminus A_{\mathcal{B}}$, and $\eta(s,a) = \frac{1}{\sum_{s' \in S} T(s,a,s')}$, $\forall s \in \hat{S}, \forall a \in \hat{A}$ indicates normalisation of T, such that $\sum_{s' \in S} \hat{T}(s,a,s') = 1$, $\forall s \in \hat{S}, \forall a \in \hat{A}$.

The task-set minimised domain contains all states and actions from the original domain which are non-futile. As a result, values in the state transition function transitioning from a futile state s, or using a futile action a are excluded. Additionally, transitions from a non-futile state s with a non-futile action a into a futile state s', are not permitted by the definition of futile states.

Theorem 3.2.6 (Properties of a Task-Set Minimised Domain)**.** *If a set of tasks $\mathcal{T}$ in the domain D are used to construct $D_{\mathcal{T}}$, then the following properties of $D_{\mathcal{T}}$ hold:*

1. *The optimal reward achievable for each task in $\mathcal{T}$ can still be obtained in $D_{\mathcal{T}}$.*
2. *Each optimal policy π_i used to achieve a task $\tau_i \in \mathcal{T}$ in the domain D is optimal for that same task τ_i in $D_{\mathcal{T}}$.*

Proof. 1. The goals $G \subseteq S$ of task τ_i, are the states which by the definition of τ_i must be visited by the optimal policy π_i in order for it to be optimal. Therefore, $G \subseteq S_{R_i}$ by definition, and so G does not contain task futile states, and so is in the reduced MDP $D_{\mathcal{T}}$ by construction.

2. A policy π_i has a zero probability of invoking a futile action, making these actions unnecessary, by construction. All states and actions reachable by this policy must, by construction, be included in $D_{\mathcal{T}}$. As optimal policy π_i transitions only through reachable states by definition, and all those states are contained in $D_{\mathcal{T}}$, so must π_i be valid in $D_{\mathcal{T}}$.

□

Proposition 3.2.7. Exploration will be safer in $D_{\mathcal{T}}$ than in D, as $D_{\mathcal{T}}$ contains fewer trap states and dead ends than D.

Proof. Futile actions and states which are common to all tasks are excluded from the reduced MDP $D_{\mathcal{T}}$ by construction. This implies that fewer trap states will be accessed during exploration, and those that remain are all reachable states, and thus on the path to a goal state, for at least one task. Any states that remain must lie on an optimal path. □

Proposition 3.2.8. For an unknown task drawn from $\mathcal{T}$, a learning agent will be expected to learn that task no slower in $D_{\mathcal{T}}$ than in D, and possibly faster.

Proof. The number of state action pairs in the reduced MDP $|(S,A)| \geq |\hat{A}|$, which implies there are no more, but possibly fewer, possible states and actions to be explored by the learning agent. With the resulting decrease in the size of the search space, learning is necessarily faster. □

These properties hold if every task encountered at test time was experienced during training time, and so was incorporated into $D_{\mathcal{T}}$. In practice, this may not be the case, and we compensate for this through the addition of the pseudo-counts α^0. Assume there is a complete set of tasks $\mathcal{T}$, such that a subset $\mathcal{T}_s \sim \mathcal{T}$ have been sampled by the agent during training. Assume also that for some new task, there is a goal state which is futile given the tasks in $\mathcal{T}_s$. In the worst case, there is a single action in a single state which will transition into this new goal state. Thus, transitions from a state in $\hat{s} \in \hat{S}$ with a single action leading to a state in $S_{\mathcal{B}}$ will be made with probability

$$P(\hat{S} \longrightarrow S_{\mathcal{B}}) = \frac{\alpha^0}{(|A|-1)(|\mathcal{T}_s|+\alpha^0)+\alpha^0} \tag{3.6}$$

in this worst case.

The effect of the probability in Equation (3.6) is that when no prior tasks have been sampled (i.e. $|\mathcal{T}_s| = 0$), this transition from $\hat{s}$ will be explored with probability $1/|A|$,

as expected. As the number of tasks experienced for which $S_{\mathcal{B}}$ is futile increases, the probability of this transition decreases proportionally to $1/|\mathcal{T}_s|$. As a result, we consider that there is a distribution over all tasks, which has been sampled to give the current action prior. α_0 must therefore reflect this sampling, and is chosen based on the variability and diversity in tasks in the domain.

As a result, exploring using action priors is a form of safe exploration, which is achieved in task $N+1$ by biasing exploration towards behaviours which have successfully been used in N related tasks in the same domain. The key assumption here is that interaction with unsafe states and actions is consistent across all tasks, such that states which are unsafe given one set of tasks cannot constitute goal states for others. Unsafe states are the subset of $S_{\mathcal{B}}$ and unsafe actions the subset of $A_{\mathcal{B}}$ which cause a large negative reward, or even damage to the agent. This effect of safe exploration is evident in the results presented in Section 3.4, through the higher initial rewards indicating both an interaction with fewer negative reward states, and fewer failed episodes.

3.3 Priors from Observations

In Section 3.2.2, action priors were defined as distributions over actions, conditioned on the current state. In this section we extend these definitions such that the action priors are instead conditioned on observations.

There are several reasons for this representation change. The most obvious reason for this is that the transition function T or even the state space S may not be task independent, and may instead differ between task instances. This may be the case, for example, when an agent is tasked with exploring buildings with different layouts. It is not sensible to condition action priors on states, if the connectivity of those states changes between task instances. Instead, the agent should condition action priors on observable features of the states – features which would persist across tasks, even if state identities do not. This representation change allows the action priors to generalise to tasks in related environments.

Another justification for using observation based priors is that one may not always have full observability of s, meaning that different states cannot be uniquely distinguished. This is the case in partially observable reinforcement learning problems (Kaelbling et al., 1998) which typically require the solution of partially observable Markov decision processes (POMDPs). State information, such as exact world coordinates of some mobile robot, is not always accessible. Similarly, there may be states

in S which have not been explored during training time, and so no action prior would be available for these states, which could then be required at test time. In both these scenarios, it is again sensible to instead base the action priors on whatever features of the state are observed. The observation based action priors thus provide the ability to transfer to unseen state and action combinations.

Basing these priors on observations rather than states involves changing the dependence of θ from $s \in S$ to $\phi : S \longrightarrow O$, where ϕ is the mapping from state space S to the observation (or perception) space O. The observed features of s are thus described by $\phi(s)$. The state based priors can now be considered as a special case of observation based priors, with $\phi(s) = s$.

Note that we are not solving a partially observable problem, but are instead informing exploration based on some partial information signals. Using observations rather than exact state descriptions allows for more general priors, as the priors are applicable to different states emitting the same observations. This also enables pooling of the experience collected from different states with similar observations, to learn more accurate action priors. There is, however, a trade-off between this generality, and the usefulness of the priors.

This trade-off is a function of the observation features, and the amount of action information captured by these features. These, in turn, depend on properties of the tasks and environments with which the agent is interacting. The more general the observation features over which the action priors are defined, the less informative the action priors will be. On the other hand, the more specific these features are (up to exact state identification), the less portable they are to new states.

Both state and observation based action priors have their uses. For example, maze-like environments stand to benefit from a state based approach, where entire wings of the maze could be pruned as dead-ends, which is not possible based on observations alone. Alternatively, in a rich environment with repeated structure, it is less likely that the training policies will have sufficiently explored the entire space, and so it is reasonable to pool together priors from different states with the same observations.

3.3.1 Using the Observation Based Priors

Changing the variables on which the action priors are conditioned from states to observations involves replacing s with $\phi(s)$ in Algorithm 2. When learning the action priors, Equations (3.5) and (3.3) are also still valid, by again replacing s with $\phi(s)$.

For convenience, we refer to this variant of Algorithm 2 as ε-greedy Q-learning with Perception-based Action Priors (ε-QPAP), given in Algorithm 4.

Algorithm 4 ε-greedy Q-learning with Perception-based Action Priors (ε-QPAP)

Require: action prior $\theta_{\phi(s)}(A)$

Initialise $Q(s,a)$ arbitrarily
for every episode $k = 1 \dots K$ **do**
  Choose initial state s
  repeat
    $\phi(s) \longleftarrow$ observations(s)
    $a \longleftarrow \begin{cases} \arg\max_a Q(s,a), & \text{w.p. } 1-\varepsilon \\ a \in A, & \text{w.p. } \varepsilon\theta_{\phi(s)}(a) \end{cases}$
    Take action a, observe r, s'
    $Q(s,a) \longleftarrow Q(s,a) + \alpha^Q[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \longleftarrow s'$
  until s is terminal
end for
return $Q(s,a)$

Similarly to Equation (3.5), the α counts are learnt by the agent online from the previous optimal policies, and updated for each $(\phi(s),a)$ pair whenever a new policy π^{t+1} is available:

$$\alpha^{t+1}_{\phi(s)}(a) \longleftarrow \begin{cases} \alpha^t_{\phi(s)}(a) + w(\pi^{t+1}) & \text{if } \pi^{t+1}(s,a) = \max_{a' \in A} \pi^{t+1}(s,a') \\ \alpha^t_{\phi(s)}(a) & \text{otherwise.} \end{cases} \tag{3.7}$$

The interpretation is that $\alpha_{\phi(s)}(a)$ reflects the number of times a was considered a good choice of action in any state s with observations $\phi(s)$ in *any* policy, added to the pseudocount priors $\alpha^0_{\phi(s)}(a)$.

The corresponding closed form of Equation (3.7) given a set of policies Π is then:

$$\alpha_{\phi(s)}(a) = \sum_{s \in [s]_\phi} \sum_{\pi \in \Pi} w(\pi) U^\pi_{\phi(s)}(a) + \alpha^0_{\phi(s)}(a), \tag{3.8}$$

where $U^\pi_{\phi(s)}(a) = \delta(\pi(\phi(s),a), \max_{a' \in A} \pi(\phi(s),a'))$, and $[s]_\phi = \{s' \in S | \phi(s) = \phi(s')\}$ represents the equivalence class of all states with the same observation features as state s. This additional summation occurs because in the general case, the priors from multiple states will map to the same observation based action priors.

To obtain the action priors $\theta_{\phi(s)}(a)$, again sample from the Dirichlet distribution: $\theta_{\phi(s)}(a) \sim Dir(\alpha_{\phi(s)})$.

3.3.2 Feature Selection

The choice of the set of observational features is an open question, depending on the capabilities of the agent. Indeed, feature learning in general is an open and difficult question, which has been considered in many contexts, e.g. (Jong and Stone, 2005; Lang and Toussaint, 2009). The possible features include the state label (as discussed previously), as well as any sensory information the agent may receive from the environment. Furthermore, these features can include aspects of the task description, or recent rewards received from the environment.

As a result, in many domains, there could be a large set of observational features. The size of Φ, the space of possible mappings, is exponential in the number of features. We are interested in identifying the optimal feature set $\phi^* \in \Phi$, which provides abstraction and dimensionality reduction, with a minimal loss of information in the action prior. Finding such a ϕ^* allows for the decomposition of the domain into a set of *capabilities* (Rosman and Ramamoorthy, 2012a), being recognisable and repeated observational contexts, with minimal uncertainty in the optimal behavioural responses, over the full set of tasks.

Let a feature f_i be a mapping from the current state of the agent in a particular task, to a set of values $f_i^1 \cdots f_i^{K_i}$. We can now set ϕ to be a set of these features. We abuse notation slightly and for a particular feature set ϕ_i we enumerate the possible settings of all its constituent features, such that $\phi_i = \phi_i^j$ means that the features in ϕ_i are set to configuration j, where these configurations are uniquely ordered such that $j \in [1, K_{\phi_i}]$, where $K_{\phi_i} = \prod_q K_q$, q is an index into the features of ϕ_i, and K_q is the number of settings for feature q.

We wish to find a feature set which can prescribe actions with the most certainty. To this end, we now define the average entropy of an action a for a particular feature set ϕ as

$$\bar{H}_\phi(a) \quad = \quad \sum_{i=1}^{K_\phi} P(\phi^i) H[P(a|\phi^i)], \tag{3.9}$$

where $P(a|\phi^i) = \theta_{\phi^i}(a)$ is the value of the action prior for a particular set of feature values, $P(\phi^i)$ is the prior probability of that set of feature values, estimated empirically from the data as $\frac{\sum_a \alpha_{\phi^i}(a)}{\sum_i \sum_a \alpha_{\phi^i}(a)}$, and $H[p] = -p\log_2 p$ is the standard entropy. The prior

$P(\phi^i)$ serves to weight each component distribution by the probability of that feature combination arising in the data.

By summing the average entropy for each action, we define the entropy of the action set A for a feature set ϕ as

$$H_\phi = \sum_{a \in A} \bar{H}_\phi(a) \tag{3.10}$$

$$= -\sum_{a \in A} \sum_{i=1}^{K_\phi} \frac{\sum_{a' \in A} \alpha_{\phi^i}(a')}{\sum_{j=1}^{K_\phi} \sum_{a' \in A} \alpha_{\phi^j}(a')} \theta_{\phi^i}(a) \log_2 \theta_{\phi^i}(a). \tag{3.11}$$

The optimal feature set is that which minimises the action set entropy. In this way, what we seek is analogous to the information invariance which is present in the observations of the agent (Donald, 1995). There is however a caveat with this simple minimisation. The more features are included in the feature set, the sparser the number of examples for each configuration of feature values. We therefore regularise the minimisation, by optimising for smaller feature sets through the application of a penalty based on the number of included features. Finding the optimal feature set ϕ^* is thus posed as solving the optimisation problem

$$\phi^* = \arg\min_{\phi \in \Phi} [H_\phi + c\|\phi\|] \tag{3.12}$$

where $\|\phi\|$ is the number of features in ϕ, and c is a parameter which controls for the weight of the regularisation term.

The major problem with this computation is that the number of possible feature sets is exponential in the number of features. We therefore present an approximate method for selecting the best set of features. This is shown in Algorithm 5, which returns the approximate minimal feature mapping $\tilde{\phi}^* \simeq \phi^*$. The key assumption is that each feature f affects the entropy of $\theta_\phi(a)$ independently. For each feature f from the full feature set ϕ_{full}, we marginalise over that feature and compute the entropy of the remaining feature set. Each of these $\|\phi_{full}\|$ individual entropy values is compared to the entropy of the full set $H_{\phi_{full}}$. The greater the increase in entropy resulting from the removal of feature f, the more important f is as a distinguishing feature in the action prior, as the addition of that feature reduces the overall entropy of the action prior distribution. The feature set chosen is thus the set of all features which result in an entropy increase greater than some threshold ω.

The increased generality from the use of observations invokes a connection to active perception (e.g. Kenney et al. (2009)): certain behaviours only make sense in

Algorithm 5 Independent Feature Selection

Require: feature set ϕ_{full}, entropy increase threshold ω

1: Compute $H_{\phi_{full}}$ by Equation (3.11)
2: **for** every feature f in ϕ_{full} **do**
3: $\quad \phi_{-f} \longleftarrow \phi_{full} \setminus f$
4: $\quad$ Compute $H_{\phi_{-f}}$ by Equation (3.11)
5: $\quad \Delta_f \longleftarrow H_{\phi_{-f}} - H_{\phi_{full}}$
6: **end for**
7: $\tilde{\phi}^* \longleftarrow \{f \in \phi_{full} | \Delta_f \geq \omega\}$
8: **return** $\tilde{\phi}^*$

the context of particular sensory features. If there is a high entropy in which actions should be taken in the context of a particular observation, then there are two possible reasons for this: 1) it may be the case that different tasks use this context differently, and so without conditioning action selection on the current task, there is no clear bias on which action to select, or 2) it may be that this observation is not informative enough to make the decision, providing scope for feature learning. This distinction can only be made in comparison to using the maximal feature set, as the most informative set of observation features. Without ground truth state information, this can only be ascertained through learning. If the observations are not informative enough, then this suggests that additional features would be useful. This provides the agent with the opportunity for trying to acquire new features, in a manner reminiscent of intrinsic motivation (see, e.g. Oudeyer et al. (2007)).

3.3.3 Online Feature Selection

Algorithm 6 describes the complete process for performing feature selection on an online agent. The agent is repeatedly presented with a new task, solves that task, and uses the new policy to refine its feature set.

Given a new task, the algorithm executes four steps. First, a policy is learnt using the current action prior and Algorithm 4. This new policy is then used to update the full prior. From the full prior, select features using Algorithm 5. Finally, extract the action prior using the selected features by means of marginalising over the excluded features.

As the feature selection is done using Algorithm 5, this requires a choice of ω. This parameter is domain specific, but in our experiments we automate its selection as the

Algorithm 6 Online Feature Selection

Let ϕ_{full} be the full feature set
Initialise full action prior θ^0_{full}
$\theta^0 \longleftarrow \theta^0_{full}$
for every new task $t = 1, 2, \ldots$ **do**
    Learn policy π^t using prior θ^{t-1} and Algorithm 4
    Update θ^t_{full} using θ^{t-1}_{full} and π^t with Eq. (3.5)
    Select features ϕ^t from θ^t_{full} using Algorithm 5
    Extract θ^t from θ^t_{full}, marginalising over $f \in \phi_{full} \setminus \phi^t$
end for

mean of the set of Δ_f values, as computed in Algorithm 5.

3.4 Experiments

3.4.1 Maze Navigation

Spatial navigation is one setting in which we believe an agent stands to make significant gains by using action priors. Our first state-based experiment therefore takes place in a 23×23 cell maze where every second row and column is passable space, and the remaining cells are obstacles, creating a lattice. This domain is shown in Figure 3.2. Each task involves a random goal location which has to be reached by the agent, and each episode of a task initialises the agent in a random location in free space. The set of actions available to the agent is to move one cell North, South, East or West. The state available to the agent is the unique number of the cell the agent is occupying. Reaching the goal provides a reward of 100, and walking into a wall a reward of -10. Rewards are discounted with $\gamma = 0.95$.

The nature of this maze world is such that in most cells, only two or three of the available actions are beneficial (do not result in colliding with a wall). Typically, every action is tried with equal probability during learning. However, any optimal policy has already learned to avoid collisions, and it is this knowledge we transfer to a new learning instance.

An example of the perception-based action priors learned from optimal Q-functions is shown in Figure 3.1 in Section 3.2.2.2, although this figure demonstrates the result of using our methods on 7×7 maze worlds (smaller than our actual 23×23 experimental

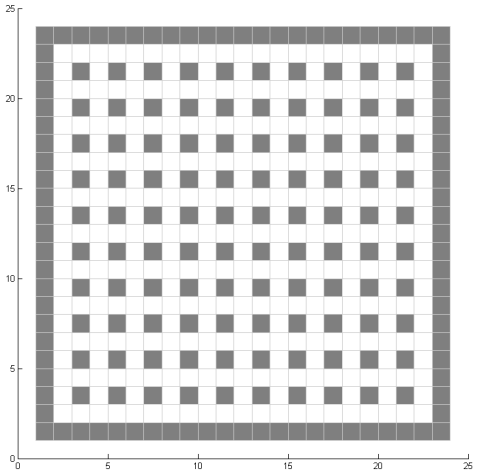

Figure 3.2: The lattice domain

worlds for ease of visualisation), extracted from Q-functions which were the optimal solutions to 50 random tasks.

The experimental procedure is as follows. We generate a set of tasks in the domain, and allow the agent to learn the corresponding optimal Q-functions. We then extract the action priors $\theta_s(a)$ from these, using the method described in Section 3.2.2, and finally provide the agent with a new set of tasks which use the priors for exploration as discussed in Section 3.2.2.3.

Figure 3.3 shows the improvement in convergence speed obtained through the addition of action priors. These results were averaged over learning of 10 different tasks, and this speed-up was achieved using the policies obtained from 10 training task optimal policies. One important observation is that the ε-QSAP algorithm immediately receives positive return, unlike Q-learning, as it has learnt to avoid harmful actions. The results do not include the training times for ε-QSAP, which can be considered a once-off overhead which need not be repeated for any future task. Alternatively, if the priors are updated with each new task instance then ε-QSAP starts with no advantage, but after 10 tasks achieves the performance benefits shown.

Note that the Q-learning baseline is equivalent to ε-QSAP with a uniform action prior. Additionally, the fact that we are modelling the action prior as a Dirichlet distribution means that none of the action probabilities ever decrease to zero (assuming they started with a uniform prior and $\alpha_s^0(a) > 0, \forall s, a$). As a result, an action is never wholly excluded from the action set, and so all convergence guarantees are retained.

The second experiment demonstrates the advantages of ε-QPAP in tasks with dif-

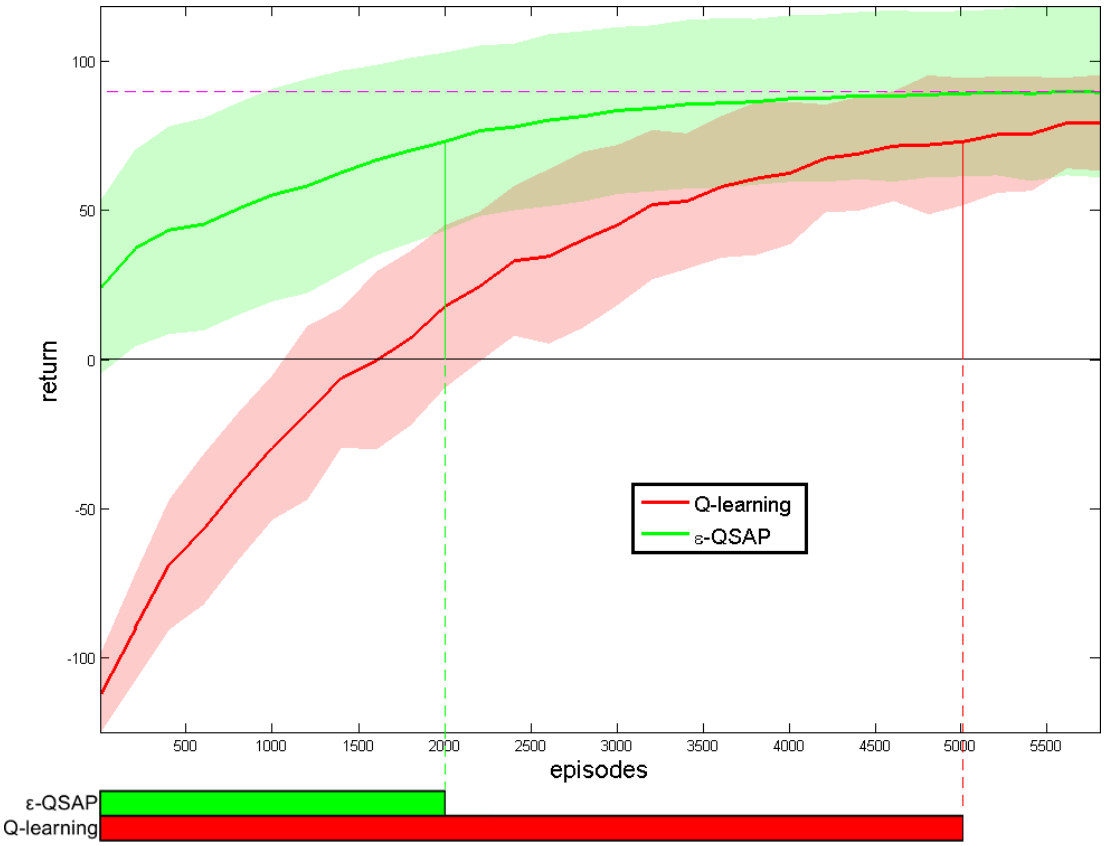

Figure 3.3: Comparing the learning rates of Q-Learning with State-based Action Priors (ε-QSAP) to Q-learning, in the 23×23 maze domain (shown in Figure 3.2). The results are averaged over 10 different random tasks. The bars below the plot indicate episodes taken for each method to reach 80% of optimal return. The shaded region indicates one standard deviation, and the dotted magenta line is the average optimal return.

ferent state spaces and transition functions. The domain here was a similar 23×23 gridworld, but instead of the lattice configuration, a random percentage of the cells are obstacles, while preserving the connectivity of the free space. Example mazes are shown in Figure 3.4.

In this domain, ε-QSAP is at a disadvantage, as the structure of the world differs greatly between trials, and in particular the connectivity of each free cell may be very different (if that cell indeed remains free between different trials). However, the perceptual contexts of the agent remain the same across trials, and so any priors learned for these percepts can be transferred between trials. We consider here simple perception of the occupancy of the 8 cells surrounding the current location of the agent. This simulates a scaled down sonar, and results in a smaller observation space than the full state space. The results are shown in Figure 3.5.

Figure 3.4: Example world instances for the second experiment, with marked start and goal locations.

As can be seen, the perception-based action priors (ε-QPAP) outperforms both other priors (state-based as well as the uniform priors of Q-learning), with a significant improvement in both initial performance and time to convergence. In this case, uniform priors actually outperform the state-based priors which are clearly wrong for this domain owing to the differences in Figures 3.2 and 3.4: they provide misleading advice in the states that were free space in the original domain, and uniform priors for the others.

3.4.2 The Factory Domain

The factory domain is an extended navigation domain that involves a mobile manipulator robot placed on a factory floor. The layout of the factory consists of an arrangement of walls through which the robot cannot move, with some procurement and assembly points placed around the factory. Additionally there are express routes demarcated on the ground, which represent preferred paths of travel, corresponding to regions where collisions with other factory processes may be less likely. The domain used in these experiments is shown in Figure 3.6.

The robot has an action set consisting of four movement actions (*North*, *South*, *East*

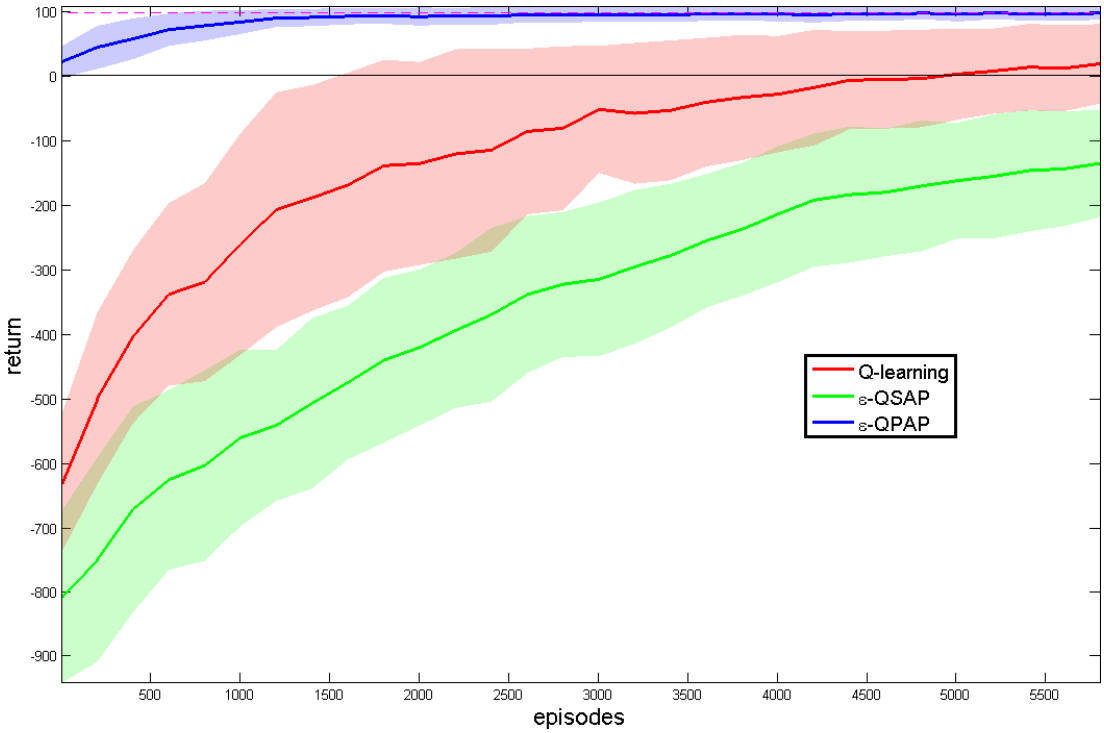

Figure 3.5: Comparing the learning rates of Q-Learning with Perception-based Action Priors (ε-QPAP), Q-Learning with State-based Action Priors (ε-QSAP) and Q-learning, in the 23×23 noise domain, with 50%-80% noise (shown in Figure 3.4). The results are averaged over 10 different random tasks. The shaded region indicates one standard deviation, and the dotted magenta line is the average optimal return.

and *West*), each of which will move the robot in the desired direction, provided there is no wall in the destination position, a *Procure* action, and an *Assemble* action. *Procure*, when used at procurement point i, provides the robot with the materials required to build component i. *Assemble*, when used at assembly point i, constructs component i, provided the robot already possesses the materials required for component i.

A task is defined as a list of components which must be assembled by the robot. The domain has 9 components, and so this list can range in length from 1 to 9, giving a total of $2^9 - 1$ different tasks.

The task rewards are defined as follows. All movement actions give a reward of -2, unless that movement results in the robot being on an express route, for a reward of -1. Collisions are damaging to the robots and so have a reward of -100. *Procure* at a procurement point corresponding to an item in the task definition which has not yet been procured gives a reward of 10. *Procure* executed anywhere else in the domain yields -10. *Assemble* at an assembly point for an item in the list which has already

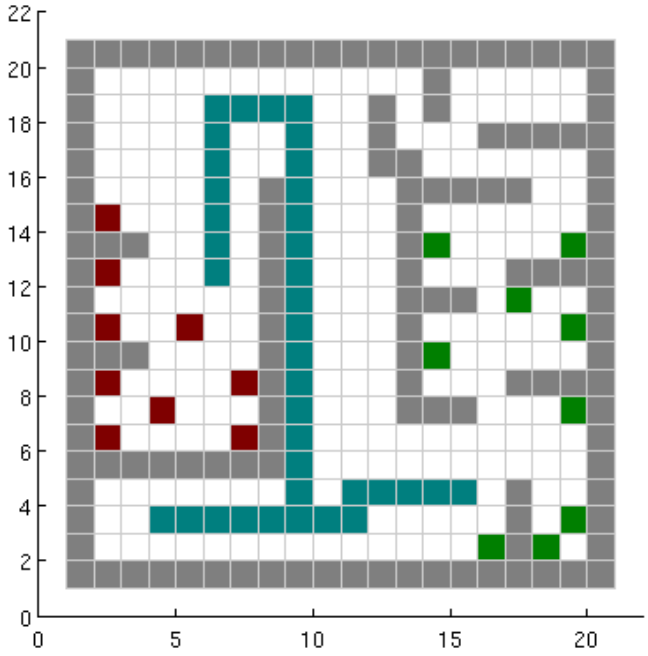

Figure 3.6: Factory domain used in the experiments. Grey cells are obstacles, white cells are free space, green cells are procurement points, red cells are assembly points, and cyan cells are express routes. This figure is best viewed in colour.

been procured but not assembled gives 10, and any other use of the *Assemble* action gives -10. Successful completion of the task gives 100 and the episode is terminated.

This domain provides a number of invariances which could be acquired by the robot. On a local level, these include avoiding collisions with walls, prefering express routes over standard free cells, and not invoking a *Procure* or *Assemble* action unless at a corresponding location. As all tasks are defined as procuring and assembling a list of components, this additionally provides scope for learning that regardless of the components required, the robot should first move towards and within the region of procurement points until all components have been procured, after which it should proceed to the region of assembly points.

3.4.3 Results with State Action Priors

The results in Figure 3.7, which compares the performance per episode of a learning agent using a set of different priors, demonstrates that using action priors reduces the cost of the initial phase of learning, which is largely concerned with coarse scale ex-

ploration. In this case, the loss incurred in the early episodes of learning is dominated by the fact that the agent explores in the wrong directions, rather than performs invalid or ill-placed actions. This figure also shows comparative performance of Q-learning with uniform priors (i.e. "standard" Q-learning), as well as with two different hand specified priors; an "expert" prior and an "incorrect" prior.

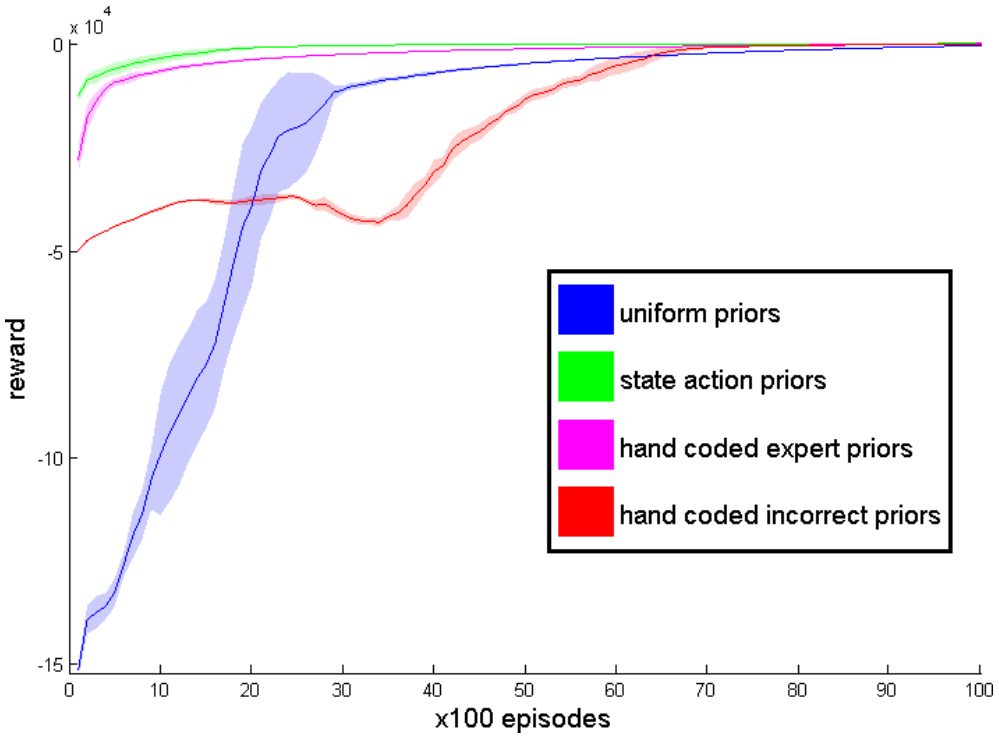

Figure 3.7: Comparative performance between Q-learning with uniform priors, Q-learning with state based action priors learned from 35 random tasks, and Q-learning with two different pre-specified priors (see text for details). These curves show learning curves averaged over 15 runs, where each task was to assemble 4 components, selected uniformly at random. The shaded region represents one standard deviation.

The "expert" prior is a prior defined over the state space, to guide the agent towards the procurement area of the factory if the agent has any unprocured items, and to the assembly area otherwise. This prior was constructed by a person, who was required to specify the best direction for each state in the domain. We note that this prior is tedious to specify by hand, as it involves an expert specifying preferred directions of motion for the entire state space of the agent (number of states in the factory $\times$ number of different item configurations). Note that, although the performance is very similar, this prior does not perform as well as the learnt prior, likely due to a perceptual bias on behalf of the expert's estimation of optimal routing.

We also compare to an "incorrect" prior. This is the same as the expert prior, but we simulate a critical mistake in the understanding of the task: when the agent has unprocured items, it moves to the assembly area, and otherwise to the procurement area. This prior still provides the agent with an improvement in the initial episodes of uniform priors, as it contains some "common sense" knowledge including not moving into walls, moving away from the start location, etc. Q-learning is still able to recover from this error, and ultimately learn the correct solution.

Figure 3.8 shows the speed up advantage in learning a set of $N = 40$ tasks, starting from scratch and then slowly accumulating the prior from each task, against learning each task from scratch. This case is for a simple version of the task, which involved procuring and assembling a single item. As a result, all task variants are likely to have been encountered by the time the agent is solving the final tasks.

On the other hand, Figure 3.9 shows that a similar effect can be observed for the case of a more complicated task, requiring the assembly of 4 randomly selected items. In this case, even by the time the learning agent has accumulated a prior composed from 40 tasks, it has only experienced a small fraction of the possible tasks in this domain. Despite this, the agent experiences very similar benefits to the single item case.

3.4.4 Results with Observation Action Priors

In order to demonstrate the effect of using the observation action priors, we present a modification of the factory domain. Recall that, as state action priors define distributions over each state of the domain, they cannot be robustly ported between similar domains. On the other hand, as observation action priors are based on local features rather than global state information, this information is more portable, albeit possibly less informative. The modified factory domain provides a test-bed to demonstrate this point.

The modified domain stipulates that the factory floor layout changes for each different task. This corresponds to either the learning agent moving between different factories, or the fact that the obstacles, procurement and assembly points may be mobile and change with some regularity (e.g. whenever a new delivery is made). The factory floor consists of a 3×3 lattice of zones, each of which is 6×6 cells. There is an outer wall, and walls in between every two zones, with random gaps in some (but not all) of these walls, such that the entire space is connected. Additionally, each

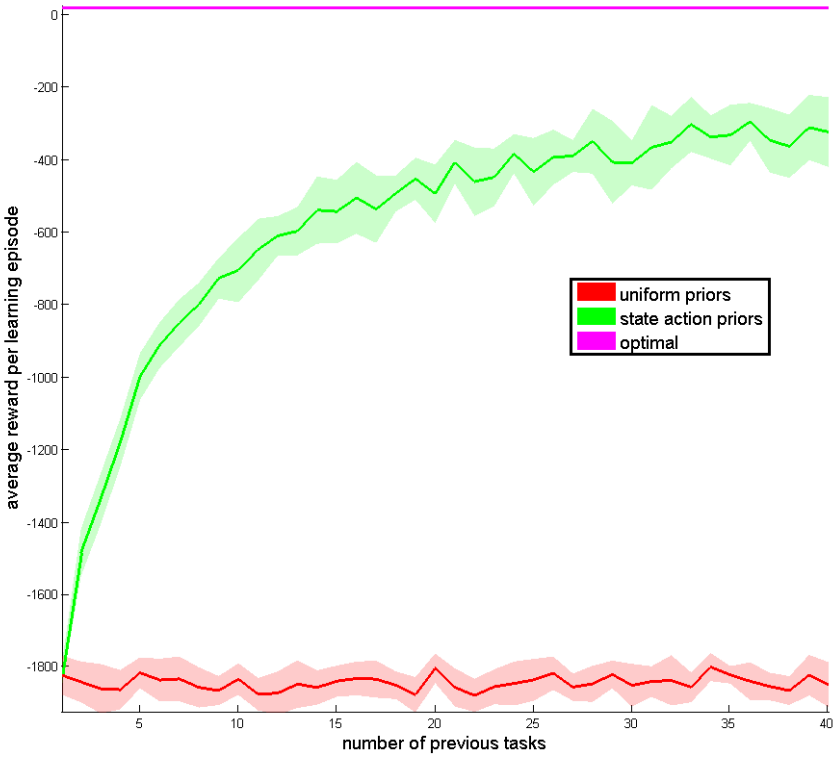

Figure 3.8: Comparative performance between Q-learning with uniform priors, and Q-learning with state based action priors, showing the effect of accumulating the action priors from an increasing number of tasks. The number of prior tasks range from 0 to 40. These curves show the average reward per learning episode averaged over 10 runs, where the task was to assemble 1 random component. There are only 9 possible tasks in this environment, and so this curve shows the effect of the action priors when all tasks have eventually been seen before. The shaded region represents one standard deviation. The “optimal” line refers to average performance of an optimal policy, which will necessarily be higher than the per episode reward of a learning algorithm.

zone contains randomly placed internal walls, again with the connectivity of the factory floor maintained. Two zones are randomly chosen as procurement zones, and two zones as assembly zones. Each of these chosen zones has either four or five of the appropriate work points placed at random. Examples of this modified factory domain are shown in Figure 3.10.

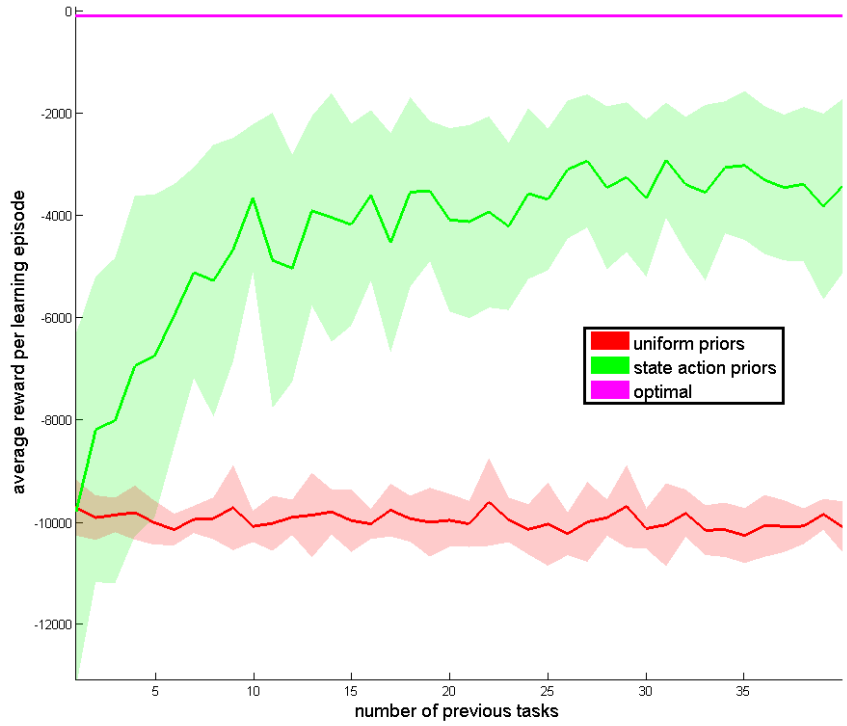

Figure 3.9: Comparative performance between Q-learning with uniform priors, and Q-learning with state based action priors, showing the effect of accumulating the action priors from an increasing number of tasks. The number of prior tasks range from 0 to 40. These curves show the average reward per learning episode averaged over 10 runs, where the task was to assemble 4 random components. The shaded region represents one standard deviation. The "optimal" line refers to average performance of an optimal policy, which will necessarily be higher than the per episode reward of a learning algorithm.

3.4.5 Feature Selection

This modified domain has a different layout for every task. As a result, every task instance has a different transition function T. This is in contrast to the original factory domain, where each task differed only in reward function R. State based action priors can therefore not be expected to be as useful as before. We thus use observation priors and discuss four particular feature sets.

Figure 3.11 demonstrates the improvement obtained by using observation priors over state priors in this modified domain. Note here that the state priors still provide some benefit, as many of the corridor and wall placings are consistent between task

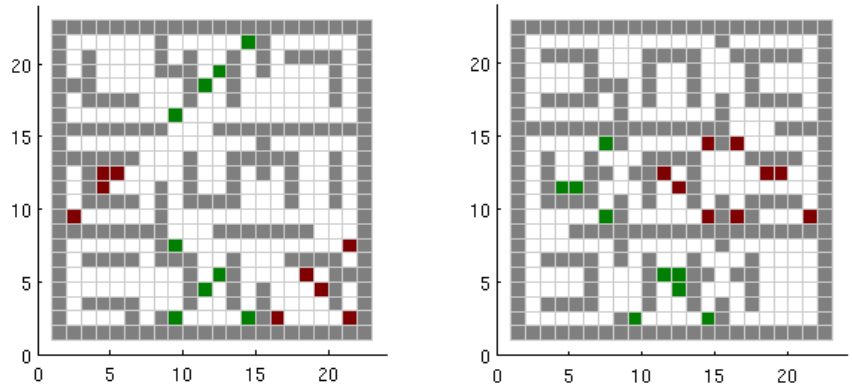

Figure 3.10: Two instances of the modified factory domain used in the experiments. Grey cells are obstacles, white cells are free space, green cells are procurement points, and red cells are assembly points. The procurement and assembly points count as traversable terrain. This figure is best viewed in colour.

and factory instances. Figure 3.11 shows the effect of four different observation priors:

- ϕ_1: This feature set consists of two elements, being the type of terrain occupied by the agent (in $\{free, wall, procure\text{-}station, assembly\text{-}station\}$), and a ternary flag indicating whether any items need to be procured or assembled.
- ϕ_2: This feature set consists of four elements, being the types of terrain of the cells adjacent to the cell occupied by the agent.
- ϕ_3: This feature set consists of six elements, being the types of terrain of the cell occupied the agent as well as the cells adjacent to that, and a ternary flag indicating whether any items need to be procured or assembled. Note that the features in ϕ_3 are the union of those in ϕ_1 and ϕ_2.
- ϕ_4: This feature set consists of ten elements, being the types of terrain of the 3×3 grid of cells around the agent's current position, and a ternary flag indicating whether any items need to be procured or assembled.

As can be seen, these four observation priors all contain information relevant to the domain, as they all provide an improvement over the baselines. There is however a significant performance difference between the four feature sets. This difference gives rise to the idea of using the priors for feature selection, as discussed in Section 3.3.2.

Surprisingly, Figure 3.11 shows that the most beneficial feature set is ϕ_3, with ϕ_2 performing almost as well. The fact that the richest feature set, ϕ_4, did not outperform

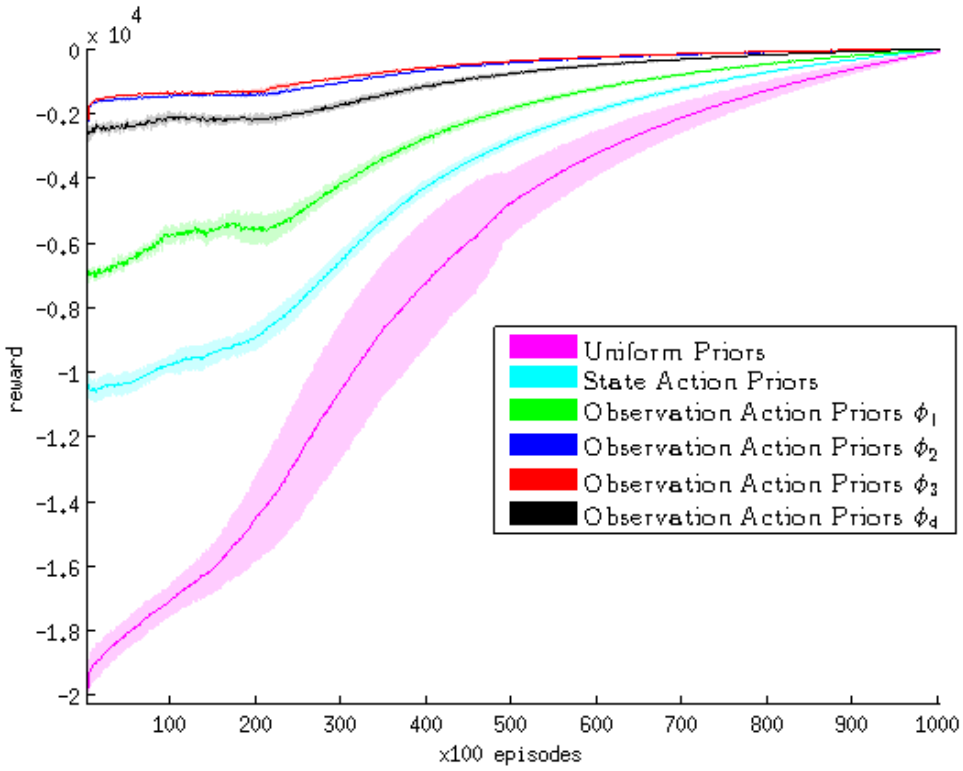

Figure 3.11: Comparative performance in the modified factory domain between Q-learning with uniform priors, Q-learning with state based action priors, and Q-learning with four different observation based action priors: ϕ_1, ϕ_2, ϕ_3 and ϕ_4. These curves show the average reward per episode averaged over 10 runs, where the task was to assemble 4 random components. In each case the prior was obtained from 80 training policies. The shaded region represents one standard deviation.

the others seems counterintuitive. The reason for this is that using these ten features results in a space of $4^9 \times 3$ observations, rather than the $4^5 \times 3$ of ϕ_3. This factor of 256 increase in the observation space means that for the amount of data provided, there were too few samples to provide accurate distributions over the actions in many of the observational settings.

We attempt to identify the set of the most useful features using Algorithm 5. These results are shown in Figure 3.12.

In this approach, we iteratively remove the features which contribute the least to reducing the entropy of the action priors. Recall that when posed as an optimisation problem in Equation (3.12), we use the term $c\|\phi\|$ as a regulariser to control for the effect of having too large a feature set: the effect seen in the case of ϕ_4 in Figure 3.11.

The results in Figure 3.12 show that the relative importance for the ten features

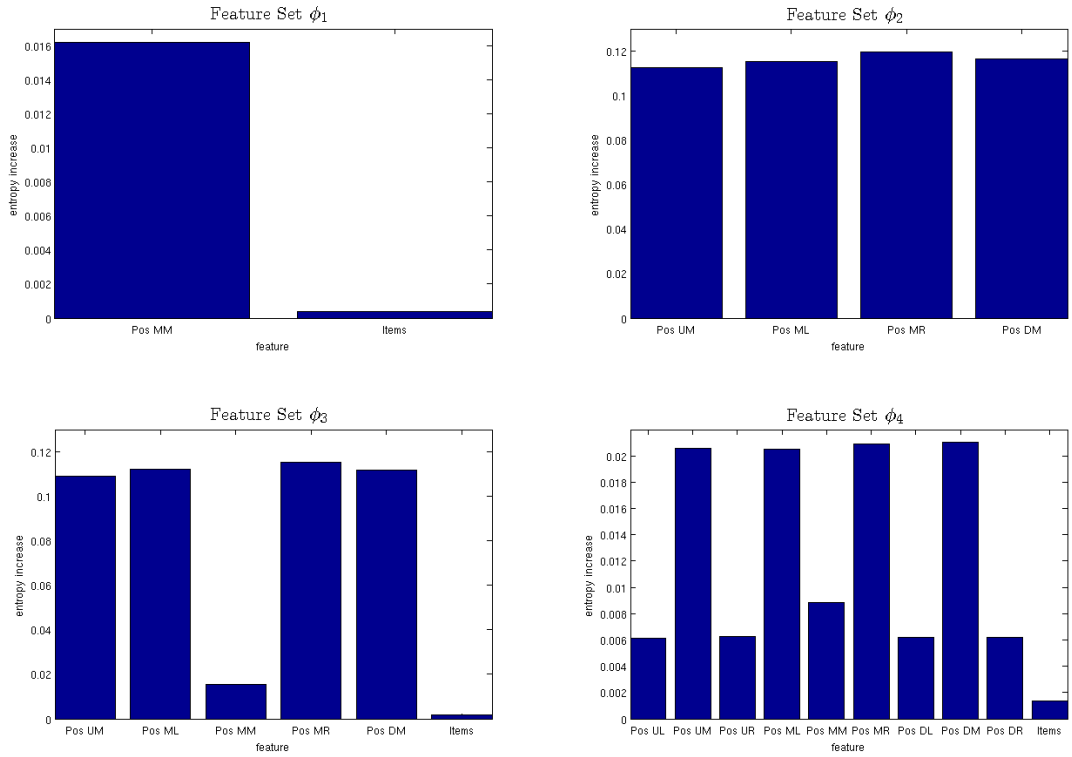

Figure 3.12: Feature importance of each feature in the four different observation based action priors: ϕ_1, ϕ_2, ϕ_3 and ϕ_4. The spatial features are labelled $PosYX$, with $Y \in \{(U)p, (M)iddle, (D)own\}$ and $X \in \{(L)eft, (M)iddle, (R)ight\}$. The feature $Items$ is a flag, indicating if the agent still needs to assemble or procure any items.

(all of which are present in ϕ_4) are consistent across the four feature sets. As may be expected, the ϕ_4 results indicate that the values of the cells diagonally adjacent to the current cell occupied by the agent are not important, as they are at best two steps away from the agent.

What is surprising at first glance is that neither the value of the cell occupied by the agent, nor the current state of the assembly carried by the agent are considered relevant. Consider the current state of assembly: this is actually already a very coarse variable, which only tells the agent that either a procurement or an assembly is required next. This is very local information, and directly affects only a small handful of the actions taken by the agent. Now consider the cell currently occupied by the agent. This indicates whether the agent is situated above an assembly or procurement point. Again, this is only useful in a small number of scenarios. Note that these features are still useful, as shown by the performance of ϕ_1 relative to state based priors or uniform priors.

What turns out to be the most useful information is the contents of the cells to the North, South, East and West of the current location of the agent. These provide two critical pieces of information to the agent. Firstly, they mitigate the negative effects that would be incurred by moving into a location occupied by a wall. Secondly, they encourage movement towards procurement and assembly points. These then constitute the most valuable features considered in our feature sets. This observation is confirmed by the fact that ϕ_2 performs very similarly to ϕ_3.

3.4.6 Online Feature Selection

We now demonstrate the performance of the adaptive priors generated through the use of online feature selection in Algorithm 6. This is shown in Figure 3.13. Note that in the first episode of learning, the performance of the agent with feature selection is actually slightly better than that of the agent with the full prior, again showing the cost incurred through the use of an overly rich feature set. The performance of both is considerably better than not using a prior. However, at convergence, all three methods achieve the same performance (shown by the three lines at the top of the figure).

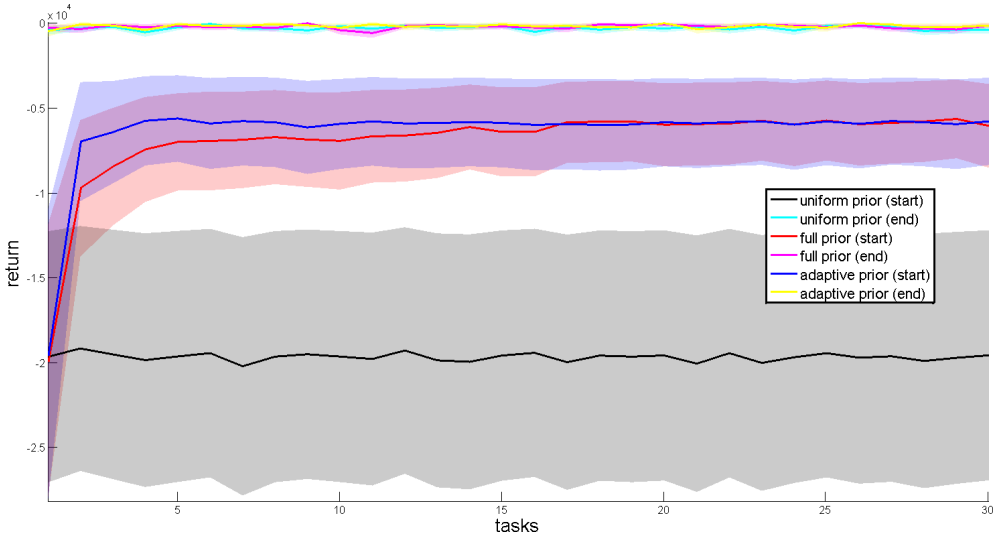

Figure 3.13: Performance of learning agent during first (start) and 30^{th} (end) episodes, over thirty tasks. Results compare the use of uniform priors, using the full set of 10 features, and using the adaptive prior of online feature selection. Results are averaged over 10 runs, and the shaded region represents one standard deviation.

Figure 3.14 shows the features selected by the algorithm as a function of the number of tasks experienced. The result is also a considerable reduction in feature space

descriptions. The algorithm is seen to converge to the four features used in ϕ_2.

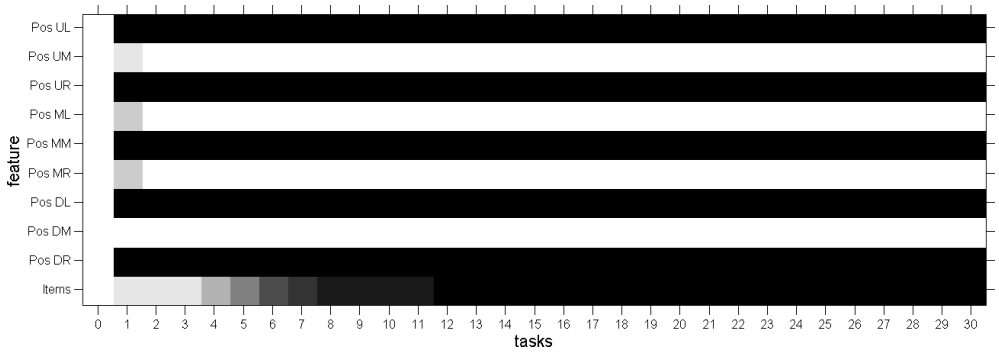

Figure 3.14: Features selected during online feature selection over 30 tasks, averaged over 10 runs. The brightness of each square indicates the percentage of runs for which that feature was selected. The spatial features are labelled $PosYX$, with $Y \in \{(U)p, (M)iddle, (D)own\}$ and $X \in \{(L)eft, (M)iddle, (R)ight\}$. The feature $Items$ is a flag, indicating if the agent still needs to assemble or procure any items.

The effect of this is that a reduced set of four features is shown to be preferable to the initial full set of ten which were proposed for this domain in ϕ_4. The benefit is that the number of possible feature configurations has thus been reduced from $4^9 \times 3$ down to 256: a reduction of 99.71%. Learning behaviours for these observation configurations is then far simpler, and provides the agent with a conveniently small repertoire of useful local behavioural capabilities.

3.4.7 Action Priors as Dirichlet Distributions

We now validate the choice of using Dirichlet distributions to model the action priors. Figure 3.15 illustrates the differences between the functions discussed in Section 3.2.2.1, being modelling the priors as proportions, Dirichlet distributions, or softmax distributions.

Proportions and the Dirichlet distributions perform comparably, which is to be expected as the proportions are the normalised Dirichlet parameters, and thus the mean of the Dirichlet. These distributions both outperform the softmax, again as expected. In particular, the softmax solution performs poorly during the initial episodes, with a high variance in performance across tasks. This is because the softmax function tends to bias probabilities towards either 0 or 1 through the exponential term, resulting in more peaked distributions. The effect is that these priors would be much better suited to some tasks than others, as indicated by the high variance.

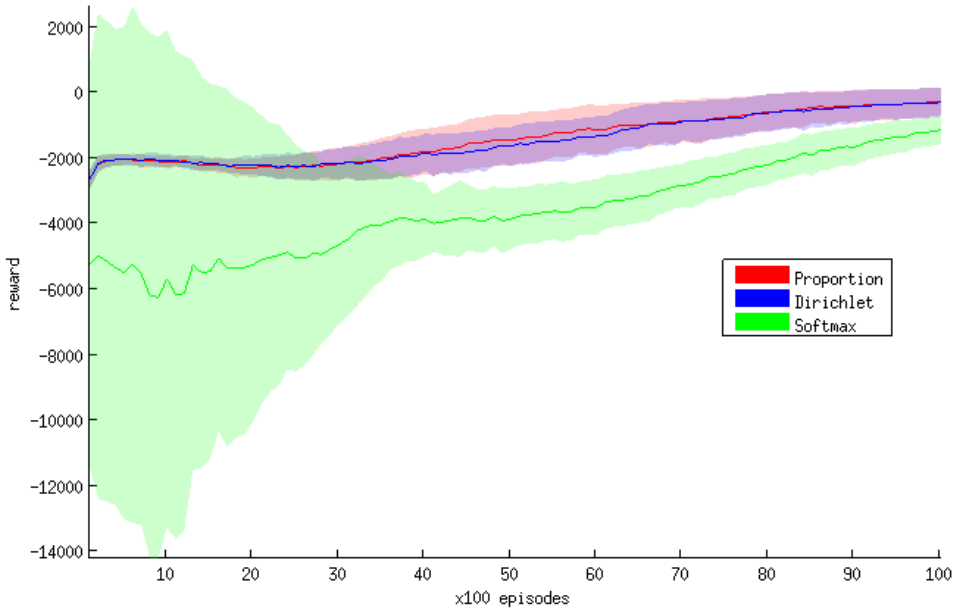

Figure 3.15: Effects of the three different probability function choices proposed in Section 3.2.2.1 in the modified factory domain: using proportions of the utility functions, Dirichlet distributions, or a softmax. These curves show the average reward per episode averaged over 20 runs, where the task was to assemble 4 random components. In each case the prior was obtained from 80 training policies. The shaded region represents one standard deviation.

3.4.8 Human Elicited Priors

A benefit of action priors is that they need not all be learnt from policies executed by the same agent. In this way, the priors can be accumulated from a number of different agents operating in the same space. Furthermore, trajectories can be demonstrated to the learning agent, perhaps by a human teacher, as a means of training.

Figure 3.16 illustrates how human elicited action priors can be used to improve learning performance on a new task. In this case, solution trajectories were provided by a human for 40 randomly selected assembly tasks in the modified factory domain. It can be seen that the learning agent performs comparably to having full prior policies, even though only trajectories were provided.

The human supplied trajectories were not assumed to be optimal in behaviour, but the human was assumed to be familiar enough with the task and the environment such that the loss incurred when compared to an optimal solution should be smaller than some bound. This demonstrates that our action prior approach works even when the

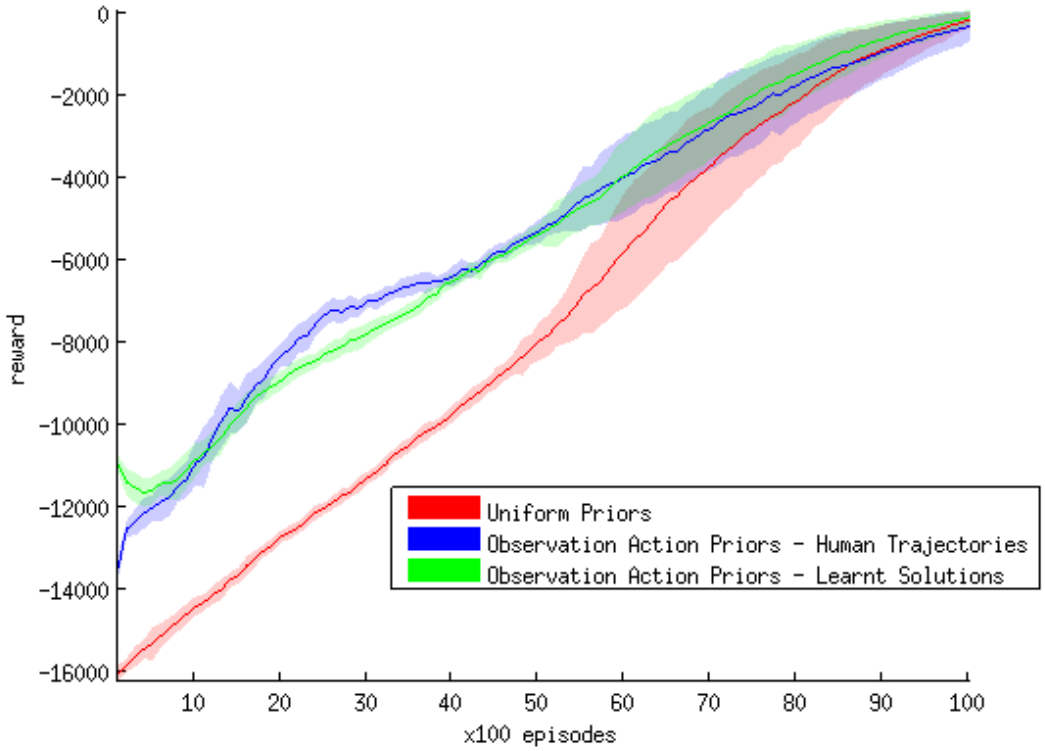

Figure 3.16: Comparative performance in the modified factory domain between Q-learning with uniform priors, Q-learning with *human elicited* observation based action priors, and Q-learning with self trained observation based action priors. These curves show the average reward per episode averaged over 20 runs, where the task was to assemble 4 random components. In each case the prior was obtained from 40 training policies. The shaded region represents one standard deviation.

training data is sub-optimal. We also note here that our method can be easily applied to a case where a number of human demonstrators could supply trajectories. Any trajectories which are estimated to be poorer in quality could be weighted by a smaller value of $w(\pi)$, in Equation 3.3. Practically, a weight could be computed for each human providing training trajectories which indicates the competence level of that human at that task, and all trajectories supplied by that human would be weighted accordingly.

These results suggest that action priors can feasibly be obtained by human demonstration, and used to model human preferences in such a way as to guide the learning of an autonomous agent on a similar task.

3.5 Priors in Humans

As was shown in Section 3.4, action priors can provide substantial benefits to learning and decision making, through their ability to prune and prioritise action choices in a context dependent manner. This works by guiding the search process towards solutions that are more likely to be useful, and have been in the past.

Making good decisions requires evaluating the effects of decisions into the future, so as to estimate the value of each choice. However, this is an expensive search process, as the number of possible futures is exponential in the number of choices to be made, with a branching factor given by the number of actions which can be taken at each point in time.

This is a problem shared by humans. When a person is faced with any planning problem, or any decision, this cannot be solved by evaluating all possible sequences of actions, for the aforementioned reason. It is thus assumed that some automatic pruning of human decision trees occur, and negative rewards (or punishments) seem to be particularly effective at inhibiting behaviours (Huys et al., 2012). There is evidence to suggest that this pruning is Pavlovian in nature, particularly when exposed to a large negative feedback, and that this pruning is task independent. There is also much evidence to suggest that humans rely heavily on prior knowledge for decision making, although that process is not always conscious (Strevens, 2013).

When making decisions which require a search through large and rich problem spaces, humans thus seem to select a subset of the valid choices for further consideration. This is supported by long-standing results in psychology. Simon and Chase (1973), for example, used the game of chess as a vehicle for exploring the theory of chunking. A chunk refers to a recognisable set of features, which in this context is a local pattern on the chess board. Various experiments were run on players of differing skill levels, testing their ability to recall and reconstruct sequences of briefly observed board positions, which were either drawn from real games or random. These results supported the conclusion that the more skilled a player, the larger the repertoire of such patterns they have stored in memory (estimated to be of the order of well over 50,000 for expert players). Having additional information associated with these chunks would account for the ability of an expert to notice advantageous moves “at a glance” – essentially pruning away poor action choices. As a result, recognising chunks reduces the need for look-ahead search.

In this way, perceptual cues are easily recognised, and in turn they trigger actions,

which could be thought of as an *intuition* of the current situation (Simon, 1992). Unconscious early perceptual processes learn the relationships between the visual elements of a scene under inspection. These are invoked fast and in parallel, and act as a pre-processing step for later stages of cognition (Harre et al., 2012).

Later work on template theory (Gobet and Simon, 1996) extends this idea, by incorporating the idea of templates, which allow for open parameters to be present in recognisable patterns. Templates provide further semantic information which may be helpful in making action selection, again pruning and biasing the search process. It has thus been conjectured in multiple studies that "experts use the rapid recognition of complex patterns, mediated by perceptual templates, in order to efficiently constrain and guide their search for good moves" (Harre et al., 2012).

This interpretation corresponds to the semantics of our action priors, which allow a search process to prioritise its exploration based on the recommended actions for any given context. Our action prior approach also hypothesises and demonstrates how these priors may be gathered from past experience.

The overall effect is that if one does not know what action to take, given uncertainty in the task, it could be chosen according to the action prior, which represents common sense and intuition in the domain. This bias can be used in instantaneous action selection to cause no immediate harm in the absence of more task specific knowledge.

3.6 Related Work

Recent research on learning policy priors (Wingate et al., 2011) (and the earlier work by Doshi-Velez et al. (2010)) has similar aspirations to our own. They propose a policy search algorithm, based on MCMC search, which learns priors over the problem domain. The method requires the specification of a hyperprior (abstract prior knowledge) over the prior, with the effect that the method learns priors which it is able to share among states. For example, the method can discover the dominant direction in a navigation domain, or that there are sequences of motor primitives which are effective and should always be prioritised during search. They perform inference on (π, θ), where π is the policy, and θ the parameters by casting this search problem as approximate inference, over which they can specify or learn the priors. Our work does not assume a known model, or kernel, over policy space. This means we cannot sample from a generative model as is typical in the Bayesian reinforcement learning setting.

An alternative method for transferring information from previous tasks to act as pri-

ors for a new task is through model-based approaches, such as those taken by Sunmola (2013). This embraces the Bayesian reinforcement learning paradigm of maintaining a distribution over all possible transition models which could describe the current task and environment, and updating this distribution (belief) every time an action is taken. Transfer here is achieved by using the experience of state transitions in previous tasks to update beliefs when it first encounters each state in the new task, before anything is known about the transition probabilities from that state. Local feature models are also used to facilitate generalisation.

The strength of transferring action probabilities, rather than model probabilities is three-fold. Firstly, the information present in action priors is very general, with the only requirement being local usefulness of actions. As such, agents learning with action priors can easily be put through learning *curricula*, and then deployed to vastly different environments. Secondly, transferring knowledge through action models is an idea applicable to a wide range of different decision making paradigms, including model-free and model-based reinforcement learning, planning and online learning. Finally, an action prior has an intuitive interpretation as an understanding of the "common sense" behaviours of some local space.

Work by Sherstov and Stone (2005) also has similar aspirations to those of our own methods. In problems with large action sets ($|A| \sim 100$, often from parametrised actions), they also try to either cut down the action set, or bias exploration in learning. The difference in this work is that the reduced action set, based on what they call the relevance of an action, is determined from the training data of optimal policies for the *entire* domain, rather than for each state or observation. This has the effect of pruning away actions that are always harmful throughout the domain, but the pruning is not context-specific.

Our interpretation of action priors as distilling domain specific knowledge from a set of task instance specific behaviours is similar to the idea of dividing a problem (or set thereof) into an agent space and a problem space (Konidaris and Barto, 2006). The agent space refers to commonalities between the problems, whereas the problem space is specific to each task. This formulation involves learning a reward predictor which, in a sense, can be used to guide action selection.

Where our approach reduces computation by biasing and restricting the search over action space, similar benefits have been found by only searching over limited aspects of the state space, particularly in relational planning problems. Notable examples include reasoning only in terms of the subset of objects that are relevant for current plan-

ning purposes (relevance grounding) (Lang and Toussaint, 2009), or using variables to stand in for the objects relevant to the current actions (deictic references) (Pasula et al., 2007). This is similar to the way in which pruning is used in search, but we prune based on the expected utility of the action which is estimated from the utility of that action over the optimal policies for a set of previous tasks.

Options (Precup et al., 1998) are a popular formalism of hierarchical reinforcement learning, and are defined as temporally extended actions with initiation sets where they can be invoked, and termination conditions. There are many approaches to learning these, see e.g. Pickett and Barto (2002). Although there are similarities between learning the initiation sets of options and action priors, they are distinct, in that an initiation set defines where the option *can physically be instantiated*, whereas an action prior describes regions where the option is *useful*. This is the same distinction as must be drawn between learning action priors and the preconditions for planning operators (e.g. Mourão et al. (2012)). For example, while pushing hard against a door may always be physically possible, this level of force would be damaging to a glass door, but that choice would not be ruled out by options or planning preconditions. Consequently, action priors not only augment preconditions, but are beneficial when using large sets of options or operators, in that they mitigate the negative impact of exploration with a large action set.

One approach to reusing experience is to decompose an environment or task into a set of subcomponents, learn optimal policies for these common elements, and then piece them together (Foster and Dayan, 2002), possibly applying transforms to make the subcomponents more general (Ravindran and Barto, 2003). This is the philosophy largely taken by the options framework. Our method differs by discovering a subset of reasonable behaviours in each perceptual state, rather than one optimal policy. Our priors can thus be used for a variety of different tasks in the same domain, although the policy must still be learned. As a result, our method is also complementary to this decomposition approach.

The idea of using a single policy from a similar problem to guide exploration is a common one. For example, policy reuse (Fernandez and Veloso, 2006) involves maintaining a library of policies and using these to seed a new learning policy. The key assumption in this work is that the task for one of the policies in the library is similar to the new task. While we acknowledge that this is a very sensible approach if the policies are indeed related, we are instead interested in extracting a more abstract level of information about the domain which is task independent, and thereby hopefully

useful for any new task.

Other authors have explored incorporating a heuristic function into the action selection process to accelerate reinforcement learning (Bianchi et al., 2007), but this work does not address the acquisition of these prior behaviours. This approach is also sensitive to the choice of values in the heuristic function, and requires setting additional parameters.

Action priors are related to the idea of learning *affordances* (Gibson, 1986), being action possibilities provided by some environment. These are commonly modelled as properties of objects, and these can be learnt from experience (see e.g. Sun et al. (2010)). The ambitions of action priors are however slightly different to that of affordances. As an example, learning affordances may equate to learning that a certain class of objects is "liftable" or "graspable" by a particular robot. We are instead interested in knowing how likely it is that lifting or grasping said object will be useful for the tasks this robot has been learning. Ideally, action priors should be applied over action sets which arise as the result of affordance learning.

Another related concept is that of optimising the ordering of behaviours in a policy library (Dey et al., 2012a,b). The idea is to use a set of classifiers or experts to select a sequence of behaviours which should be tried by an agent in its current setting, thereby providing an ordered list of fallback plans. The difficulty in our reinforcement learning setting is not only that the state or observation context is important, but our setting considers different tasks, each of which would require different local actions. Additionally, in the reinforcement learning context, it can be difficult to evaluate how good a choice a particular action is, which is important for updating the weights over the experts. This approach would also not necessarily guarantee that every action is chosen infinitely often in the limit of infinite trials, which is a requirement for the convergence of Q-learning.

3.7 Using Action Priors to Give Advice to Agents with Hidden Goals

As described in Section 3.2, action priors build a model of the general knowledge, or "common sense" behaviour, of a domain through the combination of the common elements of policies used to solve a variety of different tasks in the same domain. The effect is that by observing one or more agents carry out tasks in a domain, one obtains

a good estimate of the best actions to invoke in each situation. In previous sections we used this methodology to address the idea of accelerating the learning of new tasks in the same agent. The same approach can however be used in a multi-agent setting, where one agent provides advice to another, based on action priors it has learnt from the behaviour of numerous other agents.

Consider one agent, the *explorer*, moving about a large hospital, with the intended destination of a particular ward, but no idea of where to find that ward. Now consider that a second agent, an autonomous *advisor* installed throughout the "intelligent building", was watching the progress of the explorer, and noticed that it seemed to be lost. In this case, it would be desirable to provide some advice on likely directions in which to move. Without knowing the intended target of the explorer, but from observing its behaviour, the advisor should make suggestions based on the most commonly used paths through the hospital – the "common sense" knowledge of the building. This could guide the explorer towards major wards and waiting areas, and away from service regions. This section examines this question of how and when to provide advice to an agent with unknown goals.

Furthermore, the advisor does not know how much advice is actually needed by the explorer. If the explorer truly has no idea of what it is doing, then more information should be provided. On the other hand, especially if the advice is given via a mobile robot component, this is a costly resource that could be moved around to be shared between people – so, constant advice really is something to be minimised. Additionally, we assume that the advisor does not know if the agent is human or artificial, nor if it is acting randomly, with purpose, or learning. This situation is caricatured in Figure 3.17.

This is an important problem for both fixed-policy and learning agents. With the help of advice, a fixed-policy agent can complete a task sufficiently faster than it would have otherwise. Similarly, advice can be a boost to a learning agent in terms of both speed and safety, in that exploration can be guided away from dangerous or useless regions of the domain.

We propose to address this problem by learning a model of "common sense" behaviour over the entire domain, and marginalising over the different tasks. Offline, we learn a model of typical users in this domain in the form of action priors, from expert agents carrying out *different* tasks in the *same* domain. Online, a new user enters the domain. Both the behavioural policy and the goal of this user are unknown *a priori*. The advisor continually estimates the performance of the user by comparing behaviour

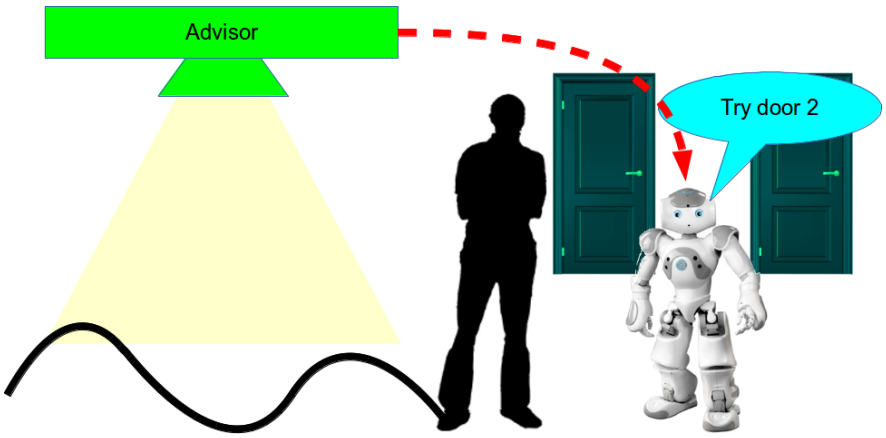

Figure 3.17: The envisaged scene: an agent (the human) is exploring some environment. Its path is monitored by an autonomous advisor, which can then deploy some resource (a robot) to offer advice to the agent.

to the action prior model, and then based on this, provides advice in states where it is estimated to be needed.

Basing advice on the behaviour of experts in the same space has limitations, in that it depends on the experts visiting the same locations and performing the same actions as would be required by the explorer. However, we wish to advise based on the way in which the domain is used – its *functional semantics*. If a particular task has not been seen before, there is nothing we can do regarding advising about that task, but there is still no need for the agent to collide with walls or enter dead ends. We consequently reason at the level of action distributions, rather than tasks, as we can leverage the fact that there are overlaps between the behaviours needed for different tasks.

In general, different types of agents may have different distributions over their expected behaviours, e.g. hospital staff vs visitors. This can be taken into account in our framework by augmenting the state space with some description of the agent, if that sensing information is available, e.g. if they are wearing a uniform or name tag.

Another example of this problem is that of building an autonomous advisor to teach humans or other autonomous agents to play complicated video games where gameplay is not linear, and there are many different subgoals. Many modern games offer environments where players may progress through the game in any of a number of different ways, or even combinations thereof. Learning the combination of all these play styles and subgoals would not be feasible, particularly if the data is sparsely provided through very few instances of each. Consequently, learning a model of common sense behaviour provides the advisor with a compact description of reasonable courses of

action.

3.7.1 Advice Giving using Action Priors

To provide advice, we do not need to assume that the advisor has the same action set as the explorer. Instead, the advisor can provide advice over the *outcomes* of the actions (Sherstov and Stone, 2005). In this way, we require that all agents must have state and action sets mappable to a common set used by the advisor. The implication of this is that the state and action representations used by each agent can differ, but effects of these actions should be common to all agents. For instance, each agent may have a different mechanism for moving, but they should all have actions with outcomes of movement in the same directions. In what follows, we assume the action sets are the same for all agents. The advisor also needs to be able to observe the outcomes of the actions taken by every agent. Learning the model of typical domain behaviours is not possible using observations of state alone as, for example, the advisor would be unable to distinguish between an agent repeatedly running into a wall, or standing still. An illustration of providing advice based on the action priors which are computed from previous trajectories through the space is shown in Figure 3.18.

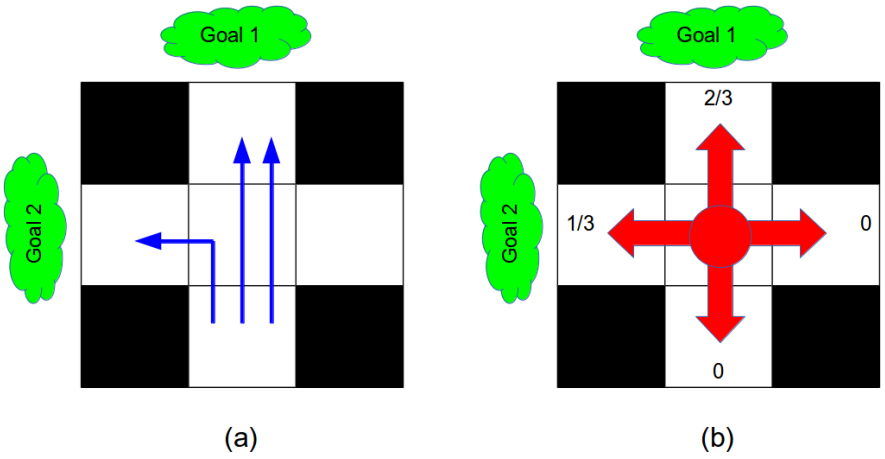

Figure 3.18: An illustration of using the action priors for advice in a 3×3 region of a larger domain. (a) Assume three expert trajectories (shown in blue) have passed through this region. (b) When providing advice to an agent situated in the centre cell, the suggested probabilities (shown in each cell) of taking each direction are computed from the behaviour of the previous trajectories.

The advisor's goal is to estimate how much help the agent needs by comparing behaviour to the action prior representing common sense behaviour in the domain, and

then based on this, to provide advice to the agent in critical locations.

3.7.1.1 What Advice to Give

When advice is given to the explorer, we assume it is given as the action prior at the state it currently occupies. In this way, we follow a model of providing *action advice* (Torrey and Taylor, 2013) as a method of instruction which does not place restrictive assumptions on the differences between the advisor and the explorer. The advice is thus a probability distribution over the action space, as obtained from the prior. If the agent is currently in state s, the advisor provides the advice $\theta_s(A)$.

The proposed protocol for using the offered advice is that the explorer should select the next action according to this advised distribution. This protocol for the advice to be provided as a distribution rather than recommending a single action allows the explorer to incorporate its own beliefs about the next action it should choose. In this way, the explorer may thus alternatively use this provided action distribution to update its own beliefs from which to select an action.

Note that if the explorer uses the proposed protocol of sampling an action from the offered advice, then the advisor could equivalently have sampled from $\theta_s(A)$ and offered a single action. The choice of protocol here would depend on the mechanism used to convey the advice to the agent.

As described in Section 3.2, the action priors describe a model of behaviours in a single domain, marginalised over multiple tasks. We thus inform the instantaneous motion of the agent such that it matches this common behaviour. This information will guide the agent towards regions of the domain in proportion to their importance for reaching the set of known domain goal locations.

3.7.1.2 When to Give Advice

There are a number of reasons why the advisor may not provide advice to the explorer at every time step. Costs may be incurred by the advisor when it provides advice, in terms of the resources required to display the advice to the explorer. In the hospital example above, these could take the form of ground lighting pointing the way, electronic signage, or dispatching a robot to the location of the agent. Additionally, interpreting this advice may cost the explorer time, and anecdotal evidence suggests that humans are easily annoyed by an artificial agent continually providing advice where it is not wanted.

The result is that the advisor should provide advice only as it is needed. To do so, it computes an estimate the amount of advice required by the agent, from the probability of the agent's trajectory under the action prior at time t. This probability is a measure of the agent's similarity to population behavioural normalcy. Because the action prior models normalcy in the domain, deviation from this corresponds to fault detection in the absence of task-specific knowledge.

Ideally, advice should only be given if the benefit of the advice outweighs the penalty incurred from the cost of giving advice. We assume the penalty of giving advice is a constant κ for the domain. This parameter could, e.g., correspond to average cost of dispatching a robot, or alternatively to some estimate of user annoyance. The benefit of giving advice at state s is the utility gain $\Delta U(s)$ from using the advice, rather than taking some other action. Because we do not know the actual values of any states in the domain, having learnt from expert trajectories only with no access to reward functions, we approximate this utility gain as a function of the difference between the probabilities of action selection under the action prior, and the likely action of the agent. This gives that

$$\Delta U(s) \simeq KL\Big[\theta_s(A), P(A|s,H)\Big], \tag{3.13}$$

where $KL[\cdot,\cdot]$ is the KL-divergence. $P(A|s,H)$ is the action selection probabilities of the explorer in state s, having followed the state-action history H, computed by

$$P(A|s,H) \;=\; \int_M P(A|s,M)P(M|H)dM, \tag{3.14}$$

with $P(A|s,M)$ being the action selection probabilities in state s under a model M. $P(M|H)$ is the probability that the agent is selecting actions according to M, given the state-action history H, and is computed according to

$$P(M|H) \;=\; \frac{P(H|M)P(M)}{\int_M P(H|M)P(M)dM}. \tag{3.15}$$

Combining these concepts, we derive a decision rule for giving advice. The explorer is given advice if the following condition holds:

$$KL\Big[\theta_s(A), \int_M P(A|s,M)P(M|H)dM\Big] \geq \kappa. \tag{3.16}$$

Condition (3.16) requires that a set of different behavioural models be defined. The action prior provides a model of normal, common sense behaviour in the domain. Additionally, other models must be defined which describe other classes of behaviour. We assume there are two models M:

1. M_1: *Normalcy*, modelled by the action prior, empirically estimates the probability of selecting actions from the behaviour of expert agents performing different tasks.

2. M_2: *Uniformity*, which models the explorer not knowing what it is doing or where it is going, involves selecting actions with uniform probability.

Without further information, we assume that these two models have equal prior probability, and so $P(M_1) = P(M_2) = 0.5$. Also note that $P(A|s,M_1) = \theta_s(A)$, and we are assuming here that an agent is lost if its action seems to be uniformly drawn at random.

3.7.1.3 Complete Procedure

The advising procedure consists of an offline training phase, and an online advising phase. These two phases may run concurrently, but whereas the online phase may last for only a single task, the training phase happens over a considerably longer duration, consisting of numerous agent interactions with the domain. The full system is shown in Figure 3.19.

- Offline: train the advisor. During this phase, trajectories are collected from a number of different agents carrying out various tasks in the same domain. From these, the advisor learns a model of typical behaviour in this domain, in the form of action priors (see Section 3.2).

- Online: a new agent, the explorer, begins execution of some unknown task. The advisor continually evaluates the trajectory of the explorer, and if Condition (3.16) is satisfied, the advisor provides advice to the explorer in the form of the action prior for the current state.

3.7.2 Experiments

3.7.2.1 Maze Domain

For these experiments we use a maze domain, being a 30×30 cell grid world through which an agent navigates by moving at each time step in one of the four cardinal directions. The maze contains many impassable corridors, emulating the layout of an extremely complicated building, which may have equipment lying about acting as obstructions (partially causing the complications). Furthermore, there are eight different goal locations (randomly placed in either a corner or the end of a corridor), and each

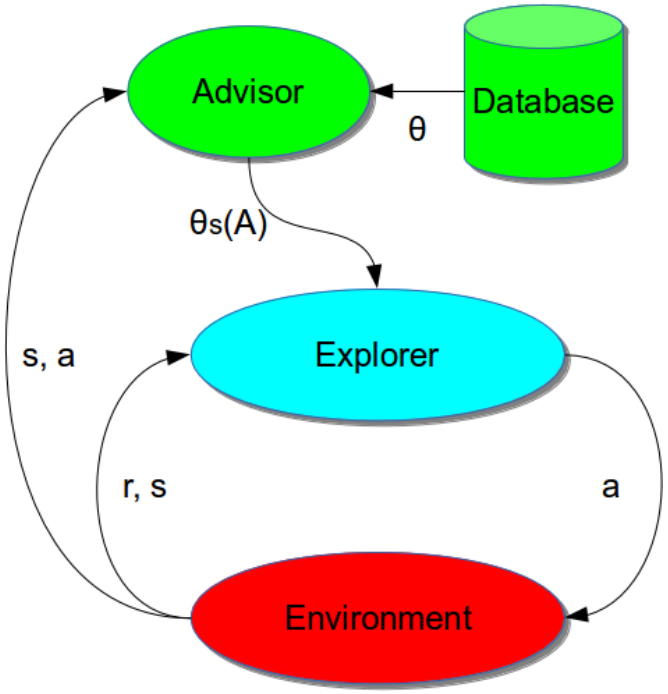

Figure 3.19: The proposed system. The explorer interacts with the environment by executing an action a, and receiving a new state s and possibly a reward r. The advisor observes the agent, by receiving a copy of s and a. It evaluates the trajectory taken by the explorer, by comparing it to the action prior θ obtained from a database of previous behaviours. If advice must be given, it is provided to the explorer in the form of the distribution $\theta_s(A)$.

agent is tasked with reaching a particular one of these. This is a delayed reward problem, with the agent receiving a reward of 1000 for reaching the goal, and 0 otherwise. The maze is depicted in Figure 3.20.

3.7.2.2 Agents

The offline training data which is provided to the advisor, is a set of expert trajectories from agents moving towards each of the eight goals. These trajectories were optimal for their respective tasks and were learnt using Q-learning. From these the action prior model M_1 is learnt, to model "normal" behaviour in the domain.

Online experiments involve different types of explorers entering the maze. Each agent has a randomly selected goal location. As a baseline, we consider the performance of an agent following an optimal trajectory. We do not know, *a priori*, how people behave and do not want to assume that people can easily zero in on a goal and execute an optimal policy. We thus experiment with various behaviour patterns to

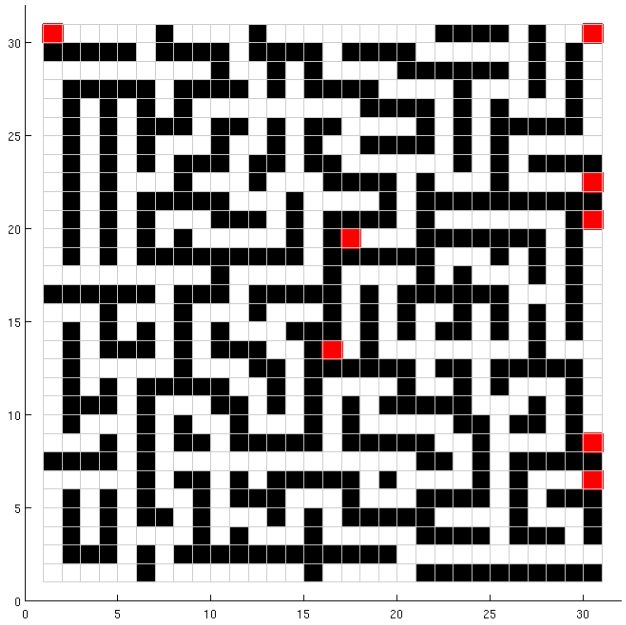

Figure 3.20: The maze used in these experiments. White cells denote free space, and black cells are obstacles. Goal locations are the eight cells marked in red.

clarify how the algorithm behaves, although not all accurately model how humans and animals search. We consider the effects of advice on the following user models:

- Completely random: at every time step pick a cardinal direction at random and attempt to move in that direction.
- Random with obstacle avoidance: select actions at random, from those which will not result in a collision.
- Always turn left: follow the well-known heuristic for navigating mazes, by always taking the leftmost path at any intersection.
- Goal seeking bug: heuristically move in the direction which will most reduce the difference in x- and y-coordinates between the agent and its goal.
- Q-learning: this non-stationary agent learns using ε-greedy Q-learning (Sutton and Barto, 1998).

All of the fixed-policy agents will select an action according to the offered distribution when advice is given. The Q-learning agent uses the advice as an exploration policy with probability ε, as described in Algorithm 2.

Agent	**Without Advice**	**With Advice**
Random motion	50	900
Random with obstacle avoidance	50	750
Always turn left	150	750
Goal seeking bug	100	800
Q-learning	150	1000
Optimal trajectories	*1000*	*1000*

Table 3.1: Performance of various agents on the maze domain

Each fixed-policy agent performs 20 random one-shot tasks of up to 5,000 steps, with and without advice. The learning agents learn for 5,000 episodes, where each episode is limited to only 200 steps, making this a difficult learning task.

3.7.2.3 Results

The results for the different agents with and without advice are shown in Table 3.1.

Performance curves and probabilities that the agent is acting according to M_1 are shown in Figure 3.21 for the fixed-policy agents, and reward curves in Figure 3.22 for the learning agents. The probabilities in Condition (3.16) are computed over a moving window considering only the most recent 300 time steps of the transition history H of the explorer, in order to avoid underflow.

Figure 3.21 shows the improvement in performance when advice is given to the different types of agents, over performance without advice. Although the agents all follow different action selection procedures, injection of advice results in them resembling normal behaviour in the domain more closely, and subsequently improving performance significantly.

In Figure 3.22, one can notice a similar improvement in the performance of Q-learning by guiding the learning agent towards sensible goal locations. The difficulty with a domain such as that in Figure 3.20 is that a specific sequence of correct actions is required, and the standard learning agent does not receive information to guide its progress. Using advice, the learning agent can discover the correct goal location within a handful of episodes, something which is not possible otherwise.

Figure 3.23 shows the amount of advice given to the agents as κ in Condition (3.16) is varied. κ is the cost of giving advice, to both the explorer and the advisor. As κ is increased, the quantity of advice provided decreases. Although not included

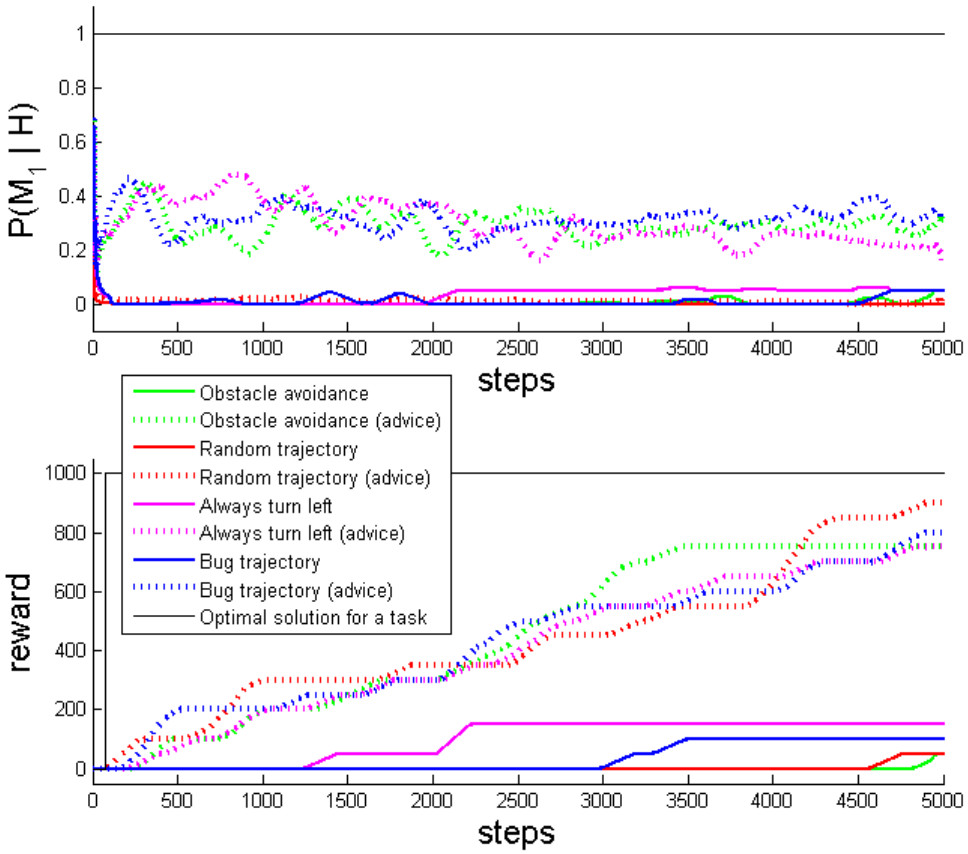

Figure 3.21: $P(M_1|H)$ and rewards of different fixed-policy agents, averaged over 20 random tasks with $\kappa = 0.5$. Solid lines show non-advised performance, and dotted lines are advised. Results are smoothed for readability.

here, we note that the Q-learning agent behaves initially as the random agent, and after convergence behaves as optimal.

3.7.3 Related Work

A similar and related problem to our own is that of one learning agent teaching another to perform a particular task (Torrey and Taylor, 2013). Our problem differs, as our domain supports a number of different tasks, and so the advisor does not know the explorer's goal. Advice is more useful in our case as a distribution over different possibilities. Further, our problem does not explicitly restrict the amount of advice to a constant, which seems unnatural, but instead computes the trade-off in utility from giving the advice.

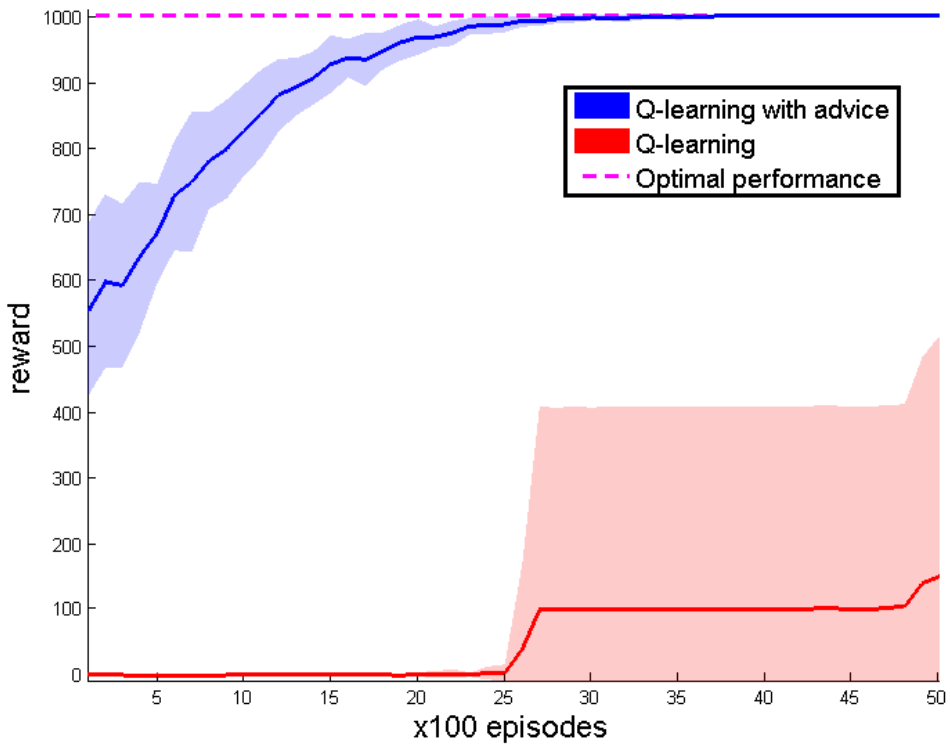

Figure 3.22: Performance from advice giving with a learning agent, averaged over 20 tasks with $\kappa = 0.5$. The shaded region denotes one standard deviation.

An alternative approach to learning a model of common sense behaviour in the domain, would be learning latent variable models to infer the current activities. One method is to learn the typical behaviours of individual agents (e.g. Liao et al. (2007)), but our application assumes many unique users who are not active in the system for long enough to be modelled. Instead one could also learn models of all the different tasks in the domain, using e.g. hidden-goal MDPs. Again, we include the possibility in our problem of there being a large number of goals, and furthermore providing advice from hidden-goal MDPs is PSPACE-complete (Fern and Tadepalli, 2010a).

Our work is very similar in spirit to that of Price and Boutilier (2003) who, through the use of a Bayesian model, allow a learning agent to update beliefs over the MDP dynamics by observing an expert in the domain. The primary difference to our application, is that we do not provide the expert trajectories in their entirety to the learner. Instead, the advisor gathers this information (from multiple experts) and is able to supply localised pieces of this knowledge to the agent.

Having one agent advise another is a form of transfer learning (Taylor and Stone,

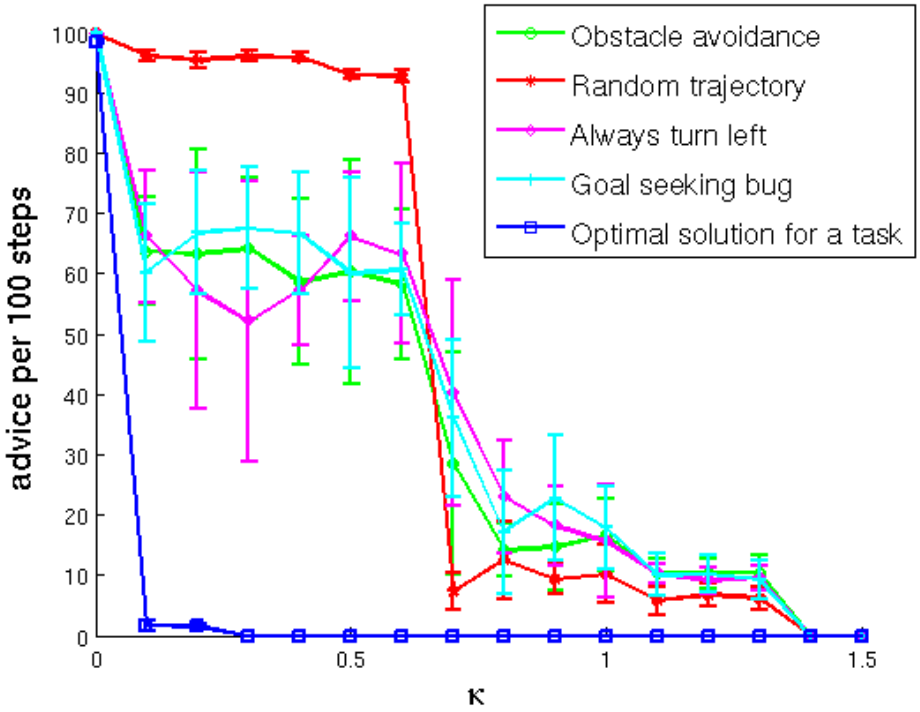

Figure 3.23: Amount of advice given to different agents per 100 steps as a function of κ. $\kappa = 0$ implies that advice is always given. 10 trajectories of 1000 time steps were generated for each agent at each value of κ. The error bars denote one standard deviation.

2009), in that it is a mechanism for simplifying and speeding up the learning of a new task by using the expertise of one agent to bootstrap another. There are many other approaches to transferring knowledge in this way, with two common restrictions: 1) the advisor and learner agree on the task and goal, and 2) the amount of teaching/advice is unlimited. Common inter-agent teaching methods include imitation learning, or learning from demonstration (Argall et al., 2009; Meltzoff et al., 2009), and in particular to the reinforcement learning case, apprenticeship learning (Abbeel and Ng, 2004; Syed and Schapire, 2008; Rosman and Ramamoorthy, 2010). Finally, a teacher may also instruct a learner in a more passive manner by deconstructing the main task into a sequence of simpler tasks of increasing difficulty. This approach is typically referred to as reward shaping or curriculum learning (Konidaris and Barto, 2006; Erez and Smart, 2008; Bengio et al., 2009; Knox and Stone, 2009).

3.8 Conclusion

This chapter introduces the concept of action priors, as distributions over the action set of an agent, conditioned on either the state or observations received by the agent. This provides a mechanism whereby an agent can, over the course of a lifetime, accumulate general knowledge about a domain in the form of local behavioural invariances. This is an important form of knowledge transfer, as it allows for a decision making agent to bias its search process towards previously useful choices.

The same principle demonstrated here in the context of reinforcement learning could additionally be applied to planning, similar to the idea of learning initial conditions (Schmill et al., 2000), and the action priors could be used to either prune certain actions or provide a preference list for depth-first search. In either case, this should guide the search toward solutions that have been previously encountered, allowing a deeper focused search with minimal additional computational overhead.

Furthermore, the principle of action priors could also be applied to online and bandit algorithms. This would have the positive effect of enabling features of the problem to propose an initial distribution over the value of the arms, which in turn could lead to faster convergence with less exploration.

Action priors also have the potential to be useful to collections of agents. Learning action priors constitutes learning "common sense" behaviour in a domain. By acquiring this information from multiple agents performing different tasks, an external agent can build a model which can then be used to provide advice and training to new agents, even when their specific goals are unknown.

We have shown that action priors are useful in practice, and they draw parallels to the way in which humans prune action choices automatically when making decisions. Hopefully further work in this direction can serve to strengthen this connection of accelerating learning between human and artificial decision making.

Chapter 4

Bayesian Policy Reuse

Work presented in this chapter was done in collaboration with Majd Hawasly, M. M. Hassan Mahmud and Pushmeet Kohli.

Parts of the work presented in this chapter also appear in: Mahmud et al. (2014).

4.1 Introduction

Online personalisation is becoming a core concept both in human-robot and human-computer interaction, driven largely by a proliferation of new sensors and input devices which allow for a more natural means of communicating with hardware. Consider for example an interactive interface, such as might be used for tele-operation of a robot, which interprets the gestures of a human user, set up in a public space such as a museum or a store. The device is required to map human movement to interpretable instructions, so as to provide the user with information or other services. The difficulty in this setting is that the same device may be expected to interact with a wide and diverse pool of users, who differ both at the low level of interaction speeds and body sizes, and at the higher level of which gestures seem appropriate for particular commands. The device should autonomously calibrate itself to the class of user, and a mismatch in intentions could result in a failed interaction. On the other hand, taking too long to calibrate is likely to frustrate the user, who may then abandon the interaction.

This problem can be characterised as a short-term interactive adaptation to a new situation. This problem appears in interactive situations other than interaction with humans. As an example, consider a system for localising and monitoring poachers in a large wildlife reserve. The agent in this setting is an intelligent base station which can deploy light-weight drones to scan particular locations in the park for unusual activity.

The number of drone deployments in this problem is limited, as the poachers can be expected to spend a limited time stalking their target before leaving.

The key component of the aforementioned problems is the need for sample-efficient decision making, as the agent is required to adapt or respond to scenarios before they have ended. The interaction is inherently short, so the agent needs to perform well in the limited time available. Then, solution methods need to have both low sample complexity and low regret. Thus, the question we address in this chapter is how to *act well* in a *sample-efficient* manner in a large space of possible tasks.

While it is unreasonable to assume that any new task instance could be solved from scratch in the constrained interaction time, it is plausible to consider seeding the process with a set of policies of solved instances, in what can be seen as a strategy for transfer learning (Taylor and Stone, 2009). For example, the interactive interface may ship with a set of different classes of user profiles which have been acquired offline, and the monitoring system may have a collection of pre-learnt behaviours to navigate the park when a warning is issued.

We term this problem of short-lived sequential policy selection for a new instance the *policy reuse* problem, and we define it formally as follows. Let an agent be a decision making entity in a specific domain, and let it be equipped with a policy library Π for tasks in that domain. The agent is presented with an unknown task which must be solved within a limited and small number of trials. At the beginning of each trial episode, the agent can select one policy from Π to execute for the full episode. The goal of the agent is thus to

> *select policies to minimise the total regret incurred in the limited task duration with respect to the performance of the best alternative from* Π *in hindsight.*

The online choice from a set of alternatives for minimal regret in a new situation could be posed as a multi-armed bandit. Here, each arm would correspond to a pre-learnt policy, and our problem becomes a sequential optimal selection of fixed policies. Solving this problem in general is difficult as it maps into the intractable finite-horizon online bandit problem (Niño-Mora, 2011). On the other hand, traditional bandit approaches have to try each available arm on the new task in order to gauge its performance, which may be a very costly procedure from a sample complexity point of view.

Instead, one can exploit knowledge which has been acquired offline to accelerate online response times. We propose a solution for policy reuse that exploits the natural correlation between policies captured offline through testing under canonical operating

conditions, by maintaining a Bayesian belief over the nature of the new task instance in relation to the seen ones. Then, executing a policy provides the agent with information not only about the policy chosen but also about the suitability of all policies. This information is used to update the belief, which facilitates choosing the next policy to execute.

A version of the policy reuse problem appears in Mahmud et al. (2013), where it is used to test a set of landmark policies retrieved through clustering in the space of MDPs. The term 'policy reuse' was used by Fernandez and Veloso (2006) in a different context. There, a learning agent is equipped with a library of previous policies to aid in exploration, as they enable the agent to collect relevant information quickly to accelerate learning. In our case, learning is infeasible, and policy reuse is the only way to achieve the objective of the agent.

4.1.1 Contributions

The primary contributions made in this chapter are as follows:

1. We introduce Bayesian Policy Reuse (BPR) as a general Bayesian framework for solving the policy reuse problem (Section 4.2).

2. We present several specific instantiations of BPR using different policy selection mechanisms (Section 4.5.2), and compare them on a domain modelling a surveillance problem.

3. We provide an empirical analysis of the components of our model, considering different classes of observation signal, and the trade-off between library size and sample complexity.

4. We present an application of interface adaptation, with a variant of the BPR algorithm known as BEAT for solving this problem.

4.2 Bayesian Policy Reuse

We now pose the policy reuse transfer problem in a Bayesian framework. Throughout the discussion, we adopt the following notational conventions: $\mathrm{P}(\cdot)$ refers to a probability, $\mathrm{E}[\cdot]$ refers to an expectation, $\mathrm{H}(\cdot)$ refers to entropy, and $\Delta(\cdot)$ is a distribution.

For a set of tasks $\mathcal{X}$ and a set of policies Π, Bayesian policy reuse involves two key components. The first, $\mathrm{P}(U|\mathcal{X},\Pi)$, where U is utility, is the *performance model*; a probability model over performance of the set of policies Π on the set of seen tasks $\mathcal{X}$. This information is available in an offline phase. The second key component is the *observation model*, defined as a probability distribution $\mathrm{P}(\Sigma|\mathcal{X},\Pi)$ over Σ, the space of possible observation signals which can be provided by the environment. This latter distribution describes some kind of information which is correlated with the performance and that can be observed online. When the performance information is directly observable online, the observation model and the performance model may be essentially the same.

A 1-D caricature of the Bayesian policy reuse problem is depicted in Figure 4.1, where, given a new task $x^* \in \mathcal{X}$, the agent is required to select the best policy $\pi^* \in \Pi$ in as few trials as possible, whilst accumulating as little regret as possible in the interim. The agent has prior knowledge only in the form of performance models for each policy in Π on a set of tasks from $\mathcal{X}$ (as well as observation models, not depicted here).

Having an observation model means that applying a known policy on a new task instance provides the agent with a sample observational *signal*, which is used to update a 'similarity' measure, the belief, over the seen tasks. This belief allows for the selection of a policy at the next time step, so as to optimise expected performance. This is the core idea behind Bayesian policy reuse.

We present the general form of Bayesian Policy Reuse (BPR) in Algorithm 7.

Algorithm 7 Bayesian Policy Reuse (BPR)

Require: Problem space $\mathcal{X}$, policy library Π, observation space Σ, prior over the problem space $\mathrm{P}(\mathcal{X})$, observation model $\mathrm{P}(\Sigma|\mathcal{X},\Pi)$, performance model $\mathrm{P}(U|\mathcal{X},\Pi)$, number of episodes K.

1: Initialise beliefs: $\beta^0(\mathcal{X}) \longleftarrow \mathrm{P}(\mathcal{X})$.
2: **for** episodes $t = 1 \ldots K$ **do**
3: Select a policy $\pi^t \in \Pi$, using the current belief of the task β^{t-1}.
4: Apply π^t on the task instance.
5: Obtain an observation signal σ^t from the environment.
6: Update the belief $\beta^t(\mathcal{X}) \propto \mathrm{P}(\sigma^t|\mathcal{X},\pi^t)\beta^{t-1}(\mathcal{X})$.
7: **end for**

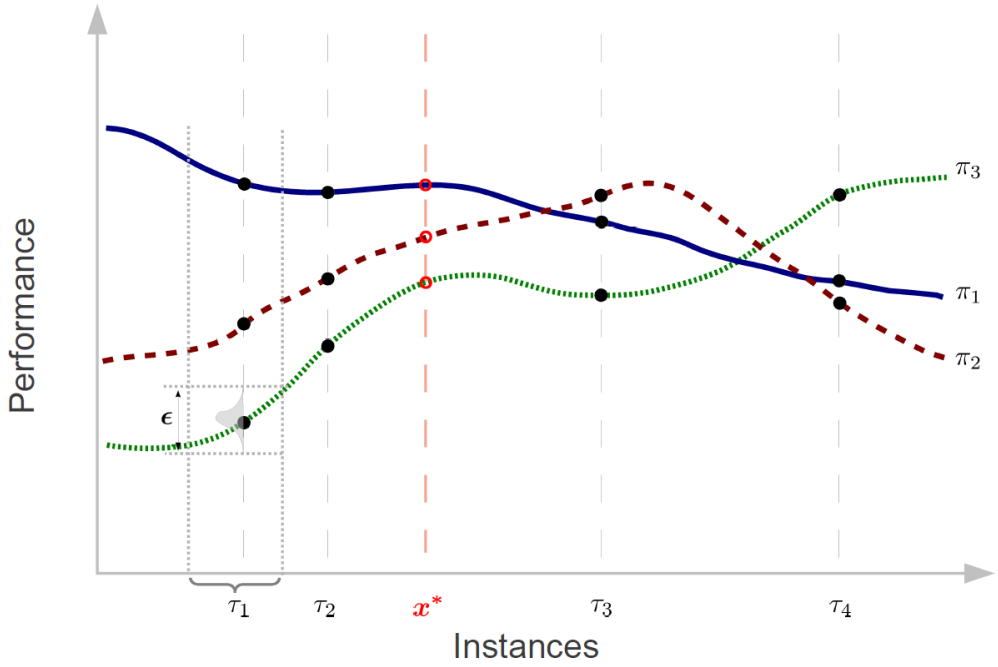

Figure 4.1: A simplified depiction of the Bayesian Policy Reuse problem. The agent has access to a library of policies (π_1, π_2 and π_3 in the figure), and has previously experienced a set of task instances ($\tau_1 \ldots \tau_4$), as well as samples of the utilities of the library policies on these instances (the black dots are the means of these estimates, while the agent maintains distributions of the utility as illustrated by $\mathrm{P}(U|\tau_1, \pi_3)$). The agent is presented with a new unknown task instance (x^*), and it is asked to select the best policy from the library (optimising between the red hollow points) without having to try every individual option (in less than 3 trials in this example). The agent has no knowledge about either the complete curves, or where the task instances occur in the problem space, but tries to infer this from utility similarity. Not depicted here are the observation models.

4.2.1 Chapter Organisation

The problem space $\mathcal{X}$ is first defined in Section 4.3. The performance model $\mathrm{P}(U|\mathcal{X},\Pi)$ and the observation model $\mathrm{P}(\Sigma|\mathcal{X},\Pi)$ are discussed further in Sections 4.3.4 and 4.4 respectively. The two core steps of BPR are the Bayesian update of beliefs over seen tasks (line 6 of Algorithm 7), discussed in more detail in Section 4.4, and selecting a policy to reuse (line 3 of Algorithm 7), detailed in Section 4.5.

4.3 Problem Space

4.3.1 Tasks

Let a task be specified by a Markov Decision Process (MDP). An MDP is defined as a tuple $\mu = (S,A,T,R,\gamma)$, where S is a finite set of states; A is a finite set of actions which can be taken by the agent; $T : S \times A \times S \to [0,1]$ is the state transition function where $T(s,a,s')$ gives the probability of transitioning from state s to state s' after taking action a; $R : S \times A \times S \to \mathbb{R}$ is the reward function, where $R(s,a,s')$ is the reward received by the agent when transitioning from state s to s' with action a; and finally, $\gamma \in [0,1]$ is a discounting factor. As T is a probability function, $\sum_{s' \in S} T(s,a,s') = 1, \forall a \in A, \forall s \in S$. Denote the space of all MDPs $\mathcal{M}$. We will consider episodic tasks, i.e. tasks that have a specified time horizon.

A policy $\pi : S \times A \to [0,1]$ for an MDP is a mapping from states to actions, which describes the probability of taking any action from a state. The return, or utility, generated from running the policy π on a task instance x is the accumulated discounted reward $U_x^\pi = \sum_{i=0}^{\infty} \gamma^i r_i$, for r_i being the reward received at step i. We omit the subscript x when the task identity is unambiguous. If the task is episodic, $U^\pi = \sum_{i=0}^{K} \gamma^i r_i$, with K being the length of the episode. The goal of reinforcement learning is to learn an optimal policy $\pi^* = \arg\max_\pi U^\pi$ which maximises the total expected return of an MDP μ, where typically T and R are unknown.

We denote a collection of policies known by the agent as Π, and refer to it as the policy library.

4.3.2 Regret

In order to evaluate the performance of our approach, we define *regret* as the policy selection metric to be optimised by Bayesian Policy Reuse.

Definition 4.3.1 (Library Regret)**.** For a library of policies Π and for a policy selection algorithm $\xi : \mathcal{X} \to \Pi$ that selects a policy for the new task instance $x^* \in \mathcal{X}$, the *library regret* of ξ is defined as

$$\mathcal{R}(\xi) = U_{x^*}^{\pi^*} - U_{x^*}^{\xi(x^*)},$$

where $\pi^* = \arg\max_{\pi \in \Pi} U_{x^*}^{\pi}$, the best policy in hindsight *in the library* for the task instance x^*, and $U_{x^*}^{\pi^*} = \max_{\pi \in \Pi} U_{x^*}^{\pi}$ is the best performance that can be achieved on task x^* with the library Π.

Definition 4.3.2 (Average Library Regret)**.** For a library of policies Π and for a policy selection algorithm $\xi : \mathcal{X} \to \Pi$, the *average library regret* of ξ over K trials is defined as the average of the library regrets for the individual trials,

$$\mathcal{R}^K(\xi) \quad = \quad \frac{1}{K} \sum_{t=1}^{K} \mathcal{R}(\xi^t).$$

The metric we minimise in BPR is the average library regret $\mathcal{R}^K(\xi)$ for K trials. That is, the goal of BPR is to not only find the right solution at the end of the K trials, possibly through expensive exploration endeavours, but also to optimise performance even when exploring in the short duration of the task.

We will refer to this metric simply as 'regret' throughout the rest of the chapter.

4.3.3 Types

When the problem space of BPR is the task space $\mathcal{M}$, it is very likely that maintaining a belief would require a large number of samples, and would hence be expensive to maintain and to compute with. Depending on the application, we may find that there is a natural notion of clustering in $\mathcal{M}$, where many tasks are similar with minor variations. Previous work has looked into finding the clustering in a space of tasks; see for example Mahmud et al. (2013) for a detailed account. We do not try to discover the clustering in this work, but we assume it exists.

To this end, we introduce a simpler notion of types as ε-balls of tasks in task space, where we cluster with respect to the performance of a collection of policies on the tasks. Another way to define types is the notion of classes of tasks as probability distributions over task parameters as used by Wilson et al. (2007).

Definition 4.3.3 (Type)**.** A type τ is a subset of tasks such that for any two tasks μ_i, μ_j from a type τ, and for all policies π in a set of policies Π, the difference in utility is

upper-bounded by some $\varepsilon \in \mathbb{R}$:

$$\mu_i, \mu_j \in \tau \Leftrightarrow |U_i^\pi - U_j^\pi| \leq \varepsilon, \ \ \forall \pi \in \Pi,$$

where $U_i^\pi \in \mathbb{R}$ is the utility from executing policy π on task μ_i. Then, μ_i and μ_j are ε-*equivalent* under the policies Π.

Figure 4.1 depicts four example types, where each accounts for an ε region in performance space, as explicitly shown for τ_1.

Note that these types do not have to be disjoint, i.e. there may exist tasks that belong to multiple types. We denote the space of types with $\mathcal{T}$. Assuming disjoint types, we can work with $\mathcal{T}$ as the problem space of BPR, inducing a hierarchical structure in the space $\mathcal{M}$. The environment can then be represented with the generative model in Figure 4.2(a) where a type τ is drawn from a hyperprior $\tau \sim G_0$, and then a task is drawn from that type $\mu \sim \Delta^\tau(\mu)$. By definition, the set of MDPs generated by a single type are ε-equivalent under Π, hence the BPR regret of representing all the MDPs in τ with any one of them cannot be more than ε. Let us call that chosen MDP a *landmark MDP* of type τ, and denote this by μ_τ. This would reduce the hierarchical structure into the approximate model shown in Figure 4.2(b), where the prior acts immediately on a set of landmark MDPs μ_τ, $\tau \in \mathcal{T}$. The benefit of this for BPR is that each individual stored alternative of the seen tasks is a representative for a region in the original task space, defined by a maximum loss of ε. Maintaining only this set of landmarks removes near duplicate tasks from consideration.

For the remainder of this chapter, we will use the type space $\mathcal{T}$ as the problem space, although we note that the methods we propose would allow the full task space $\mathcal{M}$ to be alternatively used.

4.3.4 Performance Model

One of the key components of BPR is the performance model of policies on task instances, which holds the distribution of returns of a policy for the seen tasks. Using types, a performance model represents the variability in return under the various tasks in a type.

Definition 4.3.4 (Performance Model)**.** For a policy π and a type τ, the *performance model* is a probability distribution over the utility of π when applied to all tasks $\mu \in \tau$.

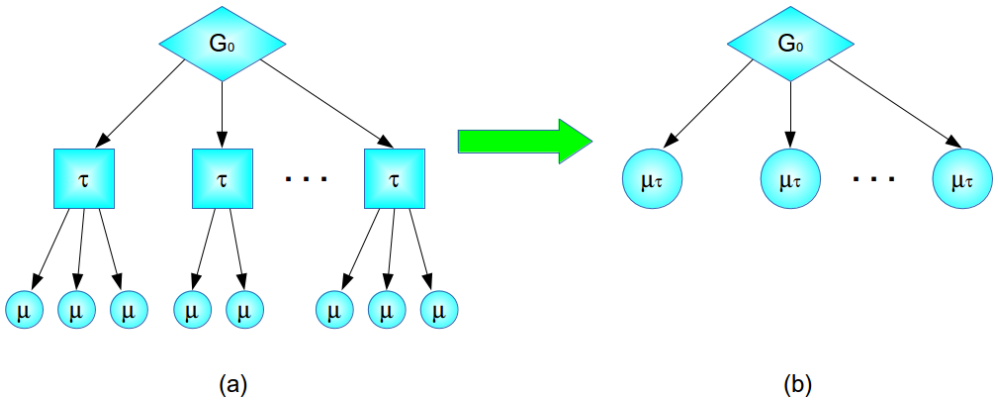

Figure 4.2: Problem space abstraction model under disjoint types. (a) Tasks μ are related by types τ, with a generating distribution G_0 over them. (b) A simplification of the hierarchical structure under ε-equivalence. The tasks of each type are represented by a single task μ_τ.

Figure 4.1 depicts the performance profile for π_3 on type τ_1 in the form of a Gaussian distribution. Recall that for a single type, and each policy, the domain of the performance model would be of size ε or less. The agent maintains performance models for all the policies it has in the library Π and for all the types it has experienced.

4.4 Observation Signals and Beliefs

Definition 4.4.1 (Signal)**.** A *signal* $\sigma \in \Sigma$ is any information which is correlated to the performance of a policy and which is provided to the agent in an online execution of the policy on a task.

The most straightforward signal is the performance itself, unless this is not directly observable (e.g. in cases where the payoff may only be known after a long time horizon). However, the information content and richness of a signal determines how easily an agent can identify the type of the new task with respect to the seen types. In order to perform this identification, the agent learns a model of how types, policies and signals relate in the offline phase.

4.4.1 Observation Model

For some choice of signal space, the agent learns canonical models of the signal values expected from every type-policy pair.

Definition 4.4.2 (Observation Model)**.** For a policy π and type τ, the *observation model* is a probability distribution over the signals that may result by applying that policy to that type, $\mathcal{F}_\pi^\tau(\sigma) = \mathrm{P}(\sigma|\tau,\pi)$.

We consider the following offline procedure to learn the signal models for a policy library Π:

1. The type label τ is announced.
2. A set of tasks are generated from the type τ.
3. The agent runs all the policies from the library Π on all the instances of τ, and observes the resultant signals.
4. Distributions $\mathcal{F}_\pi^\tau = \Delta(\sigma)$ are fitted to the data.

The benefit of these models is that they connect observable online information to the latent type label. Identifying the true type (or the most similar one) of the new instance is sufficient to play a good policy from the policy library, which is the goal of the policy reuse problem.

4.4.2 Some Candidate Signals

4.4.2.1 State-Action-State Tuples

The richest information signal which could be accrued by the agent is the history of all (s,a,s') tuples encountered during the execution of a policy. Thus, the observation model in this case is exactly the expected transition function of the MDPs under the type τ, even though the agent would only learn part of that model to which it has access in the offline phase. The expressiveness of this signal does have a drawback, in that it is expensive to learn and maintain these models for every possible type.

This form of signal is useful in cases where some environmental factors may affect the behaviour of the agent in a way that does not directly relate to attaining an episodic goal.

4.4.2.2 Instantaneous Rewards

Another form of information is the instantaneous reward $r \in \mathbb{R}$ received during the execution of a policy, and hence the observation model is the expected reward function

for the MDPs in the type. This may provide a relatively fine-grained knowledge on the behaviour of the task when intermediate rewards are informative.

This is a more abstracted signal than the state-action-state tuples, and is likely to be useful in scenarios where the task has a number of subcomponents which individually contribute to overall performance.

4.4.2.3 Episodic Returns

An example of a sparser signal is the total utility $U_\tau^\pi \in \mathbb{R}$ accrued over the full episode of using a policy in a task. This signal is useful for problems of delayed reward, where intermediate states cannot be valued easily, but the extent to which the task was successfully completed defines the return. The observation model of such a scalar signal is much more compact, and thereby easier to learn, than the previous two proposals. We also note that for our envisioned applications, the execution of a policy cannot be terminated prematurely, meaning that an episodic return signal is always available to the agent before selecting a new policy.

In our framework, using episodic returns as signals has the additional advantage that this information is already captured in the performance model, which relieves the agent from maintaining two separate models, as $\mathrm{P}(U|\tau,\pi) = \mathcal{F}_\pi^\tau(U)$ for all π and τ.

4.4.3 Bayesian Belief over Types

Definition 4.4.3 (Type Belief)**.** For a set of seen types $\mathcal{T}$ and a new instance x^*, the *Type Belief* is a probability distribution over $\mathcal{T}$ that measures to what extent x^* matches the types in their observation signals.

The type belief, or *belief* for short, is a surrogate measure of similarity in type space. It approximates where a new instance may be located in relation to the known types, which act as bases of the unknown type space. Belief is initialised with the prior probability over the environment generating an instance of each of the types, labelled G_0 in Figure 4.2.

After each execution of a policy on the new unknown task, the environment provides an observation signal to the agent, which is used to update beliefs. Line 6 in Algorithm 7 computes the posterior probability over the task space $\beta^t(\mathcal{T})$ using Bayes'

rule:

$$\beta^t(\tau) = \frac{\mathrm{P}(\sigma^t|\tau,\pi^t)\beta^{t-1}(\tau)}{\sum_{\tau'\in\mathcal{T}}\mathrm{P}(\sigma^t|\tau',\pi^t)\beta^{t-1}(\tau')} \tag{4.1}$$

$$= \frac{\mathcal{F}^{\tau}_{\pi^t}(\sigma^t)\beta^{t-1}(\tau)}{\sum_{\tau'\in\mathcal{T}}\mathcal{F}^{\tau'}_{\pi^t}(\sigma^t)\beta^{t-1}(\tau')}, \quad \forall\tau\in\mathcal{T}, \tag{4.2}$$

where π^t is the policy played in episode t, and σ^t is the signal received thereafter.

We use β to refer to β^t where this is not ambiguous.

4.5 Policy Selection

Given the current type belief, the agent is required to choose a policy for the next episode for two concurrent purposes: to acquire useful information about the new instance, and at the same time to avoid accumulating additional regret.

At the core of this policy selection problem is the trade-off between exploration and exploitation, an important problem since early work on the analysis of Markov chains and decision processes (Martin, 1965). When a policy is executed at some time t, the agent receives both some utility and information about the true type of the new task. We wish to gain as much information about the task as possible, so as to choose policies optimally in the future, but at the same time minimise performance losses due to sub-optimal[1] policy choices. We can define the following MDP to describe the process:

- The states are the continuous belief states $\beta\in[0,1]^{|\mathcal{T}|-1}$.
- The actions are the available policies $\pi\in\Pi$.
- The reward process is defined by the utility of the unknown test type τ^* after execution of a policy π, $U\sim\mathrm{P}(U|\tau^*,\pi)$.

Under this formulation, the value of a policy choice π is given by

$$Q(\beta,\pi) = \int_{U\in\mathbb{R}}\mathrm{P}(U|\tau^*,\pi)\left(U+\max_{\pi'\in\Pi}Q(\beta^U,\pi')\right)\mathrm{d}U, \tag{4.3}$$

where β^U is the belief after incorporating the observed utility U. Then, the optimal policy selection for some belief β is the greedy optimisation, $\pi^* = \arg\max_{\pi\in\Pi}Q(\beta,\pi)$.

[1] Note that we call the best policy in the library for a specific instance its *optimal policy*, as we are not considering problems with time feasible for learning.

Computing the Q-function in Equation (4.3) directly is not possible for two reasons. Firstly, the performance profile $P(U|\tau^*,\pi)$ is not explicitly known for the true target type τ^* *a priori*, and thus neither is β^U. One possible solution is to approximate this with the expected performance under the belief β, $\hat{P}(U|\tau^*,\pi) = \sum_{\tau\in\mathcal{T}} \beta(\tau_i)P(U|\tau_i,\pi)$. This is the Bayesian optimum with respect to the current belief, but that still represents an approximation to the true performance profile that would be generated by type τ^*. Secondly, even if we approximate the performance profile, the state in Equation (4.3) is continuous and hence discretisation or function approximation is needed, which does not generalise well to different problem settings.

We now propose several policy selection mechanisms for dealing with this problem. A first approach is through ε-greedy exploration, where with probability $1-\varepsilon$ we choose the policy which maximises the expected utility under the belief β, and with probability ε we choose a policy from the policy library uniformly at random. This additional random exploration component perturbs the belief from local minima.

A second approach is through sampling the belief β. This involves sampling a type from the belief, and playing the best response to that type from the policy. In this case, the sampled type acts as an approximation of the true unknown type, and exploration is achieved through the sampling process.

The third approach to this problem is through the method of employing heuristics that estimate a value for each policy, so as to achieve a balance between exploitation and a limited degree of look-ahead to approximate optimal exploration. This approach is discussed further in Section 4.5.1.

4.5.1 Exploration Heuristics

The goal of the agent is to maximise utility over the full K episodes of the task. This corresponds to minimising the cumulative, rather than instantaneous, regret. A purely greedy policy selection mechanism would fail to choose exploratory options which return the information needed for the belief to converge to the closest type. That is, a greedy approach would be myopically but not asymptotically optimal, and may result in the agent becoming trapped in local maxima of the utility function. On the other hand, a purely exploratory policy selection mechanism could be designed to ensure that the most information possible is elicited in expectation, but this would not make an effort to improve performance. We thus require a mechanism to explore as well as exploit; find a better policy to maximise asymptotic utility, and exploit the current

estimates of which are good policies to maximise myopic utility.

Using the prevalent approach in Bayesian optimisation, we greedily maximise, instead of utility, a surrogate function[2] that takes into account both the expected return from executing a policy, with a notion of the variance of this return estimate (e.g. Brochu et al. (2010)). These functions have been widely considered in the multi-armed bandit (MAB) literature, and for the finite-horizon total-reward multi-armed bandit problem, Lai and Robbins (1978) show that index-based methods achieve optimal performance asymptotically. In general there are a large number of ways in which they can be defined, ranging from early examples such as the Gittins index (Gittins and Jones, 1974) (for infinite horizon problems) to more recent methods such as the knowledge gradient (Powell, 2010). The benefit of such methods is that they involve estimating a separate value of each policy (or arm, in the relevant literature), independent of other policies.

Although these different surrogate utility functions balance exploration and exploitation, in what follows we refer to them as *exploration heuristics*.

4.5.2 Bayesian Policy Reuse with Exploration Heuristics

By incorporating the notion of an exploration heuristic that computes an index ν_π for a policy π into Algorithm 7, we obtain the proto-algorithm Bayesian Policy Reuse with Exploration Heuristics (BPR-EH) described in Algorithm 8. Note that by defining the problem space to be the type space $\mathcal{T}$, we can use G_0 as the prior over types.

Next, we discuss some candidate heuristics $\mathcal{V}$ that can implement line 3 in the algorithm, and we define four variants of the BPR-EH algorithm, as

- BPR-PI using probability of improvement (Section 4.5.2.1),
- BPR-EI using expected improvement (Section 4.5.2.1),
- BPR-BE using belief entropy (Section 4.5.2.2),
- BPR-KG using knowledge gradient (Section 4.5.2.3).

We note that there are a number of such action selection mechanisms which could be used in place of $\mathcal{V}$, see e.g. Berry and Fristedt (1985) or Bubeck and Cesa-Bianchi (2012) for further examples. Algorithm 8 requires the use of some such selection

[2]This idea of forming a surrogate function is also known as adding an exploration bonus to a reward function (Wyatt, 1997).

Algorithm 8 Bayesian Policy Reuse with Exploration Heuristics (BPR-EH)

Require: Type space $\mathcal{T}$, Policy library Π, observation space Σ, prior over the type space $\mathrm{P}(\mathcal{T})$, observation model $\mathrm{P}(\Sigma|\mathcal{T},\Pi)$, performance model $\mathrm{P}(U|\mathcal{T},\Pi)$, number of episodes K, an exploration heuristic $\mathcal{V}$.

1: Initialise beliefs: $\beta^0(\mathcal{T}) \longleftarrow G_0$.
2: **for** episodes $t = 1 \ldots K$ **do**
3: Compute $\nu_\pi = \mathcal{V}(\pi, \beta^{t-1})$ for all $\pi \in \Pi$.
4: $\pi^t \longleftarrow \arg\max_{\pi\in\Pi} \nu_\pi$.
5: Apply π^t to the task instance.
6: Obtain the observation signal σ^t from the environment.
7: Update the belief β^t using σ^t by Equation (4.1).
8: **end for**

mechanism, but in general we are agnostic to the choice of this mechanism. The following four approaches are presented as examples.

4.5.2.1 Probability of Improvement and Expected Improvement

One heuristic for policy selection is through the probability with which a specific policy can achieve a hypothesised increase in performance. Assume that U^+ is some utility which is larger than the current best estimate under the current knowledge, $U^+ > U^t = \max_{\pi\in\Pi} \tilde{U}^t(\pi)$, where $\tilde{U}^t(\pi)$ is defined as

$$\tilde{U}^t(\pi) = \sum_{\tau\in\mathcal{T}} \beta^t(\tau) \int_{U^{min}}^{U^{max}} U\mathrm{P}(U|\tau,\pi)\mathrm{d}U \tag{4.4}$$

$$= \sum_{\tau\in\mathcal{T}} \beta^t(\tau)\mathrm{E}[U|\tau,\pi]. \tag{4.5}$$

The *probability of improvement* (PI) principle chooses the policy that maximises the term

$$\pi^t = \arg\max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau)\mathrm{P}(U^+|\tau,\pi).$$

The choice of U^+ is not straightforward, and this choice is the primary factor affecting the performance of this exploration principle. One approach to addressing this choice, is through the related idea of *expected improvement* (EI). This requires integration over all the possible values of improvement $U^t < U^+ < U^{max}$, and the policy is

chosen with respect to the best potential. That is,

$$\begin{aligned}
\pi^t &= \arg\max_{\pi\in\Pi} \int_{U^t}^{U^{max}} \sum_{\tau\in\mathcal{T}} \beta(\tau)\mathrm{P}(U^+|\tau,\pi)\mathrm{d}U^+ \\
&= \arg\max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau) \int_{U^t}^{U^{max}} \mathrm{P}(U^+|\tau,\pi)\mathrm{d}U^+ \\
&= \arg\max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau)(1-\mathrm{F}(U^t|\tau,\pi)) \\
&= \arg\min_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau)\mathrm{F}(U^t|\tau,\pi),
\end{aligned}$$

where $\mathrm{F}(U|\tau,\pi) = \int_{-\infty}^{U} \mathrm{P}(u|\tau,\pi)\mathrm{d}u$ is the cumulative distribution function of U for π and τ.

4.5.2.2 Belief Entropy

Both PI and EI principles select a policy which has the potential to achieve higher utility. An alternate approach is to select the policy which will have the greatest effect in reducing the uncertainty over the type space.

The *belief entropy* (BE) approach seeks to estimate the effect of each policy in reducing uncertainty over type space, represented by the entropy of the belief. For each policy $\pi \in \Pi$, estimate the expected entropy of the belief after executing π as

$$\mathrm{H}(\beta|\pi) = -\beta^\pi \log \beta^\pi,$$

where β^π is the updated belief after seeing the signal expected from running π:

$$\begin{aligned}
\beta^\pi(\tau) &= \mathrm{E}_\sigma\left[\frac{\mathcal{F}_\pi^\tau(\sigma)\beta(\tau)}{\sum_{\tau'\in\mathcal{T}} \mathcal{F}_\pi^{\tau'}(\sigma)\beta(\tau')}\right] \\
&= \int_{\sigma\in\Sigma} \mathcal{F}_\pi^\tau(\sigma) \frac{\mathcal{F}_\pi^\tau(\sigma)\beta(\tau)}{\sum_{\tau'\in\mathcal{T}} \mathcal{F}_\pi^{\tau'}(\sigma)\beta(\tau')}\mathrm{d}\sigma.
\end{aligned}$$

Then, selecting the policy

$$\pi^t = \arg\min_{\pi\in\Pi} \mathrm{H}(\beta|\pi)$$

reduces the most uncertainty in the belief in expectation. This is in essence a purely exploratory policy. To incorporate exploitation of the current state of knowledge, we rather select

$$\pi^t = \arg\max_{\pi\in\Pi} \left(\tilde{U}^t(\pi) - \kappa\mathrm{H}(\beta|\pi)\right),$$

where $\kappa \in \mathbb{R}$ is a positive constant controlling the exploration-exploitation trade-off, and $\tilde{U}^t(\pi)$ is the expected utility of π under the current belief, given by Equation 4.4.

4.5.2.3 Knowledge Gradient

Another approach is to use the *knowledge gradient* (KG), which aims to balance exploration and exploitation through the optimisation of myopic return whilst maintaining asymptotic optimality (Powell, 2010). The principle behind this approach is to estimate a one step look-ahead, and select the policy which maximises utility over both the current time step, and the next in terms of the information gained.

To select a policy using the knowledge gradient, we choose the policy π^t which maximises the online knowledge gradient at time t

$$\pi^t \quad = \quad \arg\max_{\pi\in\Pi} \left(\tilde{U}^t(\pi) + (K-t)\nu_\pi^{\mathrm{KG},t} \right),$$

trading-off between $\tilde{U}^t(\pi)$, of Equation 4.4, and $\nu_\pi^{\mathrm{KG},t}$, the offline knowledge gradient of π for a horizon of K trials, which essentially measures the performance of a one-step look-ahead in the process, given as

$$\nu_\pi^{\mathrm{KG},t} \quad = \quad \mathrm{E}_\beta \left[\max_{\pi'} \tilde{U}^{t+1}(\pi') - \max_{\pi'} \tilde{U}^t(\pi') \quad \middle| \quad \pi \right], \tag{4.6}$$

which is the difference in the expectation, with respect to β, of the best performance of any policy at $t+1$ if π was played at t, with that of the best policy at t. By marginalising over the signal, the first term in the expectation is

$$\tilde{U}^{t+1}(\pi')\,|\pi \quad = \quad \sum_{\tau\in\mathcal{T}} \beta^t(\tau) \int_{\sigma\in\Sigma} \mathcal{F}_\pi^\tau(\sigma) \sum_{\tau'\in\mathcal{T}} \beta^{t+1}(\tau') \mathrm{E}[U|\tau',\pi'] \mathrm{d}\sigma,$$

where β^{t+1} is the updated belief resulting from the previous belief β and the signal σ through Equation (4.1). The second term in the expectation of Equation (4.6) is given by Equation (4.4).

An approach to action selection which is similar in principle is *myopic value of perfect information* (Dearden et al., 1998), which estimates the balance of the expected gains which could occur if an improved policy is found during exploration, against the cost which would be incurred by selecting a suboptimal policy.

4.6 Experiments

4.6.1 Golf Club Selection

As an initial, illustrative experiment we consider the problem of a simulated robot golfer taking a shot on an unknown golf course, where it *cannot reliably estimate the*

distance to the hole. The robot is only allowed to take $K = 3$ shots (less than the number of clubs) from a fixed position from the hole, and the task is evaluated by how close to the hole the ball ends. The robot can choose any club from a set of available clubs, and assume it has a default canonical stroke with each club.

In this setting, we consider the type space $\mathcal{T}$ to be a set of different golfing experiences the robot had before, each defined by how far the target was (other factors, such as weather conditions, could be factored into this as well). The performance of a club for some hole is defined as the negative of the absolute distance to the hole, such that this quantity must be maximised.

Then, given the fixed stroke of the robot, the choice of a club corresponds to a policy. For each, the robot has a performance profile (in this case, the profile is over final distance of the ball from the hole) for the different courses that the robot experienced. We assume a small selection of four clubs, with properties shown in Table 4.1 for the robot canonical stroke. The distances shown in this table are the simulated ground truth values, and are not known to the robot.

Club	Average Yardage	Standard Deviation of Yardage
$\pi_1 =$ 3-wood	215	8.0
$\pi_2 =$ 3-iron	180	7.2
$\pi_3 =$ 6-iron	150	6.0
$\pi_4 =$ 9-iron	115	4.4

Table 4.1: Statistics of the ranges (yardage) of the four clubs used in the golf club selection experiment. We choose to model the performance of each club by a Gaussian distribution. We assume the robot is competent with each club, and so the standard deviation is small, but related to the distance hit.

The robot cannot measure distances in the field, but for a feedback signal, it can crudely estimate a qualitative description of the result of a shot as falling into one of several categories (such as *near* or *very far*). Note that this is not the performance signal, but is a weaker observation correlated with performance. The distributions over these qualitative categories (the observation models) are known to the agent for each club on each of the training types it has encountered. We assume the robot has extensive training on four particular holes, with distances $\tau_{110} = 110$yds, $\tau_{150} = 150$yds, $\tau_{170} = 170$yds and $\tau_{220} = 220$yds. The performance signals are shown in Figure 4.3.

When the robot faces a new hole, BPR allows the robot to overcome its inability

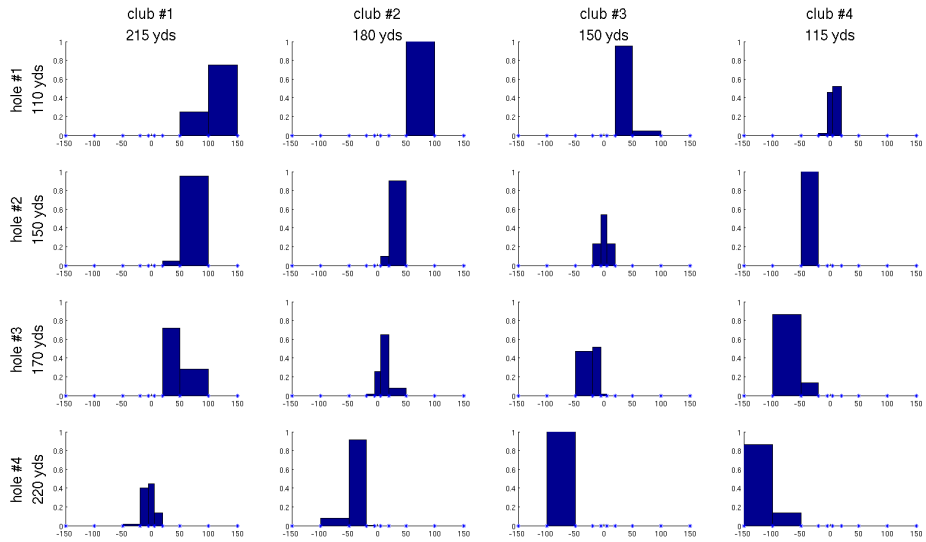

Figure 4.3: Performance signals for the four golf clubs in Table 4.1, on four training holes with distances 110yds, 150yds, 170yds and 220yds. The signals are probabilities of the ball landing in the corresponding coarse distance category. The width of each category bin has been scaled to reflect the distance range it signifies. The x-axis is the distance to the hole, such that negative values indicate under-shooting, and positive distances of over-shooting the hole.

to judge the distance to the hole by using the feedback from any shot to update an estimate of the most similar previous task, using the distributions in Figure 4.3. This belief enables the robot to choose the club/clubs which would have been the best choice for the most similar previous task/tasks.

Consider, as an example, a hole 179 yards away. A coarse estimate of the distance can be incorporated as a prior over $\mathcal{T}$, otherwise an uniformed prior is used.

For a worked-out example, assume the robot is using greedy policy selection, and assume that it selects π_1 for the first shot due to a uniform prior, and that this resulted in an over-shot of 35 yards. The robot cannot gauge this error more accurately than that it falls into the category corresponding to over-shooting in the range of 20 to 50 yards. This signal will update the belief of the robot over the four types, and by Figure 4.3, the closest type to produce such a behaviour would be $\tau_{170} = 170$ yards. The new belief dictates that the best club to use for anything similar to τ_{170} is π_2. Using π_2, the hole is over-shot by 13 yards, corresponding to the category with the range 5 to 20 yards. With the same calculation, the most similar previous type is again τ_{170}, keeping the

best club as π_2, and leading the belief to converge. Indeed, given the ground truth in Table 4.1, this is the best choice for the 179 yard task. Table 4.2 describes this process over the course of 8 consecutive shots taken by the robot.

Shot	1	2	3	4	5	6	7	8
Club	1	2	2	2	2	2	2	2
Error	35.3657	13.1603	4.2821	6.7768	2.0744	11.0469	8.1516	2.4527
Category	20–50	5–20	-5–5	5–20	-5–5	5–20	5–20	-5–5
β entropy	1.3863	0.2237	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 4.2: The 179 yard example. For each of 8 consecutive shots: the choice of club, the true error in distance to the hole, the coarse category within which this error lies (the signal received by the agent), and the entropy of the belief. This shows convergence after the third shot, although the correct club was used from the second shot onwards. The oscillating error is a result of the variance in the club yardage. Although the task length was $K = 3$ strokes, we show these results for longer to illustrate convergence.

Figure 4.4 shows the performance of BPR with greedy policy selection in the golf club selection task averaged over 100 unknown golf course holes, with ranges randomly selected between 120 and 220 yards. This shows that on average, by the second shot, the robot will have selected a club capable of bringing the ball within 10–15 yards of the hole.

4.6.2 Online Personalisation

Consider a service offered telephonically, such as telephone banking for example, where the bank tries to improve the speed of answering telephonic queries by having a personalised model for understanding the speech of each user, and perhaps responding with a voice preferred by that user. In a traditional speech recognition system, the user may have time to train the system to her own voice, but this is not possible in this scenario. As a result, the phone service may have a number of different pretrained language models and responses, and over the course of many interactions with the same user, try to estimate the best such model to use.

Let a user i be defined by a preference in language model $\lambda_i \in \{1,\ldots,L\}$, where L is the number of such models. The policy π executed by the telephonic agent also

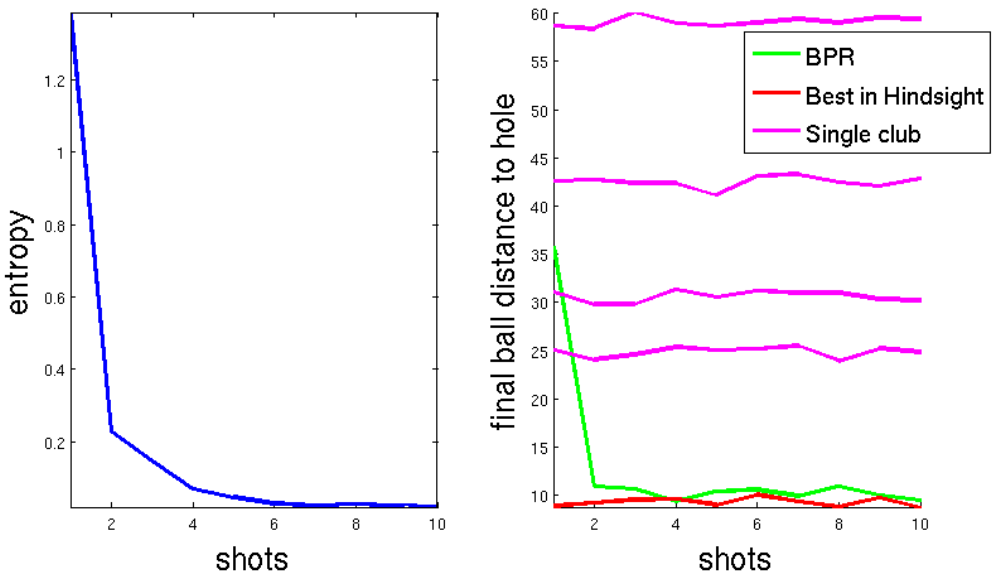

Figure 4.4: Performance of BPR on the golf club example, with results averaged over 100 unknown holes, showing the decrease in entropy of the belief β and average distance to the hole (where lower scores are better). Performance of the pure four clubs (being the average performance of each single club over all 100 holes), as well as the best club for each hole in retrospect, is shown for regret comparison. Although the task length was $K = 3$ strokes, we show these results for longer to illustrate convergence. Error bars have been omitted for clarity.

corresponds to a choice of language model, i.e. $\pi \in \{1,\ldots,L\}$. The goal of the agent is to identify the user preference λ_i, whilst minimising frustration to the user.

Assume that each telephonic interaction proceeds by means of the transition system through the six states given in Figure 4.5. In every state, there is only one action which can be taken by the system, being to use the chosen language model. At the beginning of the call, the user is in the *start* state. We assume the system can identify the user (perhaps by caller ID), and selects a language model. If, at any point, the system can deal with the user's request, the call ends successfully. If not, we assume the user becomes gradually more irritated with the system, passing through states *frustrated* and *annoyed*. If the user reaches state *angry* and still has an unresolved request, she is transferred to a human. This counts as an unsuccessful interaction. Alternatively, at any point the user may hang up the call, which also terminates the interaction unsuccessfully.

The transition dynamics of this problem depend on a parameter $\rho = \frac{L-|(\pi-\lambda_i)|}{L}$

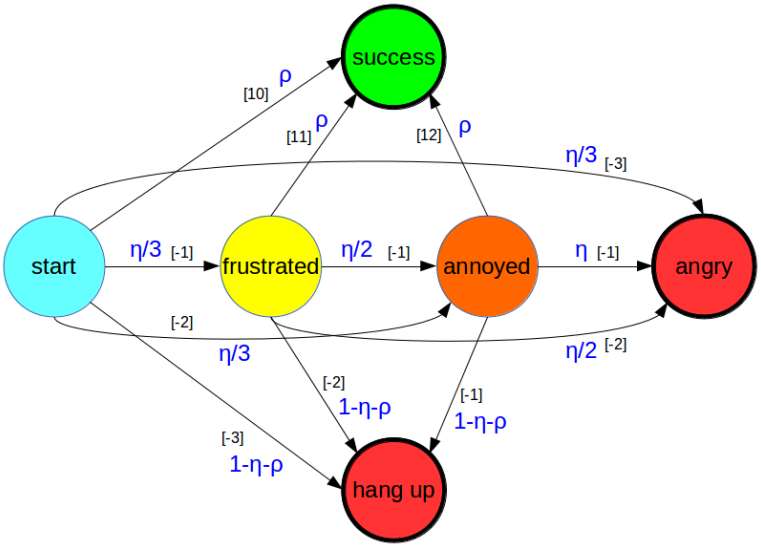

Figure 4.5: Transition system describing the online telephonic personalisation example. Circles are states, thick bordered circles are terminal states, small black edge labels in square brackets are the transition rewards, and large blue edge labels are transition probabilities. See text for a description of the parameters ρ and η.

which describes how well the selected language model π can be understood by a user of type λ_i. An additional parameter η governs the trade-off between the user becoming gradually more frustrated, and simply hanging up when the system doesn't respond as expected. In our experiments, $\eta = 0.3$, unless $\pi = \lambda_i$, in which case $\eta = 0$.

The aim of this experiment is to demonstrate the effect of using different observation signals to update beliefs, as described in Section 4.4.1. As a result, the transition dynamics and the rewards of this domain, shown in Figure 4.5, have been selected such that only two utility signals are possible: $U = 10$ for a successful completion of the task, and $U = 3$ otherwise. Similarly, any state that transitions to the unsuccessful outcome *angry* state, receives the same reward for a transition to the unsuccessful outcome *hang up* state. Finally, all transition probabilities between the states *start*, *frustrated*, *annoyed*, and *angry* are independent of ρ, and thus the type. This set up mirrors the fact that in general the state sequence given by the signal (s, a, s') is more informative than the reward sequence (s, a, r), which is in turn more informative than the utility signal U, being the total reward. However, in many applications U may be the only one of these signals available to the agent, for example gauging the frustration of the caller may not be possible.

Figure 4.6 shows comparative performance between BPR, using sampling of the

belief as a policy selection mechanism, with these three candidate signals. As expected, the lowest regret (and variance in regret) is achieved using the (s,a,s') signal, followed by the (s,a,r), and finally the U signal. We do note that all three signals eventually converge to zero regret.

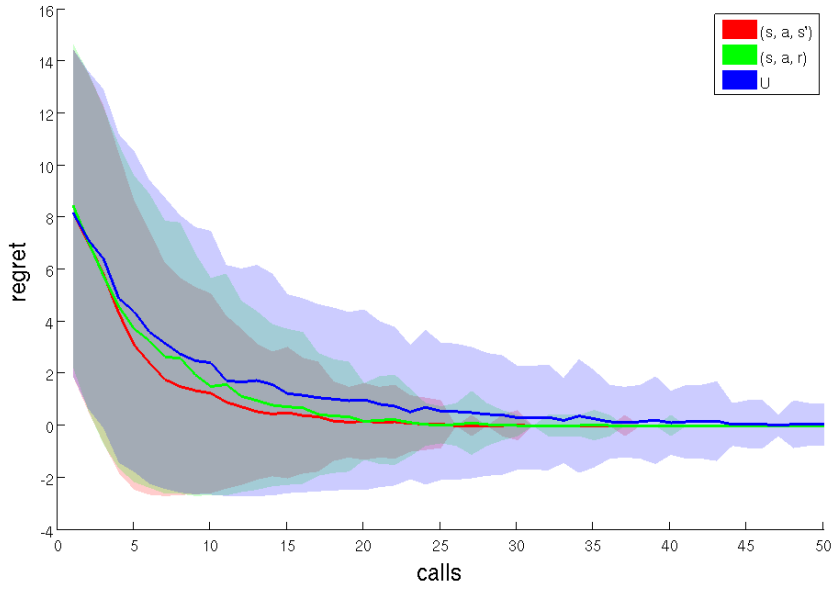

Figure 4.6: Regret, showing comparative performance of BPR on the telephone banking domain, using (s,a,s'), (s,a,r), and U as signals.

4.6.3 Pacman

The Pacman domain involves an autonomous agent (the Pacman) moving around a discrete two-dimensional maze with free space arranged in a figure-eight layout, and tasked with collecting items of food located in a subset of the free (non-obstacle) cells in the maze. There is one adversary (the ghost) which moves around according to some stationary policy. The task is episodic and ends in a win if the Pacman collects all food items, or a loss if the Pacman collides with the ghost. The Pacman agent can move up, down, left and right into neighbouring free cells. Figure 4.7 shows the domain.

The Pacman agent receives a reward of 10 for each food collected, 100 for winning, and -100 for losing. The state and action space are always the same, but the motion strategies used by the adversary change from episode to episode, reflecting a type τ that must be inferred. These include moving backwards and forwards across the top row, traversing the entire domain clockwise or anti-clockwise, or moving randomly.

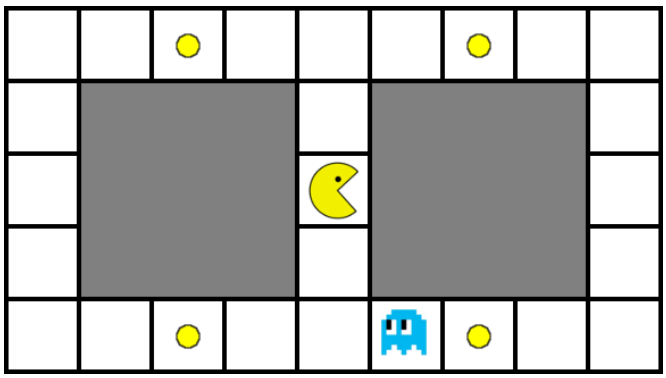

Figure 4.7: The Pacman domain, featuring the Pacman agent in the centre cell, an adversarial ghost in blue, and four food items.

The Pacman is only able to observe the position of the ghost if it is within line of sight; i.e., the same row or column with all intermediary spaces free. This leads to a partially observable process from the agent's point of view.

For each movement strategy used by the ghost adversary (type), the Pacman agent is equipped with an optimal policy trained offline using Q-learning. We use the total utility received by the agent in an episode as the observation signal, and model the corresponding performance functions as Gaussian distributions over the utility signals.

Figure 4.8 shows the performance of BPR using sampling of β as a policy selection mechanism compared to Probabilistic Policy Reuse with Q-learning (PRQ) (Fernandez and Veloso, 2006) and standard Q-learning.

These experiments were run without discounting, i.e. $\gamma = 1$, over $N = 10000$ episodes for a maximum of 200 time steps. The PRQ parameters were set as $\tau = 0$ and $\Delta\tau = 0.05$ as used by Fernandez and Veloso (2006). The algorithms used learning rate $\alpha = 0.5$ and exploration probability $\varepsilon = 0.5$, both of which were decreased by $0.5/N$ per episode.

These results demonstrate that BPR, being a bandit-style algorithm equipped with pre-trained policies, is able to converge orders of magnitude faster than learning from scratch (Q-learning), even when that learning is seeded by policy transfer (PRQ).

This experiment further demonstrates the ability of BPR to handle partially observable domains, as well as domains involving the execution of complex policies.

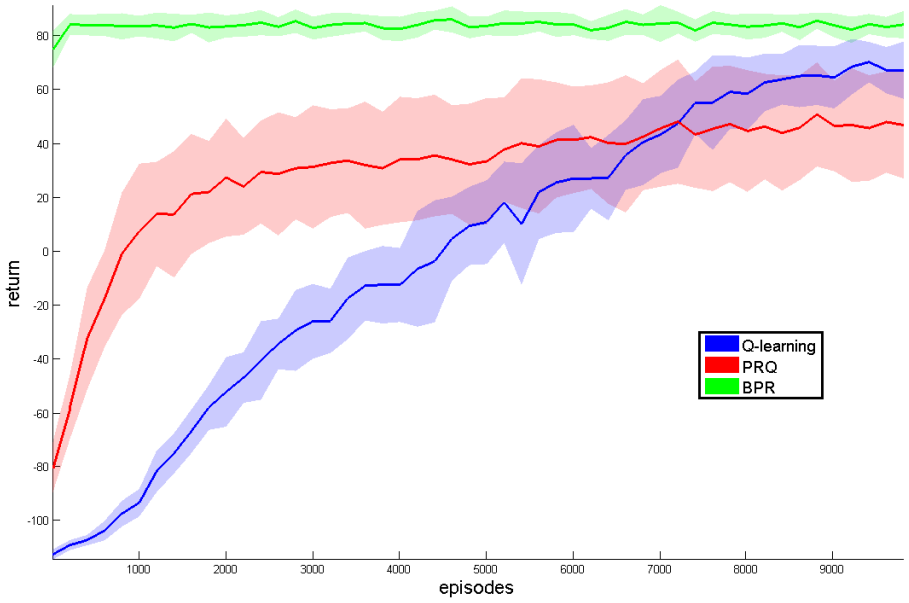

Figure 4.8: The comparative convergence of performance on different Pacman tasks between BPR, PRQ and Q-learning, averaged over twenty random tasks. The shaded region denotes one standard deviation.

4.6.4 Surveillance Domain

The surveillance domain models the monitoring problem laid out in the introduction. Assume a base station is tasked with monitoring a wildlife reserve spread out over some large geographical region. The reserve suffers from poaching and so the base station is required to detect and respond to poachers on the ground. The base station has a fixed location, and so it monitors the region by deploying a low-flying (in order to observe behaviour on the ground) light-weight autonomous drone to complete particular objectives, using different strategies. The episodic commands issued by the base station may be to deploy to a specific location, scan for unusual activity, and then report back. After each such episode the drone returns with some information, being an indicator of whether or not there was any suspicious activity in the designated region. The base station is required to use that information to better decide on the next strategy to be used by the deployed drone.

Concretely, we consider a 26×26 cell grid world, which represents the wildlife reserve, and the base station is situated at a fixed location in one corner. We assume that there are 68 target locations of interest, perhaps being areas with a particularly

high concentration of wildlife. These areas are arranged around four 'hills', the tops of which provide better vantage points. Figure 4.9 depicts this setting.

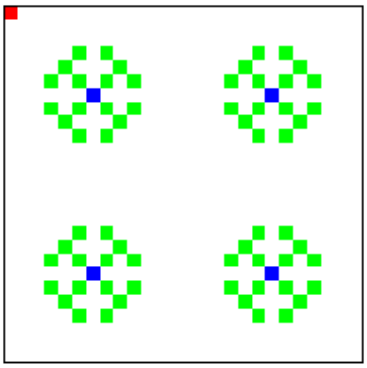

Figure 4.9: Example of the surveillance domain. The red cell is the base station, green cells correspond to surveillance locations, and blue cells are hill tops. The base station is tasked with deploying drones to find poachers which may be at one of the surveillance locations.

At each episode, the base station deploys the drone to one of the 68 locations. These 68 target locations each correspond to a different type, as we assume that each represents a large geographical area, and thus multiple locations in the reserve. For each type, we assume that the drone has a pre-learnt policy for reaching and surveying that area, which includes elements of local control, such as dealing with wind perturbations and avoiding trees.

The observation model is described by the drone being able to identify whether an intruder is at the location it visits, or somewhere nearby, in a diagonally adjacent cell. The only exceptions are the centres of each hill, which by corresponding to a high vantage point, provide a weak signal stating that the intruder is around the hill. Additionally this signal process is noisy. For a distance d between the region surveyed and the region occupied by the poachers, the reward R received by the agent is

$$R \longleftarrow \begin{cases} 200 - 30d + \psi & \text{if agent surveys a hilltop and } d \leq 15 \\ 200 - 20d + \psi & \text{if agent surveys another location and } d \leq 3 \\ \psi & \text{otherwise,} \end{cases}$$

where $\psi \sim \mathcal{N}(10, 20)$ is Gaussian noise.

Figure 4.10 presents a comparison between six variants of the BPR algorithm. Four are the exploration heuristics proposed in Section 4.5, being BPR-KG, BPR-BE, BPR-

PI, BPR-EI. Additionally, we use two further heuristics, being sampling the belief β, and ε-greedy selection, which chooses the best policy according to β with probability $1-\varepsilon$, and selects a policy uniformly at random with probability ε, where $\varepsilon = 0.3$. These six variants were run on the domain in Figure 4.9, and averaged over 10 random tasks, with standard deviations of the regret shown in Table 4.3.

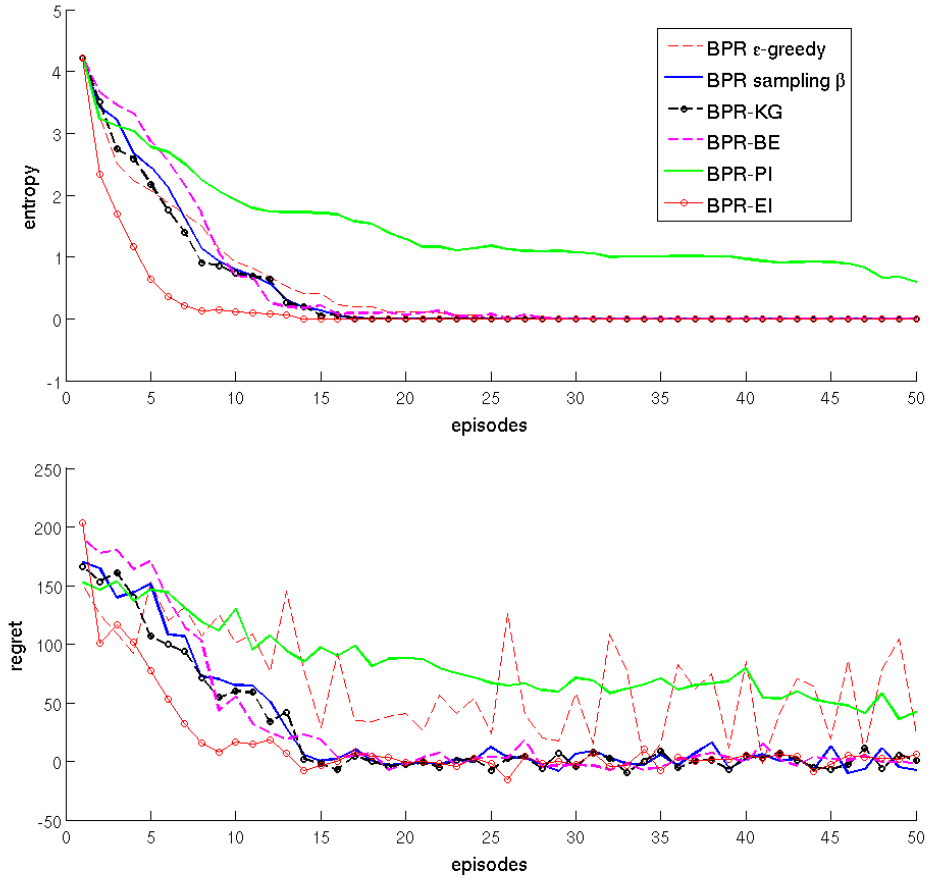

Figure 4.10: Comparison of the six policy selection heuristics on the 68 task surveillance domain, averaged over 10 random tasks. (a) The entropy of the belief $\beta(\tau)$ after each episode. (b) The regret after each episode. Error bars are omitted for clarity, but standard deviations of the regret are shown in Table 4.3.

Note in Figure 4.10 that BPR-BE, BPR-KG, and BPR with sampling the belief β all converge in about 15 episodes, which is approximately a quarter the number that would be required by a brute force strategy which involved testing every policy in turn. Both BPR-PI and BPR with ε-greedy selection fail to converge within the allotted 50 episodes. BPR-EI shows the most rapid convergence.

episode	ε-greedy	sampling	BPR-KG	BPR-BE	BPR-PI	BPR-EI
5	72.193	103.07	76.577	96.529	15.695	27.801
10	97.999	86.469	75.268	91.288	62.4	33.112
20	83.21	18.834	7.1152	17.74	72.173	16.011
50	86.172	10.897	18.489	13.813	101.69	11.142

Table 4.3: Standard deviations of the regret for the six BPR variants shown in Figure 4.10, after episodes 5, 10, 20 and 50.

We now compare the performance of BPR to different approaches for solving the same problem, namely multi-armed bandits for which we use UCB1 (Auer et al., 2002a), and Bayesian optimisation represented by GP-UCB (Srinivas et al., 2009). We note upfront that although these are the most similar to our own in terms of the problems they solve, the assumptions they place on the problem space are different, and thus so is the information they use. These results are presented in Figure 4.11, showing comparative performance between these approaches on the surveillance domain, averaged over 50 tasks. We use BPR-EI in this experiment, as it was the best performing BPR variant as seen in Figure 4.10.

We provide the UCB bandit algorithm with prior information, in the form of the expected performance of each policy given the task distribution $G_0(\tau)$, assumed to be uniform in this case. This prevents UCB from having to test each arm first on the new task, which requires 68 episodes in this case before it can begin the decision process. It is still slower to converge than BPR, as information from each episode only allows UCB to update the performance estimate of a single policy, whereas BPR can make global updates over the policy space.

This same problem would be encountered by other methods which involve estimating the performance of each policy (arm in the original literature) individually. A notable example is *interval estimation* (Kaelbling, 1990) which, for each policy, estimates a statistical upper bound on the value of that policy from previous executions of the policy based on the assumption that these values are, for example, normally distributed (Vermorel and Mohri, 2005). As with the UCB method, an approach such as this is inherently slower, as each policy can only update its own statistics, and not those of other policies. Our method is able to achieve this through the use of the $\mathcal{F}_\pi^\tau$ observation model, which allows us to update beliefs over all types simultaneously, in turn simultaneously updating estimates of the performance of every policy through the

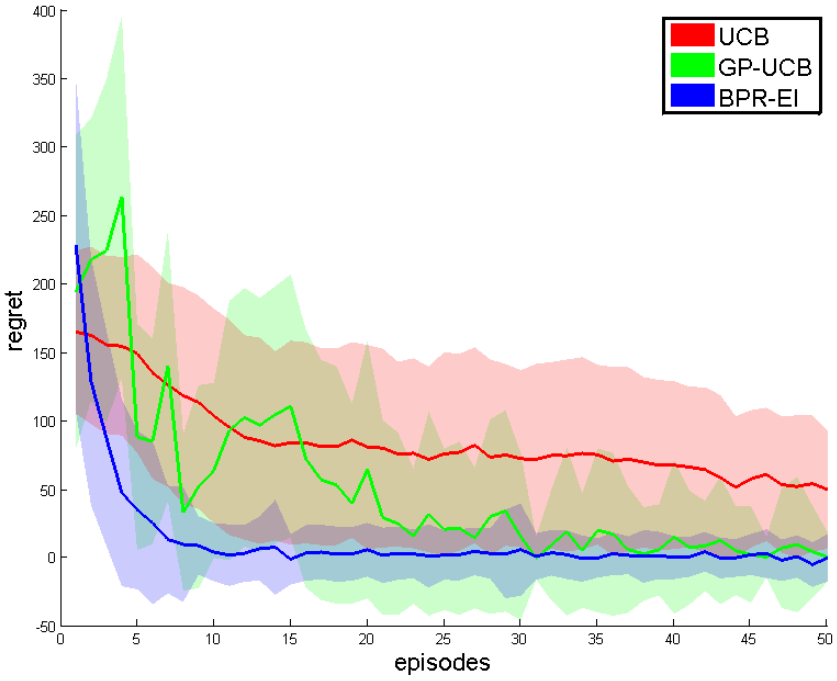

Figure 4.11: Comparison of the episodic regret of BPR (represented by BPR-EI), a bandit approach (UCB1), and a Bayesian optimisation approach (GP-UCB) on the 68 task surveillance domain, averaged over 50 random tasks. Shaded regions represent one standard deviation.

performance model $\mathrm{P}(U|\mathcal{T},\Pi)$. Even incorporating accurate prior probabilities and estimates of value does not help in this case, as each task is equi-probable.

On the other hand, an optimisation approach such as GP-UCB is better suited to this problem, as it operates with the same restriction as BPR of maintaining low sample complexity. However, unlike BPR, Bayesian optimisation requires a metric in policy space. This information is not known in this problem, but is approximated from performance in the training tasks. As a result of this approximation, sampling a single point in GP-UCB (corresponding to executing a policy) again only provides information about a local neighbourhood in policy space, whereas the same action allows BPR to update beliefs over the entire task space.

Further discussion of the differences between BPR and both bandits and optimisation approaches is provided in Sections 4.7.2 and 4.7.3 respectively.

Figure 4.12 shows the trade-off between library size and sample complexity with respect to the regret. For each setting of library size and sample complexity pairs, we

average over 200 trials. For each trial, a random subset of the full task set is used as the policy library, and the task is drawn from the full task set, meaning that this includes both seen and unseen tasks in the online phase. As can be seen, performance can be improved by increasing either the library size, or the time allocated (in terms of number of episodes) to complete the new task.

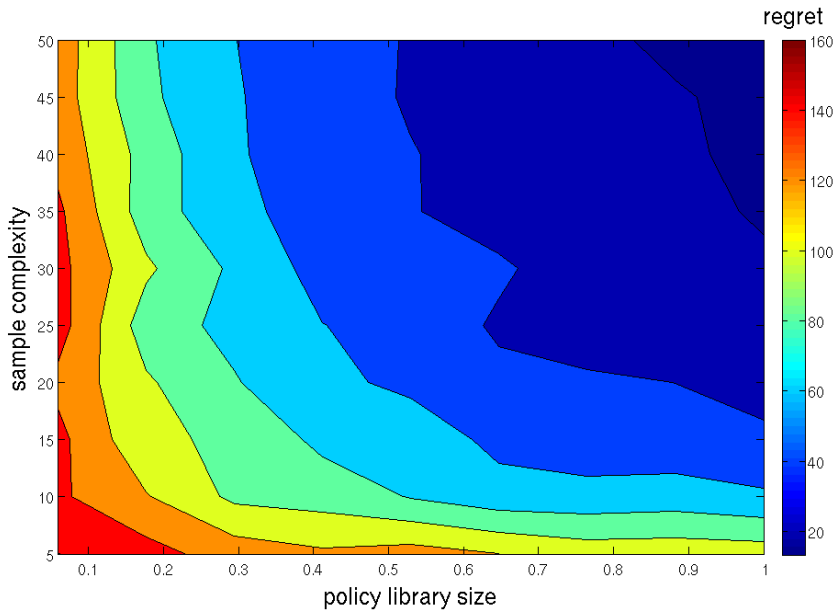

Figure 4.12: Average episodic regret for running BPR-EI on the 68 task surveillance domain, with different library sizes (as a proportion of the full task space size) and number of episodes (sample complexity), averaged over 200 random tasks.

4.7 Discussion and Related Work

4.7.1 Transfer Learning

The optimal selection from a set of provided policies for a new task is in essence a transfer learning problem (see the detailed review by Taylor and Stone (2009)). Specifically, Bayesian policy reuse aims to select the policy in a library Π which is the best policy for the most similar seen type to transfer into a new, initially unknown, task instance. One transfer approach that considers the similarity between source and target tasks is by Lazaric (2008), where generated (s, a, r, s') samples from the target task are used to estimate similarity to source tasks. Similarity is measured by the average

probability of the generated transitions happening under the source task. Then, samples from the more similar source tasks are used to seed the learning of the target task, while less similar tasks are avoided, escaping negative transfer. Bayesian policy reuse does not assume learning is feasible in the first place, so that it relies on transferring a useful policy immediately. Also, we use a Bayesian similarity measure which allows exploiting prior knowledge of the task space.

4.7.2 Correlated Bandits

Using a once-off signal per episode relates BPR to an instance of correlated bandits. In this problem, the decision-making agent is required to pull an arm every decision time slot from a fixed set of arms and observe its return, which will then be used to update the estimates of the values of not only the arm that was pulled, but instead of a subset of all the arms. In the case of BPR, the arms correspond to policies, and the new task instance is the bandit 'machine' that generates utilities per pull of an arm (execution of a policy).

In the correlated bandits literature, the form of correlation between the arms is known to the agent. Usually, this happens to be the functional form of the reward curve. The agent's task is then to identify the parameters of that curve, so that the hypothesis of the best arm moves in the parameter space of the reward curve. In response surface bandits (Ginebra and Clayton, 1995), the rewards of the arms are correlated through a functional form with unknown parameters, with a prior over these parameters, and with a known metric on the policy space. In our work, we do not assume any functional form for the response surface, and we assume that the metric on the policy space is unknown. More recently, Mersereau et al. (2009) present a greedy policy which takes advantage of the correlation between the arms in their reward functions, assuming a linear form with one parameter, with a known prior.

In our framework, we do not specify any functional form for the response surface, but only specify assumptions on the continuity and smoothness of the surface. Instead, we treat the known types as a set of learnt bandit machines with known behaviour for each of the different arms. These behaviours define local 'kernels' on the response surface, which we then approximate by a sparse kernel machine, and then track a hypothesis of the best arm using that space. This is similar to the Gaussian process framework, but in our case, the lack of a metric on the policy space prevents the definition of the covariance functions needed there. This point is elaborated in Section

4.7.3.

In another thread, dependent bandits (Pandey et al., 2007) assume that the arms in a multi-armed bandit can be clustered into different groups, the members of each have correlated reward distribution parameters. Then, each cluster is represented with one representative arm, and the algorithm proceeds in two steps: first, a cluster is chosen by a variant of UCB1 (Auer et al., 2002a) applied to the set of representative arms, then the same method is used again to choose between the arms of the chosen cluster. We assume in our work that the set of seen tasks span and represent the space well, but we do not dwell on how this set of tasks can be selected. Clustering is one good candidate for that, and one particular example of identifying the important types in a task space can be seen in the work of Mahmud et al. (2013).

4.7.3 Relation to Bayesian Optimisation

If the problem of Bayesian policy reuse is treated as an instance of Bayesian optimisation, we consider the objective

$$\pi^* = \arg\max_{\pi \in \Pi} \mathrm{E}[U|x^*, \pi],$$

where $x^* \in \mathcal{X}$ is the unknown process with which the agent is interacting, and $\mathrm{E}[U|x^*, \pi]$ is the unknown expected performance when playing π on x^*. This optimisation involves a selection from a discrete set of alternative policies ($\pi \in \Pi$), corresponding to sampling the performance function at a discrete set of locations. Sampling this function is expensive, corresponding to executing a policy for an episode, and as a result the performance function must be optimised in as few samples as possible.

However, a Bayesian optimisation solution requires the optimised function to be modelled as a Gaussian Process (GP). There are two issues here:

1. Observations in BPR need not be the performance itself (see Section 4.4), while the GP model is appropriate only where these two are the same.

2. BPR does not assume knowledge of the metric in policy space. This is however required for Bayesian optimisation, so as to define a kernel function for the Gaussian process. An exception is in the case where policies all belong to a parametrised family of behaviours, placing the metric in parameter space as a proxy for policy space.

Still, we assume smoothness and continuity of the response surface for similar tasks and policies, which is a standard assumption in Gaussian process regression. Bayesian

policy reuse uses a belief that tracks the most similar seen type, and then reuses the best policy for that type. This belief can be understood as the mixing coefficient in a mixture model that represents the response surface.

To see this, consider Figure 4.13 which shows an example 2D response surface. Each type is represented by a 'band' on that surface; a set of curves only precisely known (in terms of means and variances) at the set of known policies. Projecting these 'bands' into performance space gives a probabilistic description of the performance of the different policies on that type. Each of these projections would be a component of the mixture model, and would represent the type's contribution to the surface.

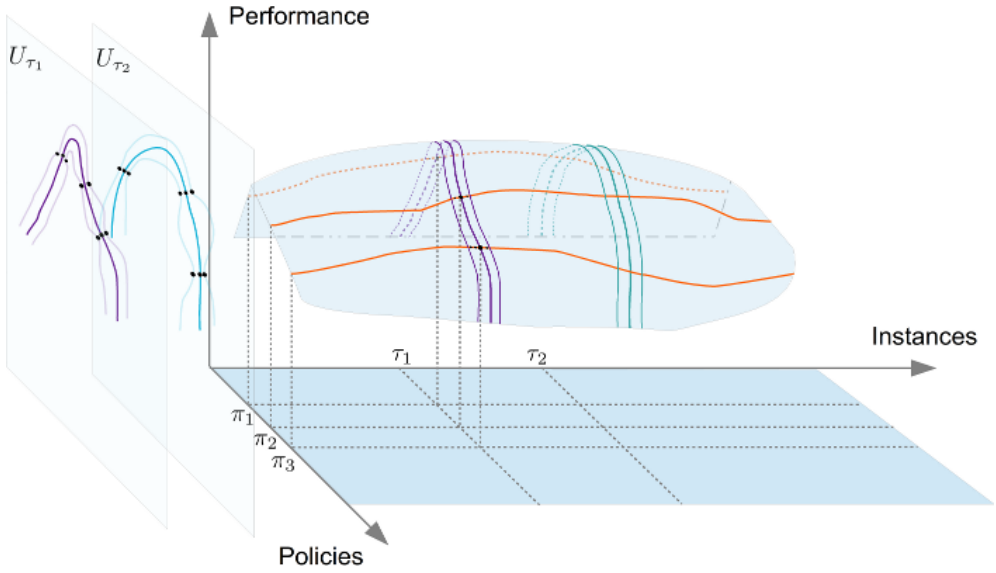

Figure 4.13: An example 2D response surface. The 'bands' on the curve show two types, and the lines that run through the curve from left to right are policy performances for all types. The agent only has access to the intersection of types' bands with policy curves (the black dots). Shown on the left are the performance curves of the two types τ_1 and τ_2 under all policies. These are represented as Gaussian processes in the Policies-Performance plane.

Any new task instance corresponds to an *unknown* curve on the surface. Given that the only knowledge possessed by the agent from the surface is these type probabilistic models, Bayesian policy reuse implicitly assumes that they act as a basis that span the space of possible curves, so that the performance under any new task can be represented as a weighted average of the responses of the seen types[3]. To this extent,

[3]Note that this will create a bias in the agent's estimated model of the type space toward the types that have been seen more often previously. We assume that the environment is benign and that the offline phase is long enough to experience the necessary types.

the performance for the new task instance is approximately identified by a vector of weights, which in our treatment of BPR we refer to as the type belief. As a result, the BPR algorithm is one that fits a probabilistic model to a curve through sampling and adjusting weights in an approximated mixture of Gaussian processes.

4.7.4 Relation to Bayesian Reinforcement Learning

Bayesian Reinforcement Learning (BRL) is a paradigm of Reinforcement Learning that handles the uncertainty in an unknown MDP in a Bayesian manner by maintaining a probability distribution over the space of possible MDPs, and updating that distribution using the observations generated from the MDP as the interaction continues (Dearden et al., 1999). In work by Wilson et al. (2007), the problem of Multi-task Reinforcement Learning of a possibly-infinite stream of MDPs is handled in a Bayesian framework. The authors model the MDP generative process using a hierarchical infinite mixture model, in which any MDP is assumed to be generated from one of a set of initially-unknown classes, along with a prior over the classes.

Bayesian Policy Reuse can be regarded as an special instance of Bayesian Multi-task Reinforcement Learning with the following construction. Assume an MDP that has a chain of K states and a collection of actions that all connect each state in the chain to the next. The set of actions is given by Π, the policy library. The MDPs are parametrised by their type label τ. For each time step in an MDP, the agent takes an action (a policy $\pi \in \Pi$) and the MDP returns with a performance signal, U_τ^π. The task of the agent is to infer the best policy for this MDP (a sequence $\pi^0, \ldots, \pi^t$) that achieves the fastest convergence of values U.

4.7.5 Other Bayesian Approaches

Engel and Ghavamzadeh (2007) introduce a Bayesian treatment to the Policy Gradient method in reinforcement learning. The gradient of some parametrised policy space is modelled as a Gaussian process, and paths sampled from the MDP (completed episodes) are used to compute the posteriors and to optimise the policy by moving in the direction of the performance gradient. The use of Gaussian processes in policy space is similar to the interpretation of our approach, but they use them to model the gradient rather than the performance itself.

When no gradient information is available to guide the search, Wingate et al. (2011) propose to use MCMC to search in the space of policies which is endowed with a

prior. These authors discuss various kinds of hierarchical priors that can be used to bias the search. In our work, we choose the policies using exploration heuristics based on offline-acquired performance profiles rather than using kernels and policy priors. Furthermore, we have a small set of policies to search through in order to optimise the time of convergence.

4.7.6 Storage Complexity

As described in Section 4.4, the use of different signals entail different observation models and hence different storage complexities. Assume that $|S|$ is the size of the state space, $|A|$ is the size of the action space, $|\Pi|$ is the size of the policy library, N is the number of seen types, $|R|$ is the size of the reward space, T is the duration of an episode, and B is the number of bits needed to store one probability value. For the performance signal, the storage complexity of the observation model is upper bounded by $\mathcal{SC}_{perf} = |\Pi| N |R| B$ for the average reward case, and $\mathcal{SC}_{perf,\gamma} = |\Pi| N \frac{1-\gamma^T}{1-\gamma} |R| B$ for the discounted reward case. For the state-action-state signals, the term is $\mathcal{SC}_{s'} = |\Pi| N |R| |S| |A| B$, and for the immediate reward signal we get $\mathcal{SC}_r = |\Pi| N |S|^2 |A| B$. In applications that have $|R| > |S|$ we get the ordering $\mathcal{SC}_{perf} < \mathcal{SC}_r < \mathcal{SC}_{s'}$.

4.8 Online Adaptation of Interaction Environments to Diverse Users

4.8.1 Introduction

Modelling different classes of tasks as types with related optimal policies and related policy performance has been shown in this chapter to be an effective representation for reasoning about policy reuse. The notion of types thus lends itself naturally to describing different classes of people, such as the users of some interactive system, which are likely to differ in skill levels and preferences. To this end, we examine the particular application of adapting an interface itself to best suit different users, through the type and policy reuse formalisms introduced above.

Interactive interfaces, such as natural gesture-based user interfaces, have revolutionised the way we interact with robots, video games and many related applications. A key attribute of such an interface is the way in which users' high-dimensional movements are interpreted and correspondingly mapped to the command set within the

application. Many modern software applications have similar requirements. For instance, interfaces to search engines (e.g., Google, Bing), or photo-editing software (e.g., Adobe Photoshop), involve numerous configuration choices that, implicitly or explicitly, define the context of the interaction between the human user and the computational process actually being performed.

Almost all of these applications must be deployed with large user populations, with substantial diversity in the performance that results from any given pair of user and configuration setting. For instance, even with a simple interface such as joystick-based navigation within a video game or field robotics application, there is wide interface-dependent variety in dexterity and skill of users (Rosman et al., 2014). A mismatch between a user's skill level and settings such as the sensitivity level at which one interprets a particular motion as a signal can have substantial negative impact on the user's ability to perform a task. Sometimes the mismatches can be in assumptions about perceptual ability, e.g., how precisely aware the user is regarding the current computational context. As studies have shown, perceptual limitations can have a disproportionately large effect on a user's performance when the task involves interactive decisions in a dynamic environment (Valtazanos and Ramamoorthy, 2013). Similar effects are observed in software applications, such as in the performance difference between young children and skilled adults when presented with a number of choices by an interface.

These issues of personalisation in diverse applications are unified in the computational problem of automatically shaping the environment so that it is best adapted to the specific user whom the system is currently facing. Our particular interest is in adapting online, to personalise on the fly as the user interacts with the system, without long calibration phases and with only partial knowledge of the underlying traits that induce variability. As such, this domain is well suited to BPR.

The objective of this section is to present such a model of personalisation and a version of the BPR algorithm for optimally adapting the corresponding *action set* using which the user must solve the task at hand. The effectiveness of such a model and algorithm can be characterised by a few key properties – *safety*, i.e., ensuring that individualised adaptation does not lead to worse performance than a pre-configured statically optimal setting, *efficiency*, i.e., learning to obtain settings that enable the user to achieve their maximal performance, and *speed*, i.e., the ability to adapt within a few episodes (a low sample complexity) which roughly corresponds to users' patience. With this in mind, we present an empirical evaluation with real and simulated human

participants to further validate these claims.

4.8.2 Related Work

The problem of learning to adapt to user behaviour is garnering increasing attention from researchers in different domains. In one formulation, the learning agent may be assisting a user whose goal is unknown. For instance, Fern and Tadepalli (2010b) present the hidden-goal MDP to model problems such as that faced by a doorman who must anticipate when and where to go in order to help someone. In general, this is a computationally hard problem and fairly strong assumptions are required to achieve tractable results, such as that the only unknown is the goal state of the user. Indeed, this problem has a longer history in the literature on plan recognition (Charniak and Goldman, 1991).

Our focus is different in that our problem is not that of trying to predict the user's future state alone, based on the past (as in the doorman example above), but instead we try to infer latent 'personality traits' or skills/preferences of the user in order to shape their environment – to enable them to do better in their chosen task. In this sense, our work is related to recent work on personalisation (Seuken et al., 2012; Zhang et al., 2009).

In work by Hoey et al. (2010), a closely related problem is solved using partially observable MDPs (POMDPs). The problem is to synthesise a policy for issuing prompts to dementia patients that is best adapted to an unobserved 'attitude' that must be inferred. A key attribute of this model is that there are two actors – the patient and the prompting assistive device, so that the adaptation is to change what the prompting device does in order to get the patient to do what is best. This is also related to incentive decision processes (Reddi and Brunskill, 2012), environment shaping (Zhang et al., 2009), website morphing (Hauser et al., 2009), etc. In other applications, such as robotics, similar ideas have been explored in the form of mixed-observability MDPs (MOMDPs) (Ong et al., 2010), applied to intention-aware motion planning (Bandyopadhyay et al., 2012; Sisbot et al., 2007).

In related work in recommender systems, it is common to model user behaviour as a Markov Chain, based on which one adapts recommendations (Jannach et al., 2011). Sometimes, this problem is given a sequential decision making flavour (Chickering and Paek, 2007; Shani et al., 2005) such as by using a bandit learning algorithm to pick optimal recommendation strategies over time. In our work, we need more expressive

models of the user herself, whom we model as a parametrised-MDP, based on which we invoke a version of BPR as a sequential decision making algorithm.

The notion of designing a customised user interface has been studied in the HCI community, e.g. Gajos et al. (2008), where one finds ways to automatically synthesise user interfaces tuned to user abilities. The process involves offline calibration using a battery of tasks, based on which one obtains information necessary to pose the interface design problem as one of combinatorial search. While we take inspiration from such work, the focus of our own work is different as we are motivated by quick online adaptation to different users. More so than the combinatorial search, the harder problem in our setting is that of model selection to identify types of users whose relevant traits are latent and unobserved.

4.8.3 A Model of User Interaction

4.8.3.1 Interaction Shaping As Action Set Selection

In our model (Mahmud et al., 2014), the user[4] of the system acts according to a Markov decision process (MDP) (Puterman, 1994) and invokes a correspondingly optimal policy. The user is defined by her skill level (her type $\tau \in \mathcal{T}$), which determines the transition function of the MDP – this models the fact that, for instance, a less skilled user will have a different way of transitioning between states (e.g., being more variable) to that of an expert. Unlike in standard applications of optimising policies for given MDPs, we will be changing the action set α as the interaction proceeds in episodes, which is the main point of the interaction shaping process. Changing the action set corresponds to invoking a policy in policy reuse. The goal of the learner[5] is to choose action sets so as to maximise the future expected discounted reward of the user. So, the search space for our learning algorithm is again the type space $\mathcal{T}$, rather than the space of paths the user actually traverses, and the value of a type is the value of the optimal policy given the type.

Note that for the remainder of the discussion of this application domain, the choice of *action set* provided to the user by the learning agent corresponds to a policy in the policy reuse sense. We then refer to *policy* as the behaviour invoked by the user in her current task.

[4] In other applications the user may be represented by another behaviourally motivated model of choice, without fundamentally altering anything else in our framework.

[5] We refer to the learning algorithm which chooses the action sets as the *learner*.

In particular, we are concerned with a class of MDPs $\mathcal{M}$ parametrised by the type parameter τ:

$$\mathcal{M} = \{((S, A, T(\cdot;\tau), R, \gamma) | \tau \in \mathcal{T}\}. \tag{4.7}$$

These MDPs differ only in the transition function $T(s'|s,a;\tau)$, which depends on τ (the skill level of the user). $\mathcal{T}$ is the set of all types, τ. For instance, the user controlling a joystick corresponds to a MDP. Furthermore, a poorly skilled user will have poor control over the joystick which means that she will only be able to make coarse moves (regardless of the sensitivity of the joystick). Whereas, a highly skilled user will be able to execute sharp curves if given an appropriately sensitive joystick. This implies that the MDP corresponding to the task for each user type has different transition functions (for example, transitions with high and low entropy for low and high skilled users respectively).

We further assume that the action space A is large, but is partitioned, where AS is the set of all the cells of the partition. That is, if action sets $\alpha, \alpha' \in \mathsf{AS}$, then α and α' are disjoint; and $\bigcup_{\alpha \in \mathsf{AS}} \alpha = A$. We only allow user policies that take values in a single $\alpha \in \mathsf{AS}$ (we say the user policy is restricted to α). In the joystick example, different action sets $\alpha \in \mathsf{AS}$ correspond to different sensitivity levels of the joystick. The actions for a particular α/joystick sensitivity are the motion actions at that particular level of granularity. Of course, whether the user will be able to execute them depends on her skill level (i.e. the transition function ultimately also depends on the user type). The value function of a user policy π now depends on τ and is denoted by V_τ^π. For a given α, we denote $V_\tau^\alpha = \max_\pi V_\tau^\pi$ and denote by $\pi_\alpha^* = \arg\max_\pi V_\tau^\pi$, where the max is defined over π restricted to α.

Our learning problem can now be defined as follows. The user solves a sequence of MDPs from the set $\mathcal{M}$ in collaboration with the learner – that is the goal of both the user and the learner is to maximise the future expected discounted reward of the user. At the beginning of each phase of the problem, nature chooses the type τ^* (corresponding to a new user) according to a distribution $G_0(\tau)$ and in response, the learner is required to choose the action set α. The user can now only use policies restricted to α. The learner cannot observe the true τ^* but knows the value of T for each τ, that is, the observation signal models are given by the transition functions T. The user knows her own type and also knows the value of T and R. Once α is given, the user acts according to her optimal policy for that action set, π_α^*.

The role of the learner is as follows. At any point in time, the learner can step in and change the action set from α to a new action set α'. In response, the user

starts acting according to $\pi^*_{\alpha'}$ instead (we assume that the user can adapt to this change immediately). The goal of the learner is to choose the action set $\arg\max_\alpha V^\alpha_\tau$. In terms of the joystick example, this means that at the beginning some user of unknown skill (τ^*) begins to use the interface. She operates the interface to the best of her ability given the sensitivity. Given her actions, the learner agent tries to estimate the skill level and based on that either increases or decreases the sensitivity.

Given the above setup, the *a priori* optimal action set is defined to be

$$\alpha_* = \arg\max_\alpha \mathbb{E}_{G_0(\tau)}[V^\alpha_\tau]. \tag{4.8}$$

In the next section, we derive an *a posteriori optimal* action set selection policy, that is, a policy for the learner agent that adaptively and optimally chooses the action set that is optimal according to some well-defined and reasonable criteria, and also eventually converges.

4.8.4 Determining Agent Type

In the following we present our approach to adaptively determine the user type through repeated interactions. We first describe our criterion, Bayes optimality, for choosing action sets and then describe the algorithm that uses the Bayes optimal action set.

4.8.4.1 Bayes Optimal Action Set

We assume the learner has full knowledge of the set of user types $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, through having learnt the transition and reward signal models. In the experiments, we acquire these types during an initial training phase by interacting with users. This is discussed further in Section 4.8.5. Assume that at time step t of a particular interaction session we have observed a state-action sequence $sa_{0:t} = s_0 a_0 s_1 a_1 \cdots s_t$ (with $sa_{0:0} = s_0$). Given this session, the probability of a type τ_i can be computed as $P(\tau_i | sa_{0:t}) = \prod_{i=0}^{t-1} T(s_{i+1}|s_i, a_i; \tau_i)\beta^0(\tau_i)$, where $\beta^0(\cdot)$ is the prior distribution over $\mathcal{T}$. Note that this probability $P(\tau_i|sa_{0:t})$ is exactly the belief of τ_i at time t: $\beta^t(\tau_i)$.

Having defined the posterior distribution over types as the belief, at step t the learner agent is required to choose the Bayes optimal action set α_{BO} i.e. the action set maximising expectation of V^α_τ with respect to the posterior over $\mathcal{T}$. More formally,

$$\alpha_{BO}(sa_{0:t}) = \arg\max_\alpha \sum_i V^\alpha_{\tau_i} \beta^t(\tau_i). \tag{4.9}$$

For the empty sequence $\emptyset$, we define $\alpha_{BO}(\emptyset) = \arg\max_\alpha \sum_i V^\alpha_{\tau_i} \beta^0(\tau_i)$.

4.8.4.2 Learning Algorithm

Our learning algorithm, Bayesian Environment Adaptation with Types (BEAT) is listed as Algorithm 9. At each step, BEAT observes the user action and updates the belief of each type. Additionally, it chooses a new action set. With probability $1-\varepsilon_t$ it chooses the Bayes optimal action set and with probability ε_t it chooses an action set at random. The exploration terms satisfy the condition of $\sum_n \varepsilon_n = \infty$ and $\lim_{n\to\infty} \varepsilon_n = 0$ (for instance $\varepsilon_n = 1/n$).

Algorithm 9 The Bayesian Environment Adaptation with Types (BEAT) Algorithm

Require: The set of observation models $T(\cdot;\tau)$, task termination condition ω, set of types $\mathcal{T}$, prior β^0 over types.

Initialize: ε_n is a sequence with $\sum_n \varepsilon_n = \infty$ and $\lim_{n\to\infty} \varepsilon_n = 0$.
Compute $V_{\tau_i}^{\alpha}$ for each $\alpha \in \mathsf{AS}$ and type τ_i.
Let s_0 be the initial state and $t \leftarrow 0$.
while $\omega = false$ **do**
  Observe user action a_t, reward r_t and next state s_{t+1}.
  Update belief of each type: $\beta^{t+1}(\tau_i) = T(s_{t+1}|s_t, a_t; \tau_i) \times \beta^t(\tau_i)$.
  $$\alpha_{t+1} \longleftarrow \begin{cases} \alpha_{BO}(sa_{0:t+1}) \text{ (defined in (4.9))} & \text{w.p. } 1-\varepsilon_t \\ \text{a random action set} & \text{w.p. } \varepsilon_t \end{cases}$$
  $t \longleftarrow t+1$.
end while

Note that the BEAT algorithm is a special case of BPR, corresponding to the use of ε-greedy policy selection, MDP transition functions as observation models, and action sets as pre-learnt policies to reuse.

4.8.5 Experiments

In this section we present two sets of experiments in a video game domain to illustrate our method. In the first set, the user is a robot/artificial user, while in the second set the users are humans. In the following, we describe the comparison algorithms and then discuss the results in detail. In our experiments, we compare BEAT with α_* (the *a priori*/population optimal action set), and the EXP-3 algorithm (Auer et al., 2002b; Cesa-Bianchi and Lugosi, 2006) for non-stochastic multi-armed bandits (NSMB). This latter framework is a natural fit for addressing the problem of action set selection. In the NSMB problem, there are *c arms* where each arm *i* has a *pay-*

off process $x_i(t)$ associated with it. The learner runs for T steps and at each step t needs to *pull*/select one of the arms $f(t)$, giving her a payoff of $x_{f(t)}(t)$. Additionally, the learner only gets to view the payoff of the arm $f(t)$ it has chosen. The goal of the learner is to minimise its *regret* with respect to the best arm, that is minimise the quantity $\max_i \sum_{t=1}^{T} x_i(t) - \sum_{t=1}^{T} x_{f(t)}(t)$. In general it is not possible to minimise this in any meaningful sense. An optimal algorithm in the general case, for minimising the *expected regret* was developed by Auer et al. (2002b), called the EXP-3 algorithm. In our experiment, each arm corresponds to an action set run for k steps and the payoff is the total discounted reward obtained in those k steps.

4.8.5.1 Experiment Setup

Design. Figure 4.14 shows the domain used in our experiments.

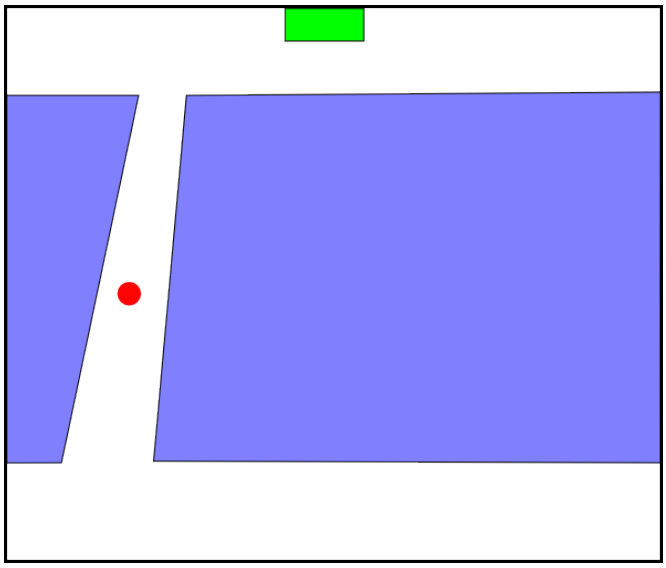

Figure 4.14: Our video game domain. The blue polygons are obstacles, the white regions are free space, the green rectangle is the goal location, and the red circle is the ball which is controlled by the user.

The goal of the user is to move a ball (red circle) from a fixed start location to the goal location (green rectangle). The ball moves with a constant speed z and at each step the user can choose to change the direction of motion into one of N, S, E, and W. Her loss (negative reward), received when she reaches the goal, is the sum of the normalised time taken and the normalised number of collisions. During play, the learner (system) chooses the speed to help minimise the loss. We capture two types of

limitations in user behaviour. The size of the ball reflects limits to the user's perceptual ability, with large ball sizes indicating reduced acuity. The noise of the motion of the ball reflects limits to motor ability, such as intrinsic jitter in the way she uses the device. Combinations of larger ball and noise imply less skill. These limitations are designed to be exaggerated in order to override natural variation in the user base.

Parameters. The state space is the location of the ball at each time step (the game field was 1000×1000). Each action is a move by z-units in a cardinal direction (N,S,E,W), where $z \in \{30,40,50,60,70\}$. Each action set α_z corresponds to all the motions at speed z. The direction at step t is the most recent direction chosen by the user at $t' \leq t$ (so, the actual play of the user is modelled in the MDP as the user choosing the same action at every step as her last choice of direction). There was a constant amount of noise (on top of noise due to type) with each action, so the ball moves in the given direction followed by (discretised) Gaussian motion (perpendicular to the direction of motion) with mean 0, and $\sigma \in [0.3,8]$ (chosen randomly for each episode). There were 6 different types: ball size $b \in \{2,5,10,20,40,60\}$, with noise levels $n_b = b/2\%$. So for type b, with probability $1-n_b$, the ball moves in the current direction, and with probability n_b, it moves in another random direction. Hence, the type modifies the base transition probabilities (1) by the noise n_b and (2) because the ball sizes determine the locations that result in collisions. We ran two sets of experiments, one with a simulated user and another with human users. In both cases, we ran an initial training phase to learn the observation models corresponding to the transition distributions for each $(b,n_b) \times z$ pair. We used $k = 1$, and $\varepsilon_0 = 0.5$ with ε decreased by 0.1 every episode.

Algorithms. The simulated user policy was the A^* algorithm, where the direction chosen at each step, given speed z, was toward the location, at distance z from the current location, that had the lowest cost. After training, we ran a test phase, during which we chose the type at random and recorded the loss over 5 episodes of BEAT, each individual action set and EXP-3 averaged over 100 runs. Note that the user knows her type but *these algorithms do not*. For the simulated user, the policy used was the same, and for the human experiment, we used three different users, each performing a sequence of many trials involving combinations of ball sizes and speeds.

4.8.5.2 Results

Simulated User. We present three types of plots. The first plot, Figure 4.15 shows the loss of each z for three representative (b,n_b) and illustrates a 'phase-transition' phe-

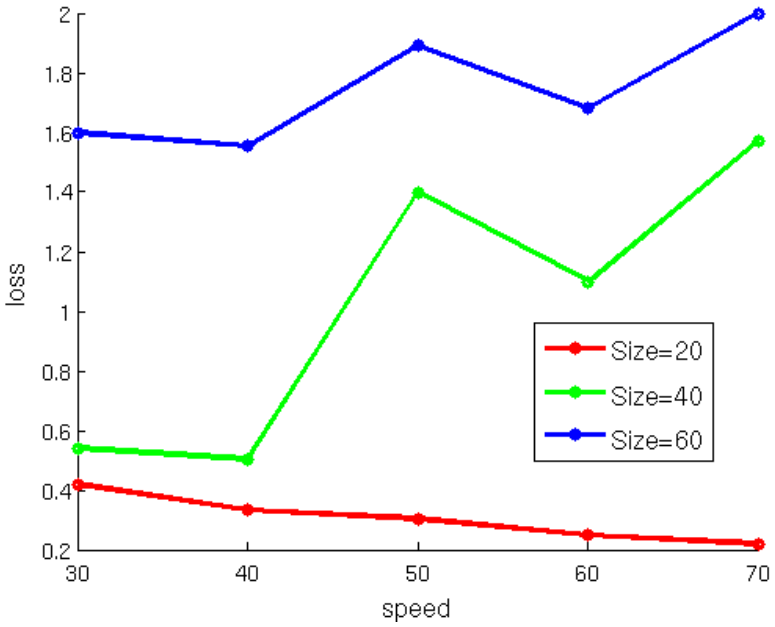

Figure 4.15: Simulated Users. The performance of each speed for each type establishing baseline domain performance.

nomenon on the loss of the speeds for the different sizes: higher speeds tend to work better for smaller sizes and vice versa. Figure 4.16 shows the losses in terms of the *population diversity*. Here, in the x-axis each x corresponds to all possible combinations of x types (so all $\binom{6}{x}$ combinations). This captures different levels of heterogeneity in user populations. In the figure, for the curve α_*, the point $\alpha_*(x)$ gives the loss of the best speed averaged across all populations of size x. Hence EXP-3(x) gives the loss of EXP-3 averaged over all populations of size x, while BEAT(x) gives the loss of BEAT averaged over all populations of size x. This curve shows, that averaged across population sizes, BEAT outperforms EXP-3 significantly. This is compatible with the results seen in the surveillance domain results in Section 4.6.4, particularly the comparison to UCB in Figure 4.11. Furthermore, while for the smaller population size, the α_* outperforms BEAT, for the larger population sizes BEAT outperforms α_* comprehensively, illustrating the need for adaptation.

Finally, Figure 4.17 shows how well BEAT is able to identify types. This plot shows that BEAT has trouble identifying the smallest type. We conjecture that this is because the behaviour of $b = 5$ and $b = 2$ are nearly indistinguishable. However, other than that, it is able to identify the correct type fairly rapidly, almost always halfway through the first episode.

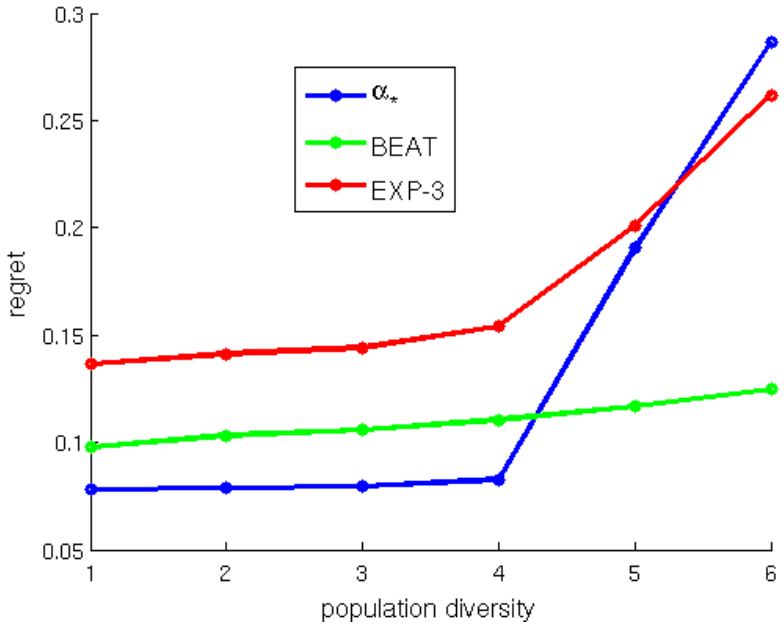

Figure 4.16: Simulated Users. The performance of best static speed α_*, EXP-3 and BEAT for each of 6 possible combinations of type populations when the user is simulated. See text for details.

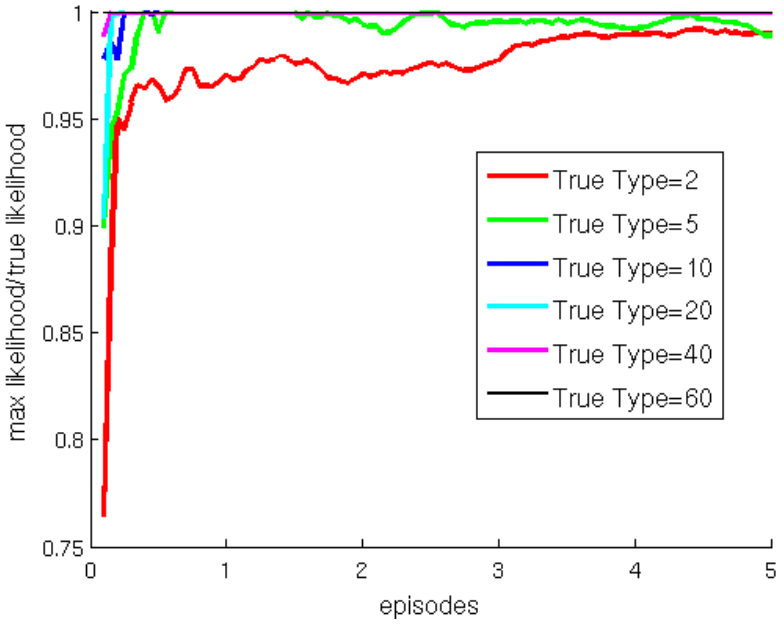

Figure 4.17: Simulated Users. The ratio of the maximum-likelihood type and posterior and the true type in BEAT, averaged over all population-diversity trials in our experiments.

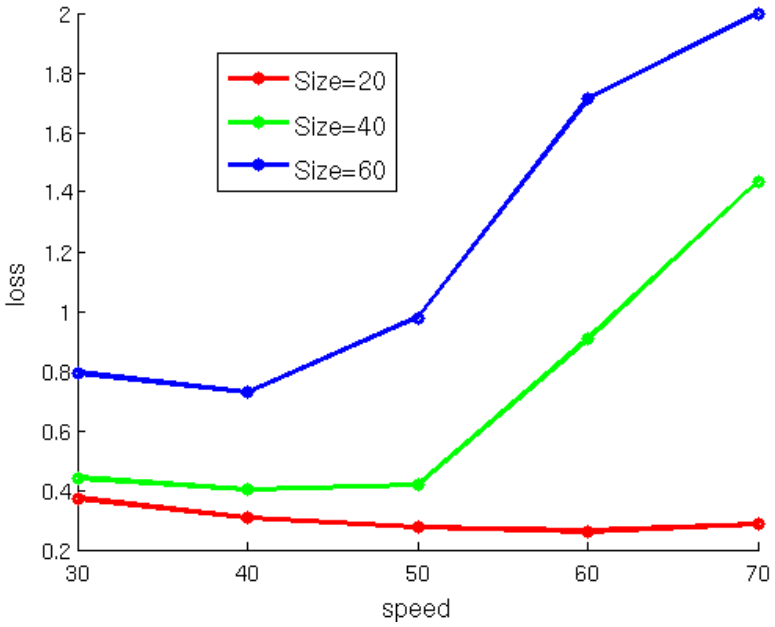

Figure 4.18: Human Users. The performance of each speed for each type, establishing baseline performance in the domain.

Human User. The results for the human experiments are presented as before in three plots. Figure 4.18 shows the phase-transition in the loss at different speeds. Figure 4.19 shows the performance of BEAT and α_* for the human experiments in terms of population diversity. We do not report the performance of EXP-3 because data collection for that algorithm takes time that exceeded the constraints of our human subjects. In these experiments, we see that BEAT significantly outperforms α_* for all population sizes, hence demonstrating the benefit of adaptation with real human subjects, corroborating our more extensive simulation results above. This is a key experimental result of this application domain. Finally, Figure 4.20 shows the efficacy of our algorithm at identifying different types. As in simulated user experiments, our algorithm has some trouble distinguishing between the very nearby types $b = 5$ and $b = 2$ but the error is not beyond the threshold of small noise.

Safety, Efficiency And Speed. As stated above, our goal in designing the algorithm was to ensure safety, efficiency and speed.

Safety: The experiments show asymptotic safety. For simulated users, BEAT does not under-perform w.r.t. any alternatives including EXP-3 and the static best speed α_*. In the human experiments, BEAT always does better than α_* (Figure 4.19). In the artificially constrained low-population-diversity setting (1-3) with simulated users, α_*

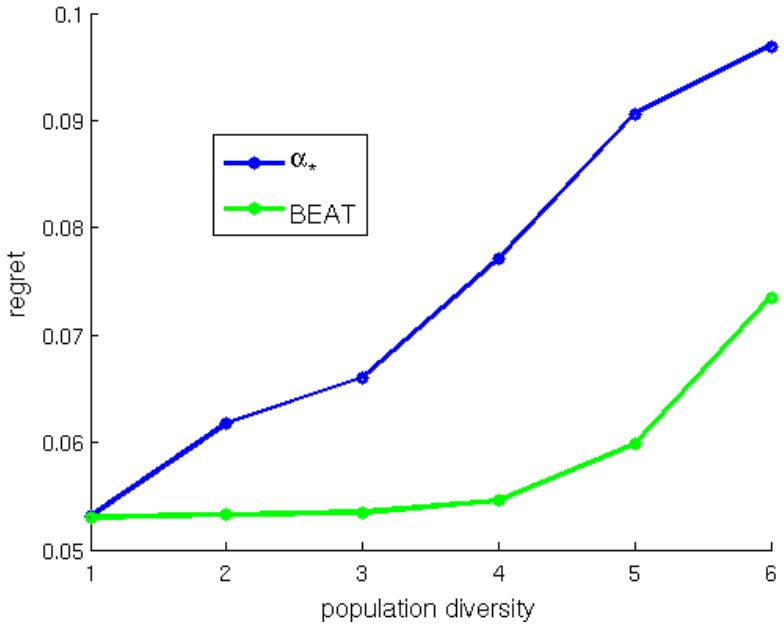

Figure 4.19: Human Users. The performance of best static speed α_* and BEAT for each of 6 possible combinations of population types. See text for details.

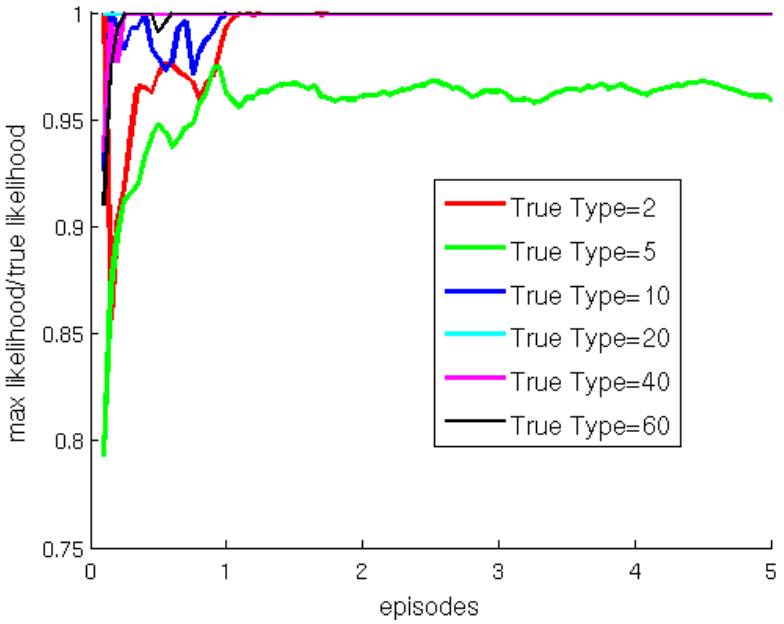

Figure 4.20: Human Users. The ratio of the maximum-likelihood type and posterior and the true type in BEAT, averaged over all population-diversity trials in our experiments when the user is human.

is marginally better than BEAT, but BEAT outperforms at realistic levels of diversity (Figure 4.16).

Efficiency: In the experiments, Figures 4.16 and 4.19 report on the regret, which also shows efficiency in practice.

Speed. Figures 4.17 and 4.20 show that we identify the true types fairly rapidly, with low sample complexity, demonstrating the speed of this approach.

4.9 Conclusion

In this chapter we address the policy reuse problem, which involves being presented with an unknown task instance and being required to select between a number of different policies so as to minimise regret within a short number of episodes. This problem is motivated by many application domains, where tasks have short durations, such as human interaction and personalisation, as well as monitoring tasks. We introduce Bayesian Policy Reuse as a Bayesian framework for solving this problem, by tracking a belief over the space of known tasks which is updated using observation signals and models trained offline for each policy. Several mechanisms for selecting policies from the belief are also described, giving rise to different variants of the core algorithm.

This approach is empirically evaluated in four domains, where we show comparisons between the different variants of BPR, as well as a bandit algorithm and a Bayesian optimisation algorithm. We also show the effect of using different observation signals, and illustrate the trade-off between library size and sample complexity required to achieve a certain level of performance in a task.

Finally as an application, we present a particular instantiation of BPR, known as Bayesian Environment Adaptation with Types (BEAT), which performs online interface adaptation and personalisation based on the performance of a user on a task. This algorithm is shown to possess three key properties in its responses: safety, efficiency, and speed.

Chapter 5

Discussion and Conclusion

5.1 Discussion

In this thesis we set out to explore mechanisms for learning and acting in domains over prolonged periods of time. The ultimate goal is to be able to deploy an agent in some rich environment where it would be expected to solve sequences of tasks, the details of which are unknown *a priori*. To this end, such an agent would need to be able to make fast decisions based on a number of task and domain variables, some of which may not be directly observable. The capabilities and performance of this agent would therefore be greatly enhanced by learning about regularities in the environment, which in turn cause regularities in behaviour.

The work presented in this thesis emphasises three core themes which contribute to the successful operation of lifelong robots.

The first of these themes is the idea of learning the structure of a domain. A better understanding of the general compositions of the family of domains encountered by an agent suggests that better representations could be developed, which in turn means that plans can be better specified and more successfully executed. Ideally, an agent should be equipped with the ability to learn and refine environmental representations autonomously. In Chapter 2, we presented an algorithm that can learn spatial relationships from labelled data, and use this to uncover functional aspects of the topological structure of the objects in a scene. This therefore illustrates one aspect of potentially long-term representation learning.

The second theme relates to learning across tasks, and using the experience gained in previous tasks to guide solutions to new tasks. This also corresponds to learning structure in the domain, albeit more dynamic structure, in that it is a property of the

behaviours of the agent within the domain. By introducing the concept of action priors in Chapter 3, we show how behaviours used in a number of different tasks can be combined, so as to identify the common elements. Together, these form a notion of "common sense" behaviour within the domain, by providing a prior over action selection which is acquired from multiple tasks. This is knowledge which an agent can gradually accumulate over an entire operational lifetime.

The third and final theme concerns the concept of reusing behaviours and selection between them based on environmental cues, rather than learning new ones from scratch. This approach works well when tasks or environments are similar, but they differ in ways which are not directly observable. Solution speeds can be many orders of magnitude faster by trying to find a good policy from a pre-learnt repertoire than to treat each problem as completely new and unrelated to previous experiences. In introducing Bayesian policy reuse in Chapter 4 we show how policies can be reused in this way, in a sample efficient manner.

5.2 Future Work

Each of the methods and techniques described in this thesis present possible avenues for future research, and we describe some of these here, either as extensions to the current work, combinations of the approaches presented herein, or further questions raised by these methods.

Extending the Language of Spatial Relations

The work on spatial relationships is limited to being a proof of concept, in that it only covers the learning of the concepts of "on" and "adjacent". This could be easily extended to include other spatial relationships between objects, such as "under" or "in front of".

In addition, incorporating a logic engine would allow for more complicated relationships to be inferred. For example, "between" would require two different "adjacent" relationships to be identified, or "in" requires the presence of "on" as well as "adjacent".

In the original work, we did not delve deeply into the uses of the topological structures of objects which arose from the contact point networks. These topologies can be used to define a functional similarity between different objects, and thus classify ob-

jects based on the way in which they are used in a scene. This choice of representation would allow for objects to be used in planning, without knowing the identities of the objects themselves. This could provide many advantages to an embodied agent, who could then form and execute plans even if some of the objects involved were unknown.

Action Priors in Different Reasoning Models

Action priors were introduced in this thesis as a means for prioritising action selection based on commonalities between families of tasks. In particular, we used them in the context of reinforcement learning. However, the principle as presented is agnostic to the decision making paradigm.

As a result, action priors are applicable to different action choice frameworks, and in particular it would be interesting to investigate using this principle to guide the search process in planning paradigms, such as Monte-Carlo tree search (Browne et al., 2012).

Context in Bayesian Policy Reuse

The work on Bayesian policy reuse presented in this thesis defines a prior distribution G_0 over the task types. This distribution is used to seed the initial belief: $\beta^0 = G_0$. In our experiments, we assume this distribution to be uniform.

A simple extension could incorporate learning this distribution from the training data. The prior could actually be made more interesting, if it was assumed that this changed over time. If such an adaptation process did occur, this could be tracked, which would lead to learning at multiple time scales: identification of an individual task over that task's duration, vs a slower tracking of the task distribution.

Finally, Bayesian policy reuse could also use external context as a signal for seeding the prior distribution to result in even faster convergence. This would be useful for cases where there are different families of tasks, such as for an assistive robot in a hospital dealing with the requests of either a patient or a doctor. This task seeding would work using the principle of action priors.

Closing the Loop

In Section 3.3.2 we touch on the issue of feature selection, by examining the entropy of the action priors over each feature in a feature set.

As an avenue for further research, it would be interesting to further investigate ways in which one could close the perception-action loop by using perceptual signals to identify candidate regions for policy learning, and in so doing establish a form of *artificial curiosity*.

The entropy in the action priors for a particular context provides an indication that there is scope for further learning. As we have already shown this to be useful for feature selection, so too could it indicate a candidate situation for learning a new behaviour or action.

Furthermore, if being presented with a new task in Bayesian policy reuse results in a belief spread over more than one of the pre-learnt types, then this suggests that the new task may have been drawn from a previously unseen type. Again, this indicates an occasion for learning the new type, as well as a corresponding optimal policy.

Systems Integration

One major avenue to be explored following the ideas presented in this thesis lies in their integration and subsequent implementation on robot hardware, and to actually deploy it into a real-world multitask domain.

The three primary threads developed in this thesis can be naturally combined. The spatial relationships and topological abstractions of objects presented in Chapter 2 provide a general basis for observation features to be used with the action priors. The action priors in turn can be used to provide context-specific priors for Bayesian policy reuse.

Together, these provide a solid framework for decision making in multitask environments. Implementing these as a complete system on a multi-purpose hardware platform should enable a robot to select between and execute a large number of behaviours quickly under novel conditions.

5.3 Conclusion

In order to build robotic systems which are capable of being deployed for long periods of time and working on a wide range of tasks in their environment, many of which may not have been conceptualised at the time of deployment, we require learning systems which have been developed with these specifications in mind.

This is a problem for which the human brain seems highly skilled. As humans, we

are able to generalise from experiences and use abstracted features of environments, together with sparse feedback from the environment, to make competent decisions in a wide range of novel situations.

This research has thus focused on the ideas of learning general domain knowledge, and reusing both this knowledge and previously learnt behaviours so as to accelerate the completion of new tasks. We have introduced algorithms for learning the spatial structure between different objects, for guiding exploration towards behaviours that have worked well in the past, and for using sparse signals and expected performance to inform policy selection.

The common theme demonstrated throughout this thesis is the strength of *abstractions* in allowing agents to learn and make decisions rapidly in new situations. We have shown how, in a number of different contexts, agents can acquire general domain knowledge from multiple instances of different tasks in the same domain. This loosely corresponds to "common sense" knowledge in humans, which we use to provide a set of behavioural guidelines for dealing with both familiar and foreign tasks and situations. In this work, we have shown abstractions for describing relationships between physical objects, for describing the functional topologies of objects, for prioritising actions based on successful choices in similar situations in the past, and for describing families of related tasks such that response policies can be selected quickly.

Three major final questions emerge when considering the ideas encapsulated in this thesis. Firstly, what other abstractions and "common sense" knowledge can and should be extracted from a domain to continually improve the performance of a lifelong agent? Clearly there are a number of other aspects of general environments to be modelled and abstracted, such as general object models, or principles of causality. Is it then possible to enumerate a complete list of the forms of foundational and general knowledge required by an agent to exist autonomously in a dynamic environment with evolving task descriptions?

Finally, we note that the different abstractions discussed within this thesis are disjoint in the ways in which they are learnt and represented. In order to develop the assistive hospital robots outlined in Chapter 1, it is important that the research community extend this line of inquiry into abstractions, and address the question of how one could develop a single unified framework for acquiring and utilising the many forms of knowledge such as those introduced herein.

This thesis presents a step towards general purpose lifelong robots, and it is our hope that future work can build on these ideas, so that we may one day soon have

robots which are able to be deployed into common environments such as homes and hospitals, and as such can augment the capabilities of the humans already present there, in meaningful and long-term support roles.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship Learning via Inverse Reinforcement Learning. *International Conference on Machine Learning*.

Alexe, B., Deselaers, T., and Ferrari, V. (2010). What is an object? *International Conference on Computer Vision and Pattern Recognition*, pages 73–80.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77.

Bandyopadhyay, T., Won, K. S., E. Frazzoli, D. H., Lee, W. S., and Rus, D. (2012). Intention-Aware Motion Planning. In *Proceedings Workshop of Algorithmic Foundations, WAFR-2012*.

Barck-Holst, C., Ralph, M., Holmar, F., and Kragic, D. (2009). Learning Grasping Affordance Using Probabilistic and Ontological Approaches. *International Conference on Advanced Robotics*.

Barto, A. G. and Mahadevan, S. (2003). Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Systems Journal*, 13:41–77.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum Learning. *International Conference on Machine Learning*.

Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*. Springer.

Bianchi, R. A. C., Ribeiro, C. H. C., and Costa, A. H. R. (2007). Heuristic Selection of Actions in Multiagent Reinforcement Learning. *International Joint Conference on Artificial Intelligence*, pages 690–695.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bookstaber, R. and Langsam, J. (1985). On the optimality of coarse behavior rules. *Journal of Theoretical Biology*, 116(2):161–193.

Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology*. MIT press.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Brooks, R. (1986). A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*.

Calabar, P. and Santos, P. E. (2011). Formalising the Fisherman's Folly puzzle. *Artificial Intelligence*, 175(1):346–377.

Carlsson, G. (2009). Topology and Data. *Bulletin of the American Mathematical Society*, 46(2):255–308.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction Learning and Games*. Cambridge University Press.

Charniak, E. and Goldman, R. (1991). A probabilistic model of plan recognition. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 1*, AAAI'91.

Chickering, D. M. and Paek, T. (2007). Personalizing influence diagrams: applying online learning strategies to dialogue management. *User Modeling and User-Adapted Interaction*, 17(1-2):71–91.

Cook, D. J. and Holder, L. B. (2007). *Mining Graph Data*. John Wiley and Sons.

Dearden, R. and Burbridge, C. (2013). Manipulation planning using learned symbolic state abstractions. *Robotics and Autonomous Systems*.

Dearden, R., Friedman, N., and Andre, D. (1999). Model based Bayesian exploration. In *Proceedings of the fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc.

Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian q-learning. In *AAAI/IAAI*, pages 761–768.

Dee, H. M., Hogg, D. C., and Cohn, A. G. (2009). Scene Modelling and Classification Using Learned Spatial Relations. *COSIT-09, Lecture Notes in Computer Science*, (5756):295–311.

Desai, C., Ramanan, D., and Fowlkes, C. (2009). Discriminative models for multi-class object layout. *International Conference on Computer Vision*, pages 229–236.

Dey, D., Liu, T. Y., Hebert, M., and Bagnell, J. A. (2012a). Contextual Sequence Prediction with Application to Control Library Optimization. *Robotics: Science and Systems*.

Dey, D., Liu, T. Y., Sofman, B., and Bagnell, J. A. (2012b). Efficient Optimization of Control Libraries. *AAAI*, pages 1983–1989.

Dimitrakakis, C. (2006). Nearly optimal exploration-exploitation decision thresholds. In *Artificial Neural Networks–ICANN 2006*, pages 850–859. Springer.

Donald, B. R. (1995). On information invariants in robotics. *Artificial Intelligence*, 72(1):217–304.

Doshi-Velez, F., Wingate, D., Roy, N., and Tenenbaum, J. B. (2010). Nonparametric Bayesian policy priors for reinforcement learning. *Advances in Neural Information Processing Systems*.

Engel, Y. and Ghavamzadeh, M. (2007). Bayesian policy gradient algorithms. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, volume 19, page 457. MIT Press.

Erez, T. and Smart, W. D. (2008). What does Shaping Mean for Computational Reinforcement Learning? *International Conference on Development and Learning*, pages 215–219.

Fern, A. and Tadepalli, P. (2010a). A Computational Decision Theory for Interactive Assistants. *Advances in Neural Information Processing Systems*.

Fern, A. and Tadepalli, P. (2010b). A Computational Decision Theory for Interactive Assistants, Advances in Neural Information Processing Systems. In *Proceedings of the* 23^{rd} *Conference on Neural Information Processing Systems*.

Fernandez, F. and Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*.

Fichtl, S., Guerin, F., Mustafa, W., Kraft, D., and Krueger, N. (2013). Learning Spatial Relations between Objects From 3D Scenes. *Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL)*.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288.

Foster, D. and Dayan, P. (2002). Structure in the Space of Value Functions. *Machine Learning*, 49:325–346.

Friston, K. J., Daunizeau, J., and Kiebel, S. J. (2009). Reinforcement learning or active inference? *PloS one*, 4(7):e6421.

Gajos, K., Wobbrock, J., and Weld, D. (2008). Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1257–1266.

Galata, A., Cohn, A. G., Magee, D. R., and Hogg, D. C. (2002). Modeling interaction using learnt qualitative spatio-temporal relations and variable length markov models. *European Conference on Artificial Intelligence*, pages 741–746.

Galleguillos, C. and Belongie, S. (2010). Context based object categorization: A critical survey. *Computer Vision and Image Understanding*, 114(6):712–722.

Galleguillos, C., Rabinovich, A., and Belongie, S. (2008). Object Categorization using Co-Occurence, Location and Appearance. *International Conference on Computer Vision and Pattern Recognition*.

Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Gibson, J. J. (1986). *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, Inc., 2nd edition.

Ginebra, J. and Clayton, M. K. (1995). Response surface bandits. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 771–784.

Gittins, J. C. and Jones, D. (1974). A dynamic allocation index for the discounted multiarmed bandit problem. *Progress in Statistics*, pages 241–266.

Gobet, F. and Simon, H. A. (1996). Templates in chess memory: A mechanism for recalling several boards. *Cognitive psychology*, 31(1):1–40.

Harre, M., Bossomaier, T., and Snyder, A. (2012). The Perceptual Cues that Reshape Expert Reasoning. *Scientific Reports*, 2(502).

Hauser, J. R., Urban, G. L., Liberali, G., and Braun, M. (2009). Website Morphing. *Marketing Science*, 28(2):202–223.

Hauser, K. and Latombe, J.-C. (2010). Multi-Modal Motion Planning in Non-Expansive Spaces. *International Journal of Robotics Research*, 29(7):897–915.

Havoutis, I. and Ramamoorthy, S. (2013). Motion planning and reactive control on learnt skill manifolds. *The International Journal of Robotics Research*, 32(9-10):1120–1150.

Hester, T. and Stone, P. (2009). An empirical comparison of abstraction in models of Markov decision processes. *Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning*, pages 18–23.

Hoey, J., Poupart, P., von Bertoldi, A., Craig, T., Boutilier, C., and Mihailidis, A. (2010). Automated Handwashing Assistance for Persons with Dementia Using

Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding*, 114(5):503–519.

Holte, R. C. and Choueiry, B. Y. (2003). Abstraction and reformulation in artificial intelligence. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1435):1197–1204.

Horvitz, E. J. and Klein, A. C. (1993). Utility-Based Abstraction and Categorization. *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 128–135.

Huys, Q. J., Eshel, N., O'Nions, E., Sheridan, L., Dayan, P., and Roiser, J. P. (2012). Bonsai trees in your head: how the Pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS computational biology*, 8(3):e1002410.

Jain, A. K. and Dorai, C. (2000). 3D object recognition: Representation and matching. *Statistics and Computing*, 10(2):167–182.

Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2011). *Recommender Systems An Introduction*. Cambridge University Press.

Jiang, X., Bowyer, K., Morioka, Y., Hiura, S., Sato, K., Inokuchi, S., Bock, M., Guerra, C., Loke, R. E., and du Buf, J. M. H. (2000). Some further results of experimental comparison of range image segmentation algorithms. *International Conference on Pattern Recognition*, 4:877–881.

Joachims, T. (1999). *Making Large-Scale SVM Learning Practical*, chapter 11. Advances in Kernel Methods - Support Vector Learning. MIT Press.

Jong, N. K. and Stone, P. (2005). State Abstraction Discovery from Irrelevant State Variables. *International Joint Conference on Artificial Intelligence*, pages 752–757.

Kaelbling, L. P. (1990). *Learning in Embedded Systems*. PhD thesis, Stanford University.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134.

Katz, D. and Brock, O. (2008). Manipulating articulated objects with interactive perception. *International Conference on Robotics and Automation*, pages 272–277.

Kenney, J., Buckley, T., and Brock, O. (2009). Interactive Segmentation for Manipulation in Unstructured Environments. *International Conference on Robotics and Automation*, pages 1343–1348.

Knox, W. B. and Stone, P. (2009). Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. *International Conference on Knowledge Capture*.

Kober, J. and Peters, J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, 84(1-2):171–203.

Konidaris, G. D. (2011). *Autonomous robot skill acquisition*. PhD thesis, University of Massachusetts Amherst.

Konidaris, G. D. and Barto, A. G. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning*, pages 489–496.

Koos, S., Cully, A., and Mouret, J.-B. (2013). High resilience in robotics with a multi-objective evolutionary algorithm. *Proceedings of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 31–32.

Kuipers, B. (1994). *Qualitative reasoning: modeling and simulation with incomplete knowledge*. The MIT Press.

Kuipers, B. J., Beeson, P., Modayil, J., and Provost, J. (2006). Bootstrap Learning of Foundational Representations. *Connection Science*, 18(2):145–158.

Lai, T. L. and Robbins, H. (1978). Adaptive design in regression and control. *Proceedings of the National Academy of Sciences*, 75(2):586–587.

Lang, T. and Toussaint, M. (2009). Relevance Grounding for Planning in Relational Domains. *European Conference on Machine Learning*.

Lazaric, A. (2008). *Knowledge transfer in reinforcement learning*. PhD thesis, Politecnico di Milano.

Leffler, B. R., Littman, M. L., and Edmunds, T. (2007). Efficient Reinforcement Learning with Relocatable Action Models. *AAAI*, pages 572–577.

Leibe, B. and Schiele, B. (2004). Scale-Invariant Object Categorization Using a Scale-Adaptive Mean-Shift Search. *Lecture Notes in Computer Science*, 3175:145–153.

Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a Unified Theory of State Abstraction for MDPs. *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539.

Liao, L., Patterson, D. J., Fox, D., and Kautz, H. (2007). Learning and Inferring Transportation Routines. *Artificial Intelligence*, 171:311–331.

Mahmud, M., Rosman, B., Ramamoorthy, S., and Kohli, P. (2014). Adapting interaction environments to diverse users through online action set selection. *In Proc. AAAI Workshop on Machine Learning for Interactive Systems (AAAI-MLIS)*.

Mahmud, M. M. H., Hawasly, M., Rosman, B., and Ramamoorthy, S. (2013). Clustering Markov Decision Processes For Continual Transfer. *arXiv preprint arXiv:1311.3959*.

Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., and Abbeel, P. (2010). Cloth Grasp Point Detection based on Multiple-View Geometric Cues with Application to Robotic Towel Folding. *International Conference on Robotics and Automation*, pages 2308–2315.

Martin, J. J. (1965). *Some Bayesian decision problems in a Markov chain*. PhD thesis, Massachusetts Institute of Technology.

Meltzoff, A. N., Kuhl, P. K., Movellan, J., and Sejnowski, T. J. (2009). Foundations for a New Science of Learning. *Science*, 325(5938):284–288.

Mersereau, A. J., Rusmevichientong, P., and Tsitsiklis, J. N. (2009). A structured multiarmed bandit problem and the greedy policy. *Automatic Control, IEEE Transactions on*, 54(12):2787–2802.

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

Mourão, K., Zettlemoyer, L. S., Petrick, R., and Steedman, M. (2012). Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*.

Niño-Mora, J. (2011). Computing a classic index for finite-horizon bandits. *INFORMS Journal on Computing*, 23(2):254–267.

Oates, J. T. (2001). *Grounding knowledge in sensors: Unsupervised learning for language and planning*. PhD thesis, University of Massachusetts Amherst.

Ong, S. C. W., Png, S. W., Hsu, D., and Lee, W. S. (2010). Planning under Uncertainty for Robotic Tasks with Mixed Observability. *I. J. Robotic Res.*, 29(8):1053–1068.

Ortega, P. A. and Braun, D. A. (2013). Generalized thompson sampling for sequential decision-making and causal inference. *arXiv preprint arXiv:1303.4431*.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *Evolutionary Computation, IEEE Transactions on*, 11(2):265–286.

Pandey, S., Chakrabarti, D., and Agarwal, D. (2007). Multi-armed bandit problems with dependent arms. In *Proceedings of the 24th international conference on Machine learning*, pages 721–728. ACM.

Pasula, H. M., Zettlemoyer, L. S., and Kaelbling, L. P. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29(1):309–352.

Pelleg, D. and Moore, A. W. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. *International Conference on Machine Learning*, pages 727–734.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

Pickett, M. and Barto, A. G. (2002). PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. *International Conference on Machine Learning*, pages 506–513.

Pierce, D. and Kuipers, B. J. (1997). Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–227.

Pinker, S. (1999). How the mind works. *Annals of the New York Academy of Sciences*, 882(1):119–127.

Pinz, A., Bischof, H., Kropatsch, W., Schweighofer, G., Haxhimusa, Y., Opelt, A., and Ion, A. (2008). Representations for Cognitive Vision: A Review of Appearance-Based, Spatio-Temporal, and Graph-Based Approaches. *Electronic letters on computer vision and image analysis*, 7(2):35–61.

Pólya, G. (1945). *How to solve it: A new aspect of mathematical method*. Princeton University Press.

Powell, W. B. (2010). The knowledge gradient for optimal learning. *Wiley Encyclopedia of Operations Research and Management Science*.

Precup, D., Sutton, R. S., and Singh, S. (1998). Theoretical results on reinforcement learning with temporally abstract options. *European Conference on Machine Learning*.

Price, B. and Boutilier, C. (2003). A bayesian approach to imitation in reinforcement learning. *IJCAI*, pages 712–720.

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

Ramamoorthy, S. (2007). *Task encoding, motion planning and intelligent control using qualitative models*. PhD thesis, The University of Texas at Austin.

Randell, D. A., Cui, Z., and Cohn, A. G. (1992). A Spatial Logic Based on Regions and Connection. *International Conference on Knowledge Representation and Reasoning*, pages 165–176.

Ravindran, B. and Barto, A. G. (2003). Relativized Options: Choosing the Right Transformation. *Proceedings of the Twentieth International Conference on Machine Learning*.

Reddi, S. and Brunskill, E. (2012). Incentive Decision Processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Rosman, B. S. and Ramamoorthy, S. (2010). A Game-Theoretic Procedure for Learning Hierarchically Structured Strategies. *IEEE International Conference on Robotics and Automation*.

Rosman, B. S. and Ramamoorthy, S. (2011). Learning spatial relationships between objects. *International Journal of Robotics Research*, 30(11):1328–1342.

Rosman, B. S. and Ramamoorthy, S. (2012a). A Multitask Representation using Reusable Local Policy Templates. *AAAI Spring Symposium Series on Designing Intelligent Robots: Reintegrating AI*.

Rosman, B. S. and Ramamoorthy, S. (2012b). What good are actions? Accelerating learning using learned action priors. *International Conference on Development and Learning and Epigenetic Robotics*.

Rosman, B. S. and Ramamoorthy, S. (2014). Giving Advice to Agents with Hidden Goals. *IEEE International Conference on Robotics and Automation*.

Rosman, B. S., Ramamoorthy, S., Mahmud, M. M. H., and Kohli, P. (2014). On User Behaviour Adaptation Under Interface Change. *Proc. International Conference on Intelligent User Interfaces (IUI)*.

Rusu, R. B., Holzbach, A., Diankov, R., Bradski, G., and Beetz, M. (2009). Perception for Mobile Manipulation and Grasping using Active Stereo. *Humanoids*, pages 632–638.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135.

Saxena, A., Driemeyer, J., and Ng, A. Y. (2008). Robotic Grasping of Novel Objects using Vision. *International Journal of Robotics Research*, 27(2):157–173.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2004). Learning Movement Primitives. *International Symposium on Robotics Research*.

Schmill, M. D., Oates, T., and Cohen, P. R. (2000). Learning Planning Operators in Real-World, Partially Observable Environments. *International Conference on Artificial Planning and Scheduling*, pages 246–253.

Seuken, S., Parkes, D. C., Horvitz, E., Jain, K., Czerwinski, M., and Tan, D. S. (2012). Market user interface design. In *ACM Conference on Electronic Commerce*, pages 898–915.

Shani, G., Heckerman, D., and Brafman, R. I. (2005). An MDP-Based Recommender System. *Journal of Machine Learning Research*, 6:1265–1295.

Sherstov, A. A. and Stone, P. (2005). Improving Action Selection in MDP's via Knowledge Transfer. *AAAI*, pages 1024–1029.

Simon, H. A. (1955). A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological review*, 63(2):129.

Simon, H. A. (1992). What is an "explanation" of behavior? *Psychological Science*, 3(3):150–161.

Simon, H. A. and Chase, W. G. (1973). Skill in Chess: Experiments with chess-playing tasks and computer simulation of skilled performance throw light on some human perceptual and memory processes. *American Scientist*, 61(4):394–403.

Sisbot, E. A., Marin-Urias, L. F., Alami, R., and Simeon, T. (2007). A Human Aware Mobile Robot Motion Planner. *IEEE Transactions on Robotics*, 23(5):874–883.

Sjöö, K. (2011). *Functional understanding of space: Representing spatial knowledge using concepts grounded in an agent's purpose*. PhD thesis, KTH Royal Institute of Technology.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

Strevens, M. (2013). *Tychomancy*. Harvard University Press.

Sun, J., Moore, J. L., Bobick, A., and Rehg, J. M. (2010). Learning Visual Object Categories for Robot Affordance Prediction. *The International Journal of Robotics Research*, 29(2-3):174–197.

Sunmola, F. T. (2013). *Optimising learning with transferable prior information*. PhD thesis, University of Birmingham.

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

Syed, U. and Schapire, R. E. (2008). A Game-Theoretic Approach to Apprenticeship Learning. *Advances in Neural Information Processing Systems*.

Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10:1633–1685.

Tenenbaum, J. B., Kemp, C. C., Griffiths, T. L., and Goodman, N. D. (2011). How to Grow a Mind: Statistics, Structure, and Abstraction. *Science*, 331:1279–1285.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294.

Thrun, S. (1996a). Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems*, pages 640–646.

Thrun, S. (1996b). Learning to learn: Introduction. In *Learning To Learn*. Citeseer.

Torrey, L. and Taylor, M. E. (2013). Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. *International Conference on Autonomous Agents and Multiagent Systems*.

Valtazanos, A. and Ramamoorthy, S. (2013). Evaluating the effects of limited perception on interactive decisions in mixed robotic environments. In *HRI '13: Proc. ACM/IEEE International Conference on Human-Robot Interaction*.

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *Machine Learning: ECML 2005*, pages 437–448. Springer.

Waltz, D. L. (1975). *Understanding Line Drawings of Scenes with Shadows*, pages 19–92. The Psychology of Computer Vision. McGraw-Hill.

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

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-task reinforcement learning: a hierarchical Bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 1015–1022. ACM.

Wingate, D., Goodman, N. D., Roy, D. M., Kaelbling, L. P., and Tenenbaum, J. B. (2011). Bayesian Policy Search with Policy Priors. *International Joint Conference on Artificial Intelligence*.

Wyatt, J. (1997). *Exploration and inference in learning from reinforcement*. PhD thesis, University of Edinburgh.

Zhang, H., Chen, Y., and Parkes, D. C. (2009). A General Approach to Environment Design with One Agent. In *IJCAI*, pages 2002–2014.

Zucker, J.-D. (2003). A grounded theory of abstraction in artificial intelligence. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1435):1293–1309.