UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering Science and Maths
School of Electronics and Computer Science

Bayesian learning for multi-agent coordination

by

Mair Allen-Williams

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

March 2009

UNIVERSITY OF SOUTHAMPTON

Abstract

Faculty of Engineering Science and Maths
School of Electronics and Computer Science

Doctor of Philosophy

BAYESIAN LEARNING FOR AGENT COORDINATION

by Mair Allen-Williams

Multi-agent systems draw together a number of significant trends in modern technology: ubiquity, decentralisation, openness, dynamism and uncertainty. As work in these fields develops, such systems face increasing challenges. Two particular challenges are decision making in uncertain and partially-observable environments, and coordination with other agents in such environments. Although uncertainty and coordination have been tackled as separate problems, formal models for an integrated approach are typically restricted to simple classes of problem and are not scalable to problems with tens of agents and millions of states.

We improve on these approaches by extending a principled Bayesian model into more challenging domains, using Bayesian networks to visualise specific cases of the model and thus as an aid in deriving the update equations for the system. One approach which has been shown to scale well for networked offline problems uses finite state machines to model other agents. We used this insight to develop an approximate scalable algorithm applicable to our general model, in combination with adapting a number of existing approximation techniques, including state clustering.

We examine the performance of this approximate algorithm on several cases of an urban rescue problem with respect to differing problem parameters. Specifically, we consider first scenarios where agents are aware of the complete situation, but are not certain about the behaviour of others; that is, our model with all elements but the actions observable. Secondly, we examine the more complex case where agents can see the actions of others, but cannot see the full state and thus are not sure about the beliefs of others. Finally, we look at the performance of the partially observable state model when the system is dynamic or open. We find that our best response algorithm consistently outperforms a handwritten strategy for the problem, more noticeably as the number of agents and the number of states involved in the problem increase.

Contents

List of Figures

List of Examples

List of Algorithms

Declaration of Authorship

I, Mair Allen-Williams, declare that this thesis titled, ‘Bayesian learning for multi-agent coordination’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ________________________________

Date: ________________________________

Acknowledgements

Thanks are due to many people for the production of this thesis...

- many thanks are due to my supervisor Nicholas Jennings for his patience and excellent help and advice.
- ... to BAE Systems and EPSRC for providing funding under the Aladdin Project
- ... to Paul for helping to edit a number of the diagrams
- ... to my sister Catrin for friendship, chocolate and proofreading
- ... to ECSWomen and Reena Pau for support along the way
- ... to Mike for many cheering emails
- ... to my godfamily for their homes
- ... to my family for their support through many years of education
- ... to the many people on IRC far and near who have listened to me rant late into the night

Abbreviations

MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
POSG	Partially Observable Stochastic Game
RL	Reinforcement Learning
NE	Nash Equilibrium
FSM	Finite State Machine
CPT	Conditional Probability Table
HMM	Hidden Markov Model
DBN	Dynamic Bayesian Network

Notation

$a_i, \mathbf{a}, A$	the action of agent i, a joint action, the set of actions
$Act(n)$	the action associated with node n of a finite state machine
b_i, b	the belief state of agent i, a belief state
$BR_i(\Sigma, b)$	agent i's best response to (Σ, b)
C	a cluster (of states, observations, ...)
$C(s)$	the equivalence class for a string s
F	a finite state machine
g	state (of a "grand MDP") comprising underlying state, parameters, models
h	historical information
l_i, L	location of an agent, set of locations
	or length of i, size of an observation space (in sparse prior formulation)
M	a model
n	the node of a finite state machine
n_x	the number of x
$o, o_i, o_{a,i}$ (or obs), O	an observation, the observation of agent i, the observation of agent i about variable a, the set of observations
O_f	an observation function
$Obs(s)$	the observations which can arise from state s
os, OS	a string (sequence) of observations, the set of observation strings
$P(x)$	probability of x
p, P	prior beliefs, the set of prior beliefs
$Q(a, s)$	the long-term value of being in state s and executing action a
r, R	a reward, the set of rewards
R_f	a reward function (from states or transitions to rewards)
s, S	a state, the set of states
t	time
T_f	a transition function
$V(s)$	the long-term value of state s

w_i, w, W	weight of an individual value in a multinomial distribution, weight vector, variable referring to weight vectors
z	a normalising constant
α	the Dirichlet parameter vector, or a learning rate
γ	an agent's degree of 'myopia'
$\delta(A = B)$	the delta function
π_j	the strategy of the single agent j
θ	the environmental dynamics

Key terms

Agent An agent receives input from the environment through its sensors and interacts with the environment to try and achieve some goal: it may be a person, an organisation, or an automatic device such as an intelligent sensor or a robot.

Bayesian probability is an interpretation of probability which describes probability as a "personal belief", based on combining any prior information with observed information.

Bayes' rule is the equation specifying how to update beliefs about the world, given new information:
$P(world = w|observations) \propto P(observations|world = w)P(world = w)$

Belief state A belief state encapsulates the beliefs an agent has about its current state: that is, probability distributions for each variable within the state.

Coordination If several agents are interacting with the same environment, their actions affect one another, directly or indirectly—this is coordination.

Disaster Response Large-scale disasters include earthquake, fire and terrorist attack, and require a timely coordinated multi-agency response.

Finite state machine A finite state machine has a set of internal states, and rules for movement between these internal states. When describing the behaviour of an intelligent agent, internal states prescribe actions, and movement between states is conditioned on observations from the environment.

Markov decision process In a Markov Decision Process, changes in the environment in response to an agent's actions are determined only by the immediate state and actions, and not by any historical information.

Uncertainty An agent in an uncertain environment does not know of all the parameters within that environment.

"It is not the place where you are that is the important thing. It is the intensity of your presence there."

Michael Quoist: A biography. Quoted in 'The Choice', Sister Kirsty

For the daisies on the wayside.

Chapter 1

Introduction

A multi-agent system is a system of interacting intelligent actors, or *agents*, responding to their environment. Among other things, multi-agent technology can be used to model or to implement large decentralised systems. As computing power and ubiquity increase, many organisations are making use of such systems: example application areas are as diverse as modelling eBay auctions (Rogers, David, Schiff, & Jennings, 2007), modelling social structures (Sun & Naveh, 2004), or creating fight scenes in films (for example, agent systems were used in *The Lord of the Rings*[1]). Consequently, scalable multi-agent technology is becoming increasingly important and multi-agent research is a lively and growing area facing many challenges. In particular, the inherent dynamism in many of these problems calls not for offline computation of solutions to problems, but rather timely online responses to new, unknown scenarios.

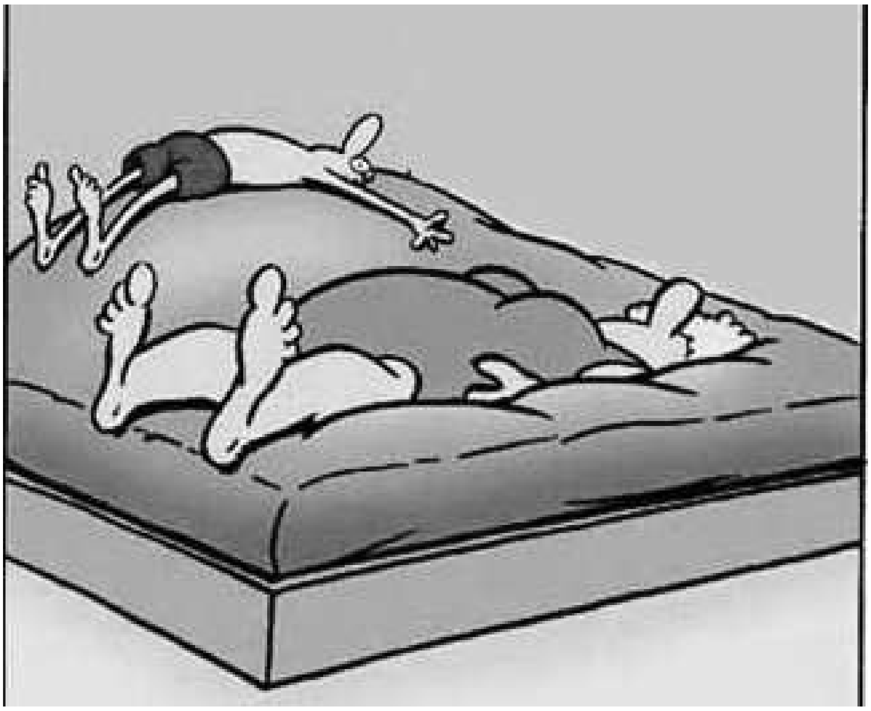
In more detail, agents acting in such unknown scenarios will frequently be *uncertain*, both about the current environment and about the behaviour of other agents. Specifically, when agents are not able to see all aspects of their current situation, the scenario is described as *partially observable* (Kaelbling, Littman, & Cassandra, 1998). In a partially observable setting, the agents must carry out a discovery phase to learn about the scenario before they can focus directly on their goals. As the scenario becomes clearer, the agent must strike a balance

[1] `http://www.massivesoftware.com/`

between exploiting its current knowledge, and exploring further: the "exploration-exploitation" tradeoff (Sutton & Barto, 1998). Now, a principled way to address this tradeoff is to make use of *Bayesian* techniques, which provide a way to incorporate the probable value of information into computations about action quality (Dearden, Friedman, & Andre, 1999).

Now, in a multi-agent system, an agent is always acting in the context of other agents, and so it must adapt its plans according to its expectations of the others. This need to take others into account, *coordination*, is therefore a key issue in a multi-agent system. In particular, in uncertain and open systems, the protocols for coordination must function against a background where agents are not fully aware of the situation, the resources available to them, or the presence or goals of the other agents. Just as exploration and exploitation are entwined, so the negotiation of coordinated behaviour in such systems is intertwined with the discovery phase, including discovering the other agents and learning about their behaviour.

Considering these interlinked issues of uncertainty and coordination, we will build upon existing techniques for decision making under uncertainty, including explicit models of other agents, thus tackling the problem of providing coordinated behaviour in uncertain and partially observable multi-agent systems. Moreover, acknowledging the growth of multi-agent systems in real and increasingly large applications, we will endeavour to factor issues associated with scalability into our solutions. First, in section 1.1, we introduce multi-agent systems in more detail, going on to identify the disaster response domain as a grounding example. Section 1.2 highlights the salient features of this domain. Given this background, in section 1.3 we introduce Bayesian learning as a suitable technique for learning and acting in multi-agent systems and in section 1.4 identify our contributions to the state of the art in this area. Finally, section 1.5 outlines the rest of the thesis.



The woman's greater weight causes her to have a much larger sphere of influence than the man. Each time she moves he will be affected by her actions, and have to act himself if he is to maintain a comfortable position. (Image from `http://babbyhageman.blogspot.com` with thanks)

Figure 1.1: Spheres of influence on a waterbed

1.1 Multi-agent systems

In an agent system, an intelligent agent is functioning in a dynamic environment. This environment provides stimulation to the agent's senses, to which the agent responds by acting on the environment. If the system is a multi-agent system, then many agents coexist in the same environment, and the actions of one agent can cause perceptible changes in another's environment (figure 1.1). Such multi-agent systems are becoming increasingly prevalent, as a result of a number of significant trends in modern technology (Wooldridge, 2002):

- **Ubiquity:** As computing chips become smaller and cheaper, it is possible to add computational power and intelligence to many kinds of devices in almost any location. Systems made of networks of these ubiquitous devices have much greater possibilities than individual devices. These systems may be mobile, in which case they must be able to adapt quickly to changing surroundings.

- **Decentralisation:** With the advent of the world wide web and other computing networks, such as grid computing and peer-to-peer networks, systems that distribute data and tasks among a network of machines are increasingly common.

- **Openness and dynamism:** Open systems are those in which agents may enter or leave at any time, while in dynamic systems the environment properties may change at any time. Many real-world systems are both open and dynamic and there has been a corresponding trend in computing towards providing interactive systems which are able to respond to a changing environment.

- **Uncertainty:** Uncertainty plays a large part in systems which respond to environmental or sensor inputs. Moreover, a trend towards increasingly large and complex systems means that frequently systems are effectively uncertain, even if they are technically deterministic.

The combination of these features describes a broad class of complex, dynamic, large-scale systems which may be implemented or modelled as multi-agent systems. As well as having the features above—decentralisation, openness and dynamism, and uncertainty—these multi-agent systems may be *heterogeneous*, containing agents with a variety of capabilities and goals representing different nodes in the system. For example, a sensor network may contain wind sensors, temperature sensors, and pressure sensors. Each sensor may have to make decisions about when and what to sense based on its own battery power, with different sensors having different kinds of battery. In another example, if the heterogeneous agents have conflicting goals—such as two PDA devices interacting on behalf of two professors, one of whom wants a free day to do some marking while the other wants to set up a meeting—then the agents are considered to be *competitive* (Tambe et al., 2006). Finally, networked systems operating under time constraints may have *bandwidth* *limitations* which must be taken into consideration by the agents (Becker, Lesser, & Zilberstein, 2005) (Dutta, Goldman, & Jennings, 2007).

Clearly, there are many challenges when working in such domains. However, the essential task of any agent operating in a multi-agent system is to process the inputs it receives, and to plan how to act, in the context of other agents (Durfee, 1999). The central tasks for the agent are, therefore, (i) information processing and (ii) coordinated decision-making (including decisions about information gathering).

The first of these two tasks, information processing, is, in its fullest sense, the task of forming a coherent world view from scattered, incomplete, potentially error-prone, even conflicting messages which the agent receives at different times from heterogeneous sources. The extent to which the agent actually needs a complete world model will depend on its decision making policy. For example, if agents in a disaster situation are organised in such a way that each agent is allocated to a particular region (in the UK, this might be a county) and functions only in that region, it may choose to maintain a model only of that region and discard messages which concern other regions (National Research Council, 2005).

However, since our focus is on multi-agent systems, our primary interest is the second task: how such agents gather and then make use of their information in a multi-agent setting. Acting optimally in such settings involves the integration of two established disciplines: decision making under uncertainty, and agent coordination (Boutilier, 1996). The fusion of these disciplines results in coordinated decision making, which will be the focus of this thesis.

Before discussing this discipline in more detail in section 1.3, we will provide a focus for our research in this area, considering a domain which has all the characteristics relevant to the growing field of multi-agent research: disaster response. The importance of this domain is highlighted by recent events such as the Asian tsunami in 2004, the London bombings in July 2005 and Hurricane Katrina in 2005. Consequently, the Aladdin Project, which was the wider project providing the context for this work, focused on disaster response. Furthermore, the importance of finding effective behaviours in disaster response scenarios has led to the development of the Robocup Rescue earthquake simulation and competition. This simulation would provide inspiration for our own work and includes a testbed for coordination algorithms[2]. Thus, this challenging domain highlights the importance of online coordination algorithms within large multi-agent systems.

1.2 Disaster response as a multi-agent system

In disaster situations such as terrorist attacks, floods or earthquakes, many different teams from a number of organisations must cooperate to attempt to recover the situation. However, their work may be interrupted by self-interested actors such as journalists, scavengers, or even terrorists. Moreover, some of the cooperating organisations may have conflicting subgoals—for example, suppose during an aeroplane crash an injured person is trapped in the wreckage very close to the "black box". The police will wish to keep the black box intact for the purposes of determining what caused the crash, while ambulance teams are concerned only with removing the injured person, perhaps necessitating the destruction of the black box unless they are very careful. The overall goal of both, of course, is something loosely akin to maintaining the wellbeing of the people affected by the disaster or who might be affected by related disasters.

Scenarios of this nature provide rich grounds for the implementation of agent systems, such as the Robocup Rescue system[3], the DEFACTO system (Schurr, Marecki, Lewis, Tambe, & Scerri, 2005) and others (e.g. (Burke, 2003), (Takeuchi, Kakumoto, & Goto, 2003)). In such applications, the extent of computer intervention may be anything from a fully automatic multi-agent system, to a human-managed system receiving advice from an agent-powered device. In between these extremes, agents may be used to implement some parts of a complete system, for example managing resources such as bandwidth (Bigham, Cuthbert, Yang, Lu, & Ryan, 2004). At one end of this scale, multi-agent systems can model

[2] `http://www.aladdinproject.org/ecskernel/index.html`

[3] `http://www.rescuesystem.org/robocuprescue/`

every aspect of the disaster response, simulating the disaster, the affected humans, and the response agents. Robocup Rescue is an example of such a system. In the future, these systems could be taken further, deploying actual robots at the scene of the disaster. Indeed, there is already some work on human-robot teams (Schurr et al., 2005). At the other end of the scale, agent systems can be used alongside the human response teams, processing data and interactively suggesting courses of action (Dorais, Bonasso, Kortenkamp, Pell, & Schreckenghost, 1998). In the middle of the scale can be found agents who defer to humans in scenarios they are uncertain about (Scerri, Sycara, & Tambe, 2004).

The focus in this work is on the use of multi-agent systems for modelling aspects of a complete disaster response. We choose this perspective because it provides the broadest view of the problem. Complete solutions can be sought, and the resulting models used in more human-interactive applications. For example, software on a networked PDA can propose courses of action to be explored by the human user. Another use for such models is to aid in training human teams. For example, the Auckland urban search and rescue department are working together with Robocup Rescue developers to develop new strategic models for their own rescue services[4].

Now, taking this complete disaster response problem as an illustrative domain for exploring multi-agent systems, we identify all the properties of scalable multi-agent systems discussed above:

Decentralised: After a large disaster, it may be impossible for any one agent to have a complete view of the system. Rescue teams which find their way to a particular location will operate as a unit rather than taking precise instructions from some central authority. Different teams may be guided by different authorities, and individuals nearby may try and help without communicating centrally at all.

Dynamic: It is unreasonable to assume that a realistic system will be static. Environmental conditions are subject to constant change and agents must

[4] `http://www.auckland.ac.nz/uoa/about/news/articles/2004/06/0011.cfm`

be able to adapt to these changes. In disaster recovery scenarios agents must react to changing weather, unexpected events such as building collapse or fires and constantly moving traffic, among many other changing conditions.

Open: Disaster scenarios will have people or units moving in and out of the system constantly. In the worst case, agents are liable to die, hence vanishing suddenly. On the other hand, as volunteers and taskforces from elsewhere rush to contribute help, new agents will enter the response system.

Uncertain: As previously discussed, in disaster recovery scenarios taking place over broad areas, it is unlikely that any one agent will have a complete view of the situation. Moreover, information which reaches the agent may be error-prone, increasing the uncertainty. At a different level of granularity, environmental conditions such as the expected weather or the height of a tide can be equally uncertain.

Heterogeneous: There are many different types of agents involved in a disaster response scenario, with a variety of capabilities and (potentially conflicting) goals. At a minimum there will be the rescue teams, each with distinct tasks: ambulances, police, helicopter teams, and there will be the people affected by the disaster. Also involved may be journalists, crime teams, and environmental agencies, to name but a few.

Competitive: As discussed, the actors at a disaster situation may have conflicting goals or subgoals, as in the example above. Furthermore, self-interested agents have no reason to attempt to resolve the conflicts cooperatively.

Bandwidth-limited: One characteristic which is common in disaster scenarios is limited communication (National Research Council, 2005). For example, mobile phone networks may become jammed, rescue units from different areas or departments, may have radios set to different frequencies, and finally, the need for timely responses will limit the amount of information which can be exchanged between rescue workers or teams before they must act.

Large: Disaster recovery scenarios may involve hundreds or thousands of distinct actors, organisations or teams, operating over a wide area.

Clearly, disaster response can provide rich examples of multi-agent systems which will guide our research and inspire test scenarios. This is the domain which we will keep in mind, as we discuss approaches to coordinated decision making.

1.3 Coordinated decision making

In this section, we discuss two sub-disciplines. First, consider a single agent whose model of the world is *uncertain*: each variable within the model is associated with a probability distribution over values, rather than a single value. Given this uncertainty, how can the agent decide on an optimal action? This decision-making under uncertainty forms the first sub-discipline and is discussed in section 1.3.1. Now, consider a group of agents acting within the same environment, each of whom must make decisions about how to interact with the others. The process by which the agents make such decisions is their *coordination* protocol, and we discuss coordination protocols in section 1.3.2.

However, bringing together these two ideas, when agents are coordinating in an uncertain scenario, their coordination decisions are made in the context of their uncertainty. In fact, by including other agents as variables in the world model, the single-agent decision making processes can be used to carry out coordinated decision making in multi-agent problems. Section 1.3.3 will explain this idea in more detail.

1.3.1 Decision making under uncertainty

In a very simple model, the agent perceives the state of the world through some kind of sensory inputs, and makes a decision about how to act based on this state. Following the agent's action, the world transitions into a new state, and the agent may receive some *reward*. This model forms the basis for reinforcement learning theory (Sutton & Barto, 1998). Underlying reinforcement learning theory is the assumption that the immediate next state is dependent only on the previous state and choice of action—the *Markov assumption*. While this property does not hold for many realistic scenarios, it is a sufficiently good approximation that the learning techniques which arise from this theory often get good results, as demonstrated by many practical examples (such as (Hoar, 1996) (Smith, 2002), (Abul, Polat, & Alhajj, 2000)).

With the Markov assumption, if the transition and reward models are completely known to the agent, the system can be solved, using the recursive Bellman equations (Sutton & Barto, 1998) (described in more detail in 2.2), to determine the expected optimal action from each world state. However, when there is uncertainty about these models, the agent must integrate the learning of the models (exploration) with acting to obtain rewards (exploitation).

There are two classes of learning techniques: model-based and model-free, both described in more detail in section 2.2.2. In the former, the agent aims to learn the environmental model and then calculate an optimal action given that model. In the latter, the agent learns a direct mapping from the state to the optimal action. Model-free learning typically involves simple updates at each step and is consequently often more efficient for one-off problems. By comparison, model-based methods can be used to carry out many simulation steps alongside each real-time step, taking advantage of otherwise idle CPU cycles in relatively slow-progressing problems. Another advantage of model-based methods is the ability to bias the system towards a particular real model, using domain knowledge to guide beliefs. Given this, we focus on model-based methods particularly because of these two properties: in scenarios such as disaster response we *will* have initial beliefs about the system based on the domain or similar disasters and would like to incorporate those beliefs into our solutions.

Most model-based learning methods—such as Q-learning, TD(λ), SARSA (Sutton & Barto, 1998)—maintain a point estimate of the learned model. This estimate is used to compute the optimal action in an exploitation step. Exploration steps, in which a random action is selected, are inserted at heuristically determined intervals. By contrast, a *Bayesian* learning method will maintain a probability distribution over all possible models, in the form of a *belief state.* A set of models is sampled, and an action chosen for each sampled model. The action taken is decided from these sample actions, each weighted by the probability of the associated model (Dearden et al., 1999). Such methods provide a principled solution to the exploration-exploitation problem. In general, the more certain the agent is about its current model, the more likely it should be to take the currently optimal action rather than an exploratory action. The Bayesian reinforcement learning model pins this intuition down mathematically and so is the basis for the work in this thesis. We introduce existing Bayesian learning models in section 2.2.3. In section 3.2 we will extend these models into a general case.

1.3.2 Approaches to coordination

Above, we have discussed agents reasoning about their environments. However, as well as reasoning about their environment, agents in a multi-agent system will be interacting with each other. This interaction can be modelled by defining a (hyper)sphere of influence for each agent within the environment (figure 1.1). Overlapping spheres of influence indicate interactions between agents (Wooldridge, 2002). Moreover, a model of how different spheres interact will form a part of the agent's model of the system, as will models of the behaviour of the other agents. Given this, making decisions in the context of these other agents is the fundamental principle of coordination (Durfee, 1999). Clearly, this is a central part of a reasoning agent in a multi-agent system. Thus, in what follows we expand on how agents can reason about the behaviour of others and incorporate that reasoning into their own behaviour.

Three, potentially overlapping, coordination mechanisms are identified by (Boutilier, 1996): conventions, communication, and learning. Firstly, **conventions** are typically the simplest form of coordination. In a convention-based system, there are a number of assumed "social rules" describing ways for agents to interact when they are aware of other agents. Coordination by convention is typically simple, scalable and requires no setup time (Fitoussi & Tennenholtz, 2000). However, it is inflexible, and relies on all participants knowing the conventions and complying with them. Secondly, **communication** is used for coordination in many kinds of system. Coordination through communication has a small setup time and some bandwidth costs. In most large systems there will be some form of communication in order to share information between agents; it will be impossible for any one agent to sense all the information it needs to function effectively in context (Dutta, Dasmahapatra, Gunn, Jennings, & Moreau, 2004). However, we expect to make limited use of communication beyond information-sharing, as the bandwidth and timeliness constraints will typically preclude it. Finally, it is possible to extend single agent **learning** into the multi-agent domain. The uncertainties of our target domain make learning techniques a natural approach to problems within this domain. In this context, learning techniques are especially appealing since they enable agents to evolve coordinated policies within uncertain state spaces.

Approaches to multi-agent learning are described in more detail in section 2.3. In general, however, learning techniques may consist of a group of learners exploring the space and converging towards an equilibrium (as in (Claus & Boutilier, 1998) and (Littman, 1994)), or by one agent explicitly learning about the behaviour of others in order to adapt its own appropriately (Chalkiadakis & Boutilier, 2003). The latter, maintaining models of the other agents separately from the environment, has the benefits of model-based learning. Additionally these models need not be treated as Markovian and agent models can be reused separately from environmental models. Therefore, this paradigm of computing "best responses" (Leslie, 2004) to agents within their environment is appropriate for our domain, and will be more flexible than treating other agents implicitly. For these reasons, this will be our approach, explained within the general model in section 3.2 and evaluated in the experimental chapters 5 and 6.

Finally, we recall the requirement that our approach to coordination should be scalable. This means that as the state space increases, it is neither feasible nor necessary to model the space in precise detail. Given this, *abstractions* can enable the state space to be reduced by combining several detailed states into one higher level state. This may be achieved by partitioning or clustering the state space in some way (Sutton & Barto, 1998), or it may be achieved by mapping a high-dimensional space into a lower dimensional one (Roy & Gordon, 2002)—these approaches are discussed in more detail in section 2.4.1. In an unknown situation, such as those we are considering, suitable partitions may not be known at the outset so that an online abstraction technique is required. Statistical clustering is one such technique, assigning states to clusters probabilistically based on their binary features (Hoar, 1996) and using the clusters as the abstract states. This dovetails well with the probabilistic techniques we have highlighted above, so this is the point of departure for our work, detailed in section 3.4.2.

Now, as well as abstracting state space, in a multi-agent learning scenario we may find that the strategy space of the other agents is too large for searching effectively. This is true especially when the situation is not completely observed and an agent's policy is a mapping from a (continuous) distribution over possible states to actions. To combat this, one way of reducing the strategy space is to assume that agent strategies are restricted to being within a particular class of strategies. In particular, *finite state machines* describe one such class and have been used effectively to solve offline multi-agent learning problems (Marecki, Gupta, Varakantham, & Tambe, 2008). Thus, in our work, we will explore how the use of finite state machines can be extended to online learning problems. Section 2.4.2 provides more background on the use of finite state machines, and in section 3.4.3 we outline the algorithm which will be evaluated in chapter 6.

1.3.3 Bayesian learning for scalable coordination

Bringing together the ideas of the previous two sections, in the light of our domain requirements, we believe that "acting" and "coordinating" in uncertain systems should be completely integrated. That is, rather than use an explicit coordination layer, agents should include their beliefs about other agents' behaviour in their action selection mechanism, and adjust their own action according to their beliefs about the other agents. By doing this, agents can make efficient decisions about coordinated actions. Moreover, we believe that this integrated approach should be based on sound theoretical principles and not heuristic techniques. Thus we motivate the use of multi-agent learning models, since these provide a basis for such coordinated action selection and are designed for uncertain domains. We therefore explore the application of multi-agent learning models to dynamic, partially observable domains.

In particular, as a point of departure we consider the Bayesian learning model of (Chalkiadakis & Boutilier, 2003) in which agents maintain probability distributions over models. This has been proven to be effective on small test problems, including some problems not handled well by previous multi-agent learning mechanisms. However, this model is only defined for the fully observable case: agents can see the actions of all the other agents, and ascertain deterministically what the current state is. Furthermore, it has not yet been tested on large domains; the sample problems have two agents and half a dozen states. In our work, we address these shortcomings. The next section outlines our research contributions in more detail.

1.4 Research contributions

In this thesis we extend the state of the art as follows:

- We describe a principled Bayesian model for coordinated decision making in partially observable systems. This model generalises the models of (Chalkiadakis & Boutilier, 2003), (Ross, Chaib-draa, & Pineau, 2008) and (Emery-Montemerlo, Gordon, Schneider, & Thrun, 2004) and is the first explicit formalisation for learning over all partially observable multi-agent systems. We then implement three special cases of this general model:

- Firstly, we demonstrate that explicitly modelling the other agents' behaviour results in effective learning. This algorithm is the first to consider explicit models of the other agents in a partially observable Bayesian learning environment, and we show that it is better than the existing techniques, which treat the other agents as a part of the environment.
- Secondly, we extend the model with finite state machines for policy abstraction, developing a new efficient online decision process for partially observable multi-agent scenarios. This algorithm is the first principled online approach which is able to generalise up to millions of states and tens of agents, and it outperforms a solution hand-designed for our disaster response problem.
- Finally, we show that our model-based solution is effective in scenarios with changes in the world—agents leaving and entering (open domains) and changes in the environment (dynamic domains), the first online learning solution to explicitly consider these cases.

The combination of these contributions is a model for coordinated decision making in rich and challenging domains, with high levels of uncertainty. This model is based on a well-founded approach, giving us confidence in its correctness and a set of guarantees about its behaviour in small systems. The model is extended into larger-scale systems using abstraction techniques, demonstrating its practical effectiveness. The system is intended to be very flexible in its applicability, guiding all or part of agent behaviour in both cooperative and competitive systems. In undertaking this work the following publications have been made to date:

1. Allen-Williams, M. and Jennings, N. R. (Forthcoming in 2009). Bayesian learning for cooperation in multi-agent systems. In C. L. Mumford and L. C. Jain, editors, *Studies in Computational Intelligence: Collaboration, Fusion and Emergence.* Springer-Verlag, London, UK. (Allen-Williams & Jennings, Forthcoming in 2009b)

2. Allen-Williams, M. and Jennings, N. R. (Forthcoming in 2009). Bayesian adaptation for complex dynamic systems. In M. Wang and Z. Sun, editors, *Handbook of Research on Complex Dynamic Process Management: Techniques for Adaptability in Turbulent Environments*. IGI Global. (Allen-Williams & Jennings, Forthcoming in 2009a)

1.5 Thesis structure

In this thesis we expand on each of the above contributions in turn:

- In chapter 2 we examine in detail the background to our research, highlighting our key decisions and how they arise from the state of the art.
- Chapter 3 extends existing Bayesian learning systems into the general case and demonstrates the use of Bayesian network diagrams as a visual aid in understanding specific cases of the model. In order to make the model tractable for large problems, we propose the use of state abstractions using clustering, and policy abstractions using finite state machines.
- In chapter 4 we instantiate the above models on a specific problem from the disaster response domain, motivated by Robocup Rescue. Using this instantiation, the ensuing chapters evaluate the model of chapter 3 for the three cases highlighted above:
- In chapter 5, we implement the special case where just actions are partially observable, comparing it with state of the art multi-agent learning algorithms.
- In chapter 6, we implement the special case where just states are partially observable, evaluating the properties of our model over several scaling parameters and comparing the model with a handwritten policy for the same scenario.

- Finally, in chapter 7 we extend the above case to scenarios in which the environment or the available agents may change during the online solution.

- Chapter 8 concludes the thesis, outlining ways in which the model could be extended to address more of the domain requirements in section 1.2 and suggesting a number of directions for future study.

Chapter 2

Literature Review

This chapter introduces the background to the work contained in this thesis, explaining the way in which the multi-agent approach to partially observable systems is developed from single-agent decision theory and justifying the decisions we have made at each step in building on the state of the art. We begin, in section 2.1, by introducing the conceptual underpinnings: the agents and environments we use. Section 2.2 will then outline the theory about agent decision making in a particular kind of environment, the Markov decision process, and will explain how these processes are relevant to our research. Then, in section 2.3 we look at extending agent decision making processes in multi-agent scenarios. Finally, section 2.4 describes some approaches to scaling up multi-agent systems.

2.1 Autonomous agents

In this thesis, we assume that an agent is an entity which is situated in some environment and reacts to that environment, in order to try and achieve some objective or goal (this definition is based on (Wooldridge, 2002), chapter 1). Such agents will be able to reason logically about their actions and the effects of the actions on the current state of the world, relating this to their goal. At times, inconsistencies and conflicts in agent goals and beliefs may crop up; the agent's reasoning mechanisms must have some means of resolving these. We believe that probabilistic methods provide a realistic way to do this for two reasons. First, such techniques are effective for reasoning in uncertain scenarios where an agent may want to reason using its belief in a particular property (Mackay, 2003). Second, probabilistic representations are typically more compact than their logic-based counterparts for both the input data and the agent models (Toni & Bentahar, 2008) (Stenning & Lambalgen, 2005).

Given such a reasoning mechanism, this mechanism must consider both the signal the agent receives from its environment, and the effects of its own actions. When reasoning, the agent may maintain an explicit world model or it may leave the model implicit as it reasons about plans. In this setting, explicit models have more potential for reasoning about states and behaviours, as they store more information explicitly. However, maintaining explicit models may be computationally and memory intensive (Excelente-Toledo & Jennings, 2005). Despite this, we believe that the aforementioned benefits of explicit models justify their use where practical. Now, if, as in many disaster arenas, the world is large and detailed, agents may only be able to create such explicit models for small parts of the world, due to memory constraints. In such worlds, it is therefore appropriate either to use a model which can store information at different levels of detail, or to reason in a simplified abstract world. In section 2.4 we discuss some possibilities for achieving each of these.

Finally, as well as reasoning about their environment, agents in a multi-agent system will interact with each other, as discussed in section 1.3.2. We will therefore show how to extend models for reasoning under uncertainty to incorporate explicit considerations of the other agents. Thus, in the following sections we will expand in detail on the dual aspects of making reasoned decisions under uncertainty, and making coordinated decisions in uncertain scenarios. This discussion is guided by the motivating domain of disaster response, as introduced in section 1.2. To help maintain this focus and illustrate our key ideas, example 1 describes an earthquake

Example 1 *Tamptono earthquake scenario*

*An earthquake has damaged the small town of Tamptono (figure 2.1), including its hospital and ambulance fleet. Buildings are still collapsing and there may be aftershocks (*dynamism*). Ambulance teams from nearby towns converge on Tamptono, travelling through the damaged streets searching for hurt victims among the rubble (*partial observability*). Although the ambulance dispatch stations are able to communicate with one another, once the ambulances are on the road to Tamptono, they find the communications networks are blocked (*bandwidth limitations*). They must therefore make decisions independently (*decentralisation*), leaning out of their windows to warn other ambulance drivers about damaged roads, exploring parts of town not yet marked by emergency services' red and white tape, or going to the aid of ambulance teams working in particularly damaged areas, such as a collapsed office building. They also need to learn about the capabilities of ambulances from other towns, who may be equipped differently or even have different goals—one apparent ambulance turns out to be a concealed news team. As the ambulances come and go, the system is* open*.*

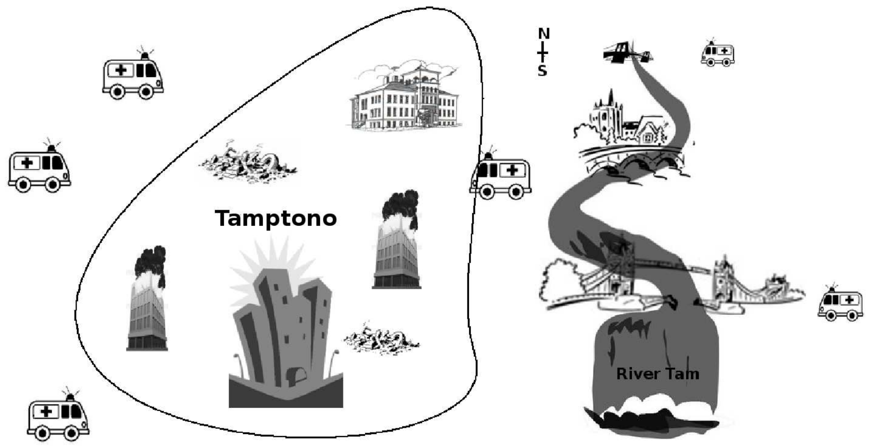


FIGURE 2.1: Tamptono, after an earthquake

scenario having many of these features which we will use as a running example in this chapter[1].

2.2 Markov decision processes

The most straightforward of this class of dynamic problems is the single agent observable Markov decision process (MDP). The MDP forms the theoretical foundation for reinforcement learning problems—when the environmental dynamics

[1] Tamptono is a fictional place. No real towns were harmed for this thesis.

are unknown—and partially observable problems, both of which we will we go on to discuss.

In an MDP (figure 2.2), the agent perceives the state of the world s through its sensory inputs, and decides on its immediate action a based on this state. Following the agent's action, the world *transitions* into a new state s', and the agent may receive some *reward* r. A key feature of such problems is *delayed reward*: states which have no or negative reward, but which ultimately lead to higher rewards (Sutton & Barto, 1998). This is a common feature of many real life problems—in the Tamptono earthquake, ambulances may use up valuable fuel for no immediate gain, and rescuers risk being hurt themselves, before the final goal of a rescue is achieved.

Determining the reward achieved from a particular action may be straightforward—a rescue worker retrieving valuables from a building may receive a fixed reward for each valuable he retrieves, or an ambulance team may receive a fixed reward for each person loaded into an ambulance alive. However, in some kinds of problem deciding a reward function may be trickier. For example, following an earthquake, buildings may be burning while humans are buried and trapped. A reward function for a team simultaneously rescuing humans and extinguishing buildings may try and put relative values on the buildings and the human lives, supplying some reward for unburnt buildings and some for live humans. An alternative reward function might try and assign higher value to some humans—for example, the prime minister, or people who can be immediately useful to the rescue. We do not discuss this issue further, but simply suppose that a reward function exists, supplied by the environment.

These intuitions are pinned down by defining a finite set of states S, a finite set of actions A, and a finite set of rewards R. The environment dynamics are then defined by (Sutton & Barto, 1998):

- A transition probability function, $T_f = P(s'|s,a)$. This defines the probability of reaching state s' from state s given that the action performed was a.

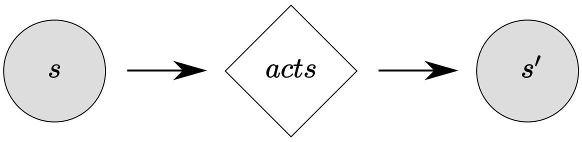


Figure 2.2: Markov decision process progression

- A reward probability function $R_f = P(r|s, a, s')$. This defines the reward achieved by taking action a from state s, resulting in state s'.

The agent makes decisions according to a policy π, where $\pi(s, a)$ defines the probability the agent will take action a from state s.

In this model, the probability of transitioning to a particular state, or achieving a particular reward, does not depend on any of the history of states and actions. This *Markov property* is the fundamental feature of Markov decision theory (Sutton & Barto, 1998). As discussed in section 1.3.1, this property rarely holds in reality, but is often an effective approximation.

Given this context, the goal for the agent is to maximise some function based on the reward obtained. This may be (Sutton & Barto, 1998):

- over a fixed time horizon
- during an *episode* in which the agent continues to act until some termination condition is reached
- the average reward over an indefinite time period, or
- the total reward over some time period

In the last case, more recent rewards may be valued more highly than earlier rewards—in particular, this encourages adaptation to *nonstationary* environments, in which the environmental models are changing over time. In disaster scenarios, we may consider either the total reward accrued (perhaps in number of lives saved) when some termination condition is reached (the scene is cleared up), or we may consider how efficiently our agents can act to accrue reward over a fixed time period. In our experiments we will be looking to see how quickly agents are able to start accruing the high rewards (since in disaster scenarios, there is no leeway for a long learning phase), but for ease of implementation we will assume continuous running of the scenario rather than enforcing a time cutoff. Specifically, in choosing an action at time $t = t_T$, the agent's aim is to optimise the expected discounted future rewards, defined by:

$$R_T^\gamma = \sum_t \gamma^t r_t \quad (t \text{ ranges from } T \text{ to } \infty) \tag{2.1}$$

where $r_t \in R$ is the reward at time t. γ is a problem specific parameter which defines the agent's *myopia*; that is, to what extent it considers delayed future rewards to be important. It balances the importance we place on future states with our need to accumulate reward now. In practical terms it will be chosen to express the extent of lookahead appropriate to the problem (consider chess as an analogy: for the most part, say, 3 steps of lookahead are sufficient to play adequately (although more may be required during the endgame) (Sadikov & Bratko, 2006)). Typically, we will use a γ value of around 0.8, making lookahead negligible after around ten steps into the future—in a fragile disaster scenario we expect this to be sufficient for most planning purposes, as the agents will have to adjust their plans to a changing situation within a few steps in any case. It is most common for reinforcement learning algorithms to set γ between 0.7 and 1, although the choice will depend on the exact problem (Sutton & Barto, 1998).

Within a Markov decision process, if the transition and reward probability functions are known, then it is possible to derive the policy which optimises this reward function, by solving the large simultaneous equations known as the Bellman Equations (2.2 and 2.3) for V^* (2.4). That is, $V_\pi(s)$ defines the long term value of state s to an agent following policy π, and V_* optimises this value for every state:

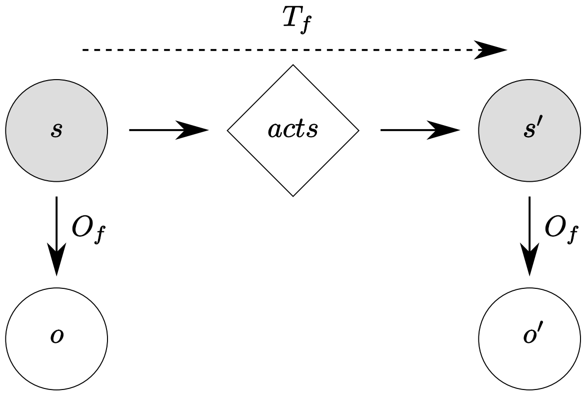


FIGURE 2.3: Partially observable Markov decision process

$$\begin{aligned} Q_\pi(s,a) &= \sum_{s'} P(s'|s,a)[r(s') + \gamma V_\pi(s')] && (2.2) \\ V_\pi(s) &= \sum_{a} P(a|\pi) Q_\pi(s,a) && (2.3) \\ V^*(s) &= \max_\pi V_\pi(s) \text{ for all } s \in S && (2.4) \end{aligned}$$

where $V_\pi(s)$ denotes the value to the agent of being in state s, given both the immediate reward and the discounted future rewards it can achieve from that state if it continues with policy π. $Q_\pi(s,a)$ denotes the value to the agent of being in state s and taking action a, given the immediate reward and the expected value V_π of the resulting state.

There are various ways of efficiently approximating these solutions in large problems, and for solving in continuous systems. Briefly, the equations can be solved iteratively, and efficiency is achieved by (a) updating the states most likely to have changed first, and (b) updating "nearby" states when a state is updated (Sutton & Barto, 1998). We do not go into the details of these solution techniques as realistically we are unlikely to know all the necessary parameters. Rather, we go on to explain how this model is extended into systems with unknowns.

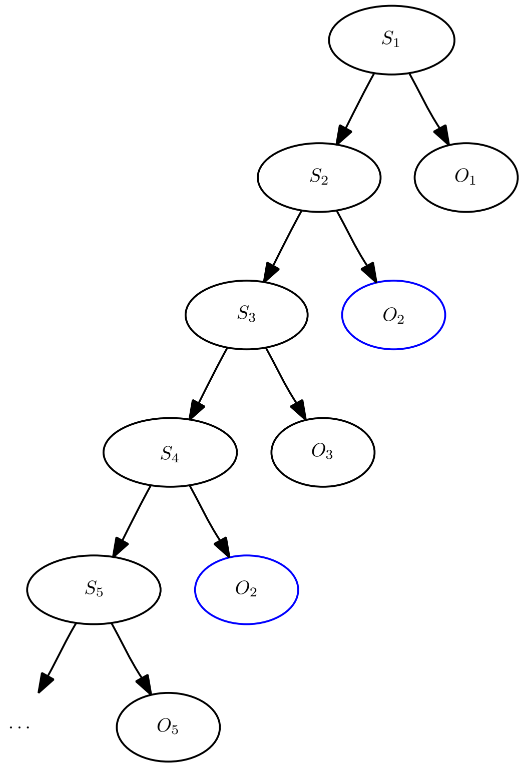


FIGURE 2.4: Partially observable Markov decision process

2.2.1 Partially observable Markov decision processes

Although MDP models will form the basis of our environment, in large or complex scenarios it is common for an agent to make local observations which allow it to form inferences about the current state (example 2), without observing the complete state directly (although in multi-agent systems, local observations may be augmented with communicated information). When the underlying process of moving from global state to global state is still (assumed to be) Markov, the scenario is described as a *partially observable Markov decision process*, or POMDP. Specifically, we assume the existence of a fixed, known model $O_f = P(s|o)$ where o is the current set of observations and s is a possible state (figure 2.3). Although the sequence of states is defined by an MDP, the sequence of observations is not. Consider the simple example in figure 2.4: if the current observation is O_2, then the probability that the next observation will be O_5 differs depending on whether the previous observation was O_3 or O_1.

Example 2 *Partial observability in the Tamptono earthquake*

Two Tamptono ambulances which survived the earthquake immediately swing into action. However, beyond the strength that they felt the earthquake to be, they have no idea of the scale or the detail of the situation. Elsewhere, as an office worker runs from a crumbling building, an approaching ambulance calls out to ask how many people were in the building—the answer (an estimate) is information which will remain local to that ambulance until much later. In other parts of town, other ambulances will have their own local information. However, the big picture will not be completed until much later on, if at all.

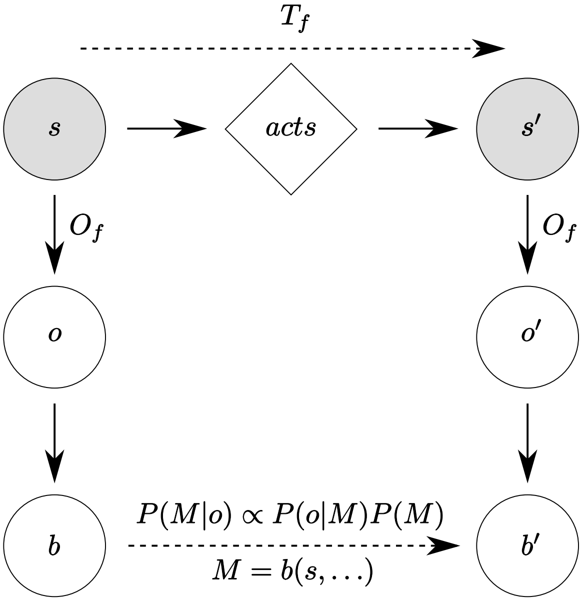


FIGURE 2.5: POMDP inducing a Bayesian belief state MDP

To solve a POMDP, we can derive from it a secondary MDP—a *belief MDP* (figure 2.5). The multi-dimensional states of this secondary MDP have one continuous variable, $b(s)$, for every possible value s of the underlying state. The value of $b(s)$ indicates the agent's belief that the underlying state is s, given the agent's prior knowledge and the history of observations and actions. The system proceeds from b to b' at each step using Bayes' rule (equation 2.5) to update the state probabilities:

$$P(x|observations) \propto P(observations|x)P(x) \tag{2.5}$$

This belief MDP is, therefore, completely known, and can be solved exactly by exploiting its properties—the value functions are convex and piecewise linear

(Kaelbling et al., 1998). Intuitively, there is a “piece” of linear value function for the policy tree arising from each possible state, and the value function for the state is the upper surface of all these segments. The witness algorithm (Kaelbling et al., 1998) is based around this notion, but does not scale to large problems. Incremental pruning (Cassandra, Littman, & Zhang, 1997) addresses some of the efficiency problems with the witness algorithm, but generally exact solution methods do not scale well and are thus not appropriate for real-world systems of the kind we are trying to address.

A more scalable approach to solving such continuous MDPs is to compute approximate value functions for belief states, exploiting the intuition that a large part of the belief space need never be visited. Techniques include point-based sampling (Izadi & Precup, 2006) (Virin, Shani, Shimony, & Brafman, 2007), dynamic programming with sampling (Atkeson & Stephens, 2008) and myopic evaluation (only looking ahead at the values of the next one or two states) (Chalkiadakis & Boutilier, 2003). A more recent offline method uses quadratically constrained linear programs to describe locally optimal policies (Amato, Bernstein, & Zilberstein, 2006), with promising results. Another novel and interesting technique is the use of principal components analysis (PCA)[2] to map the belief space into a low dimensional space, carrying out the planning in this low dimensional space (Roy & Gordon, 2002).

An alternative to the above belief-state approaches is to calculate a policy directly from the observation history, or a subset of the observation history, thus arriving at an approximate solution. The key to success with such a technique is to make a good choice of observation history, within the constraints of the memory and computation power available to the system. For example, the agent can try and detect ambiguous elements of the history by considering the possible future states which arise from certain history subsets (Dutech, 2000).

Another popular approach is to construct agent policies as *finite state machines* (sometimes called *finite state automata* or *regular automata*, and abbreviated to

[2] For an explanation of PCA, see for example (Bishop, 2004), chapter 8

FSMs). Finite state machines have a set of internal states, or nodes, and actions associated with each node. Movement from node to node is determined by the agent's observations. Using this class of policies, it has been shown that it is possible to compactly represent good approximations to the optimal agent policy (Carmel & Markovitch, 1996) (Clark & Thollard, 2004). We revisit finite state machines in section 2.4.2, however, our interest is in more explicitly model-based agents. For example, while finite state machines can provide an effective way to control a single agent, an agent which is calculating its actions from a learned model can respond more quickly when the model changes.

For our work, we propose to use a combination of solution techniques; sampling and myopic evaluation, both applicable in online solutions, since high dimensional state spaces will result in high dimensional belief spaces, necessitating several approximation techniques to become tractable. Section 3.4.4 outlines these techniques. In future work, we may investigate the combination of the PCA technique with other approximation techniques, as it provides an elegant way to reduce the state space while retaining the most important information (section 8.3.1).

Now, so far we have discussed scenarios in which the environmental dynamics (T_f, R_f and in the case of POMDPs O_f) are known. However, in many scenarios, these dynamics may be only partially known to the agent. In such cases, the agent must learn about the scenario online. The next section discusses learning techniques for MDPs and POMDPs.

2.2.2 Reinforcement learning

When there is uncertainty about the aforementioned models (T_f and R_f), as in example 3, the agent can *learn* the optimal actions through experimentation. As discussed in chapter 1, techniques for learning may be model-free, or model-based. In model-free learning techniques for Markov decision processes, such as Q-learning, TD(λ) and SARSA (Sutton & Barto, 1998), the agent stores a mapping

Example 3 *Rescue worker in Tamptono old people's home*

A lone rescue worker searches Tamptono's old people's home after the earthquake. As she works her way up the building, she takes increasing care how she treads, not knowing what structural damage the earthquake may have caused—the environmental dynamics are uncertain. *Some parts of the building were more heavily populated than others—the dining area was full of both elderly residents and waiters; most of the bedroom wings are almost empty, but the Violet Wing was being cleaned by a team of a dozen cleaners. The rescue worker does not initially know how the building was laid out or which areas were most crowded, and must discover this as she makes her way through the building.*

from each state to the optimal action for that state, updating this mapping based on experience. In a model-based learning method, the agent's experience is used to update the agent's estimates of T_f and R_f, using the Bellman equations to derive the optimal action at each point (Sutton & Barto, 1998). For each type of learning, agents must find a balance between taking what they believe to be the optimal action, and taking exploratory actions to refine their estimates.

We argued in section 1.3.1 that model-based methods can be more powerful. Specifically, in many applications, agents may have spare CPU cycles during a timestep; for example while waiting for environmental input, while carrying out a motor action, or if a timestep corresponds to some fixed unit of real time. In such applications, agents that maintain a model of the environment may use these spare cycles to simulate actions based on their model, and refine their policies accordingly. Providing the models are sufficiently accurate, this can result in much faster convergence to the optimal policy (Sutton & Barto, 1998).

Furthermore, model-based algorithms permit the use of a prior model to guide agent learning—although doing so can be a disadvantage if unwanted bias is introduced (Dearden et al., 1999) (Sutton & Barto, 1998). In a disaster scenario problem, such a prior model may be advantageous because agents can enter the scenario with some initial model based on previous knowledge of the area and previous disaster experiences, and then learn from the current experience to refine this model and hence their behaviour.

Finally, model-based algorithms may use the Bellman equations to derive the optimal behaviour for the estimated model, or the iterative techniques derived from these equations (Sutton & Barto, 1998). However, deciding behaviour based on a point estimate of the model ignores a key variant: the agent's uncertainty about its estimate. The uncertainty in the estimate should affect both the caution with which the agent behaves, and the decisions it makes about trading exploratory actions (investigating unknown regions of the environment) with exploitative actions (those which it believes will accrue high reward) (example 4). We refer back to example 3 as a demonstration of each of these points. Firstly, if the rescuer is unsure about her estimated model of her current region, she must step forward cautiously so as to jump back if a board falls away underfoot. Secondly, if she has found an occupied wing, but has left two wings unexplored, it may be that one of those wings contains many more people or more disabled people than the current one. More recently, *Bayesian* model-based techniques have been developed which explicitly include these uncertainties in the agent model.

2.2.3 Bayesian reinforcement learning

With Bayesian learning techniques, an agent stores a probability distribution over all possible models, in the form of a *belief state* (Dearden et al., 1999). The underlying (unknown) MDP thus induces a *belief-state MDP*. The transition function from belief state to belief state is defined by Bayes' rule, with the observations being the state and reward signals arising from each environmental transition.

Now, these MDPs do not have a finite state space, so cannot be solved using MDP solution techniques for finite state spaces. Specifically, the expected value of a particular action is given by:

$$E[Q(s,a)] = \int_M Q(s,a|M)P(M)$$

where M denotes a possible model ($M = (T_f, R_f)$), and $Q(s,a|M)$ is the Q-value given that particular model. There are corresponding continuous versions

Example 4 *Exploration-exploitation in the Tamptono earthquake*

The river Tam runs to the East of Tamptono. As ambulances rush in to the rescue from the East, they find that the earthquake has also destroyed several of the bridges across the river. An ambulance arriving at the riverside early after the disaster is able to learn over the radio that there is a bridge still standing two miles downriver. However, there is no data about the bridges upriver. The ambulance driver knows that there is a bridge only half a mile away, if it is still standing, and another a mile and half away, but then there are no more bridges for five miles. The decision the ambulance driver must make about whether to travel in the uncertain direction, or head straight for the bridge which is known to be standing, is an example of an exploration-exploitation *problem.*

of the Bellman equations. To solve these, it is necessary to use some means of approximating a solution. For example in (Dearden et al., 1999), sampling techniques are used, using a finite number of candidate MDPs at each step when estimating the optimal action given the current state.

Now, for an agent maintaining a probability distribution over models in this way, the transition from probability distribution to probability distribution defines a continuous Markov decision process. Each state in this process is a probability distribution over states in the world; such states are described as *belief states*. The transitions between belief states are determined by Bayes' rule.

Subject to the aforementioned approximations, the Bayesian model is a well-founded approach to decision making within single-agent problems in which the state of the world is known at all times, but the environmental dynamics and, in particular, the effects of agent actions are uncertain. Consequently, several forms of Bayesian reinforcement learning have received recent attention. For example, one approach uses linear programming with sampling (Castro & Precup, 2007) to maximise the current Q-value. Another recent approach uses Gaussian processes (Rasmussen & Williams, 2006) to maintain the value function in online learning (Reisinger, Stone, & Miikkulainen, 2008).

However, the scope of our illustrative domain is broader than the fully observable world investigated by these techniques: in particular, we expect that frequently agents will not be able to observe the full state of the world but will be acting within POMDPs.

As in the fully observable case, learning techniques for POMDPs may be model-based or model-free. For example, (Baxter & Bartlett, 2000) propose an approach for learning a policy directly, using gradients to incorporate a performance measure into the learning. Several similar techniques are described in (Aberdeen & Baxter, 2002), in which the agent uses Monte-Carlo methods to learn through interaction with the environment.

By contrast to such model-free methods, model-based methods are developed around the insight that a POMDP is a form of hidden Markov model (HMM), in which a sequence of states gives rise to a sequence of observations. POMDPs have the added complication over HMMs of including actions, but HMM solution techniques such as the recursive Baum-Welch equations (Roweis, 2003) can be adapted ((Nikovski & Nourbakhsh, 1999), (Chrisman, 1992)). However, this solution form requires a complete dataset (sequence of observations) and so is not appropriate to incremental learning. In realistic problems, such as those we are addressing, an agent must develop and update its policy as it explores the environment, so some form of online learning is necessary. An alternative to the use of Baum-Welch updates is the use of short-term memory trees to provide model updates (Shani, Brafman, & Shimony, 2005). Such trees contain variable-length sequences of observations, in order to handle the non-Markov properties of POMDPs. This approach can be integrated with an incrementally improving policy.

To sum up, all of these techniques rely on a number of approximations and assumptions about the state and hence are not entirely satisfactory. We propose, as an alternative, to extend the Bayesian model of section 2.2.3 into this POMDP domain. This formulation falls naturally into the belief-state space of POMDPs, and will provide the advantages of the Bayesian model-based methods (explicitly handling uncertainty and making use of prior knowledge) in this domain. In section 3.2 we give the details of this model. However, in many examples of large and partially observable problems, the learning agent is not acting alone. We must therefore explore the generalisation of the above approaches into multi-agent systems. This is the focus of the remainder of this chapter.

Example 5 *Ambulance traffic at the Tamptono earthquake*

Consider again the ambulance driver arriving at the River Tam after the Tamptono earthquake. If he is the only ambulance approaching the scene, he may choose not to take the risk of having to travel many miles upriver, and head straight for the bridge which is known to be standing. However, if he knows that there is a fleet of ambulances following him, he may choose to head upriver so that he can (subject to communication networks functioning) send back data about the status of the bridges to later ambulances, enabling them to update their model without the travel costs. He might also consider that if all the ambulances were to head for the single bridge, a traffic jam would form there, perhaps wasting precious time.

2.3 Multi-agent learning

Clearly, when several agents are functioning within a system, the interactions between their behaviour are relevant to the decisions they make. Example 5 illustrates this, extending the example of the previous section (4) into the multi-agent domain. In section 1.3.2 we described some common approaches to coordinating these interactions and motivated the use of multi-agent learning.

There are two main approaches to the extension of single-agent learning into such multi-agent systems. The first approach, generally applied to cooperative systems, is to consider the problem as a whole, with the ultimate aim of finding optimal joint actions, although the implementation may be decentralised with each agent learning separately. This is the typical focus of work described as “multi-agent reinforcement learning” (MARL) (Panait & Luke, 2005). By contrast, our work will focus on the way in which a single agent (or team of agents) operates, in the context of other agents. In disaster scenarios, there may be an assortment of agents, each of whose behaviour is determined by its own controlling algorithm (and not necessarily rational, as demonstrated in example 6). Furthermore, these agents may not all be cooperative. Therefore, solutions which rely on all agents behaving the same way and having the same goal are not appropriate to this kind of problem.

Instead, an approach more appropriate to our domain, usually described as “learning in games” (Fudenberg & Levine, 1998), arises from the addition of learning methods to game theory. When agents treat a multi-agent problem as a

Example 6 *Heterogeneous agents in Tamptono*

A team of rescue workers join the lone rescuer in the old people's home. Furthermore, despite all advice to the contrary and their lack of safety training, two of the old folk have also joined in. Alongside the rescue team, two policemen carefully traverse the building looking out for looting kids. In this example, we have a multi-agent problem in which several types of agent must interact and take each other into account—for example, negotiating the passage through doorways as they meet, or avoiding putting excess weight on damaged floorboards. Some of the agents are aiming to cooperate; the rescue workers spread, each searching a different region of the building. The policemen have different goals and do not contribute to the rescuers search. Not all of the agents behave predictably or rationally; one of the old folk gets confused at times.

Example 7 *Nash equilibria in the Tamptono earthquake*

Ambulances from two different hospitals are approaching a victim trapped in an unstable tunnel. If one of the ambulances attempts the rescue alone, then they risk the tunnel collapsing: rocks fall, everyone dies. If both ambulances contribute to the rescue then one ambulance can maintain stability in the tunnel while the other completes the rescue. The table below shows the payoff matrix, with the table entries being the payoffs for (Ambulance 1, Ambulance 2). The penalty for leaving for each ambulance is the cost of its wasted fuel. The Nash equilibria are at (Leave, Leave) and (Stay, Stay), with (Stay, Stay) an optimal equilibrium.

		Ambulance 1	
		Leave	*Stay to help rescue*
Ambulance 2	*Leave*	*(-5,-9)*	*(-100, -9)*
	Stay to help rescue	*(-5, -100)*	*(50,50)*

stochastic *game*, solving the problem revolves around finding a *best response* to the other players of the game; that is, finding the action which gives the single agent the best reward it can achieve, given the actions chosen by the others. When the action of every agent is a best response to the others, then the system is said to be at a Nash equilibrium (NE) (Fudenberg & Levine, 1998). There may be more than one NE in a system, and some equilibria may be better than others (example 7).

In game-theoretic formulations, if all the players iteratively keep playing best responses and if their strategies are *mixed* (stochastic), then the play will converge to a (mixed) equilibrium, in which every player's strategy is a best response to every other player—this is a cooperative problem solving approach. One of the challenges is then to direct the play so that convergence is not just to any equilibrium but to an optimal one (Claus & Boutilier, 1998). Alternatives to the simple best response approach include the *Bully* strategy which selects an action which minimises the opponents' best response, and the *tit-for-tat* strategy. Such strategies can sometimes converge to better equilibria than the traditional best response strategy (Littman & Stone, 2001) (Powers & Shoham, 2005).

However, in large and complex systems, a NE may not even be a useful target—finding an equilibrium may be too costly, for example. Furthermore, finding a NE is only possible if all the players are adjusting their strategies towards this goal. In heterogeneous settings, such as the disaster response domain, we cannot ignore the possibility that some agents may have, say, naïve fixed controllers and all we can do is make our behaviour a best response to these (Powers, Shoham, & Vu, 2007).

More realistic for our domain, therefore, will be to find a multi-agent learning algorithm capable of finding a *satisfactory* solution which has desirable properties over an indefinite period of play (Lesser, 1999). Such properties may include: convergence (of rewards, of actions, of strategies), rationality, and no-regret—this latter property means that a learning algorithm should not allow itself to be exploited by malicious opponents (Bowling, 2005). Which of these are more relevant is problem-specific. For example, the no-regret property can be ignored if all the agents are known to be cooperative, while convergence is less important to a constantly changing scenario. For our dynamic domains, convergence is not a key property since there is no guarantee of anything to converge to. (Other criteria have been proposed targetting small repeated games, such as safety (similar to no-regret) and compatibility (the agent plays well against itself) (Powers & Shoham, 2005). These are of less interest to us in our large, primarily cooperative, settings)

Given this, one effective approach to extending single-agent reinforcement learning into this game theoretic setting is the *win-or-learn-fast* (WoLF) approach: an agent's learning rate is adjusted according to its current performance, without explicitly modelling the other agents (Bowling & Veloso, 2001). A WoLF variant which uses gradients to control the learning rate achieves no-regret (Bowling, 2005). Like their single-agent counterparts, WoLF techniques can be improved upon by using a Bayesian model in which agents maintain beliefs about the behaviour of the other agents, separately from the world models (Tesauro, 2004) (Chalkiadakis & Boutilier, 2003) (Burkov & Chaib-draa, 2007). The need for heuristically determined learning rates is then eliminated, while prior information about agents can be incorporated.

In more detail, in the multi-agent Q-learning systems of (Tesauro, 2004) and (Burkov & Chaib-draa, 2007), explicit history-based models of the other agents are maintained, and incorporated into the Q-update at each step. (Chalkiadakis & Boutilier, 2003) is the model-based version of these, extending the Bayesian learning technique discussed in section 2.2.3. In this multi-agent model, the agent's belief state, b, now contains: a probability distribution over the environmental dynamics, a probability distribution over strategies, the most recent state and action choices, and a history trail which contains as much state as is necessary to accurately model the other agents' strategies[3]. At each timestep, the agent (supposing it to be agent i) selects the action a_i which leads to the greatest expected value, given the above beliefs. In this model, the expected value is computed by a Bellman equation over the belief state (equation 2.6).

$$Q(a_i, b) = \sum_{\mathbf{a_{-i}}} P(\mathbf{a_{-i}}|b) \sum_{s'} P(s'|a_i \circ \mathbf{a_{-i}}, b) \sum_{r} P(r|s', a_i \circ \mathbf{a_{-i}}, b)[r + \gamma V(b < s, \mathbf{a}, r, s' >)] \tag{2.6}$$

where

$$V(b) = \max_{a_i} Q(a_i, b)$$

and $b < s, \mathbf{a}, r, s' >$ is the updated belief state which arises from being in the belief state b, in which the current state is s, and carrying out joint action $\mathbf{a}$, resulting in new state s' and reward r. The environmental and strategy models are updated using Bayes' rule.

[3] Determining the relevant history assumes some knowledge about the other agent strategies. If the agent maintains insufficient history, then its model of opponent strategies will never be accurate and thus the results will be suboptimal. In the case where the underlying process is Markov and where all the agents are learners trying to converge to an equilibrium, we may safely assume that the previous state contains sufficient history to obtain an accurate model.

Example 8 *Multi-agency ambulance rescue in Tamptono*

In example 4, we saw an ambulance driver receiving information about the status of the each bridge on the Tam as the bridge was visited. However, suppose that two fleets of ambulances are approaching Tamptono—the regular fleet from the South and West, and a fleet of army ambulances from the North. The army ambulances use a different radio frequency from the Tamptono regional fleet. Information between the two fleets is therefore no longer shared, except by word of mouth as ambulances pass one another, resulting in ambulances from different fleets having different models of the scene.

Now, this model assumes full observability, an assumption which we have already disputed for the single-agent case, and which becomes increasingly improbable as the problems get larger. In fact, in multi-agent problems agents will usually have local observations which are not available to any other agents and can only be partially shared by communication. Therefore, we must extend these fully observable solutions with techniques from the POMDP solutions previously discussed. This is the focus of the next section.

2.3.1 Partially observable stochastic games

Example 8 describes a scenario in which agents may make local observations which are not known to the other agents. Even if all the agents begin with the same prior knowledge, each agent in such a scenario will have a different estimate of the state, or a different probability distribution over states.

Formally, the extension of the model-based approaches into such multi-agent POMDPs, or *partially observable stochastic games* (POSGs) (Emery-Montemerlo et al., 2004), consists of: a finite set of agents I, a finite set of states S, a finite set of actions A, a finite set of observations O, an initial state distribution and a set of Markovian transition probabilities $P(s', \mathbf{o}|s, \mathbf{a})$, and a set of reward functions, $R_i : S \times A \rightarrow R$. If the dynamics of the system are known, then in principle the POSG can be solved to determine the optimal action for a particular agent given a set of beliefs about the other agents (Gmytrasiewicz & Doshi, 2005). However, we have already seen that there is considerable complexity in solving POMDPs; adapting these solutions to the multi-agent environment is a challenging problem.

Example 9 *Beliefs about the beliefs of others, in the Tamptono disaster*

A very sick victim has been found at one of the disaster sites. This victim can only be rescued from the site if she is carried out on a stretcher, thus requiring two rescue agents. She is likely to die within minutes if no rescue is effected. Ambulanceman Archie has found this victim and is wondering whether to wait with her for another ambulance, or to give her up as a lost cause and search the nearby site. Archie can see another ambulance, Bob, in the distance, but does not know whether Bob is on his radio frequency, or not. Thus, Archie does not know whether Bob has heard his urgent call and might come over to the victim in time, or whether Bob believes that this site is clear.

In particular, whereas a POMDP is solved by conversion to a belief-state MDP and then solving the resulting continuous MDP, a POSG is complicated by the need to include beliefs over the other agents' belief states (see example 9). The dynamic programming solution method for POSGs finds an equilibrium by iterating through all the agents, repeatedly removing any dominated strategies from the agent's strategy space (Hansen, Bernstein, & Zilberstein, 2004). An anytime improvement on dynamic programming, the MAA* algorithm, carries out a heuristically-guided A* search through policy space (Szer, Charpillet, & Zilberstein, 2005) (Oliehoek & Vlassis, 2007). Empirically, the authors have found that optimal policies are often returned early on. However, with the exception of (Oliehoek & Vlassis, 2007) which can be applied to infinite-horizon games, these approaches find offline solutions to small finite-horizon POSGs. Other approaches such as (Nair, Tambe, Yokoo, Pynadath, & Marsella, 2003) (Yuichi, Makoto, & Atsushi, 2007) approximate by finding locally optimal policies.

More recently, a special case of much larger POSGs, the networked POSG, has attracted some interest (Kim, Nair, Varakantham, Tambe, & Yokoo, 2006) (Varakantham, Marecki, Yabu, Tambe, & Yokoo, 2007). In the case where the agents are networked according to a specific structure—such as a sensor network—it is possible to exploit this structure to develop more sophisticated strategies for agents located in critical parts of the network, and simpler strategies for agents located in less critical regions (Marecki et al., 2008).

Solutions such as the above are offline approaches, and appropriate only to finding equilibria in cooperative games. Thus they are not immediately applicable to an agent learning to act in an unfamiliar situation, even where the dynamics are known. However, the approximation techniques of (Marecki et al., 2008) may be useful for problems of the kind we are addressing. In one online approximate algorithm (Emery-Montemerlo et al., 2004), each agent tries to compute the jointly optimal action for that step and then executes its own part of the joint action. Providing that all agents are initialised with the same information (in particular, they should share a random seed), every agent will compute the same approximately optimal action so that the actions are truly cooperative. Although this algorithm is theoretically sound, it is computationally intensive and has only been tested on relatively small POSGs.

An alternative approach mirrors the extension of fully observable single-agent learning to the multi-agent case by treating other agents as part of a grand state: multi-agent POMDPs can be solved using any online POMDP algorithm by treating models of the other agents as part of the state. The branch-and-bound algorithm in (Paquet, Tobin, & Chaib-draa, 2005) is an example of such an algorithm. By incorporating problem-specific knowledge into the search (for example, treating some variables as static in the short term), this algorithm is demonstrated to work on larger scale problems. Other methods can be used to improve efficiency such as local factorisation (Kim et al., 2006) and Bayesian network representations (Sallans, 2002).

Now, these descriptions focus on problems in which the *state* is partially observable. However, in many kinds of partially observable multi-agent POMDPs, including the ambulance example (8), it will be impossible to consistently observe the actions of the other agents. We can classify such scenarios into two types: in the first type, as in our ambulance example, we simply cannot see the actions of every agent, for example because the agents have a limited field of vision; while in the second type, we can observe the effects of actions, but not the intended action itself. As an example, in many robotics problems, actions may not be completely deterministic; an agent aiming to travel in a particular direction will have some probability of successfully doing so, and some probability of travelling in another direction. Example 10 suggests a rescue scenario with this property.

Example 10 *Partially observable actions in the Tamptono rescue*

Following a number of minor quakes on the heels of the big earthquake, access to Tamptono has become so bad that a helicopter is sent to drop rescue supplies and food to the main town squares and largest buildings. Unfortunately, the wind blows the packages about as they fall towards the town, and many of them do not reach their intended target.

In a single-agent problem, the agent can model this second case within the transition function, the non-determinism of its own actions swept into the probability $P(s'|s, a)$. However, in a multi-agent problem where other agents' strategies are being modelled, it may be useful to model the underlying strategy of the opponents. Then, if the agents are upgraded (or in the helicopter example, if the weather conditions change) so that $P(effect|action)$ is changed, but their strategies remain unchanged, the learning agent can adapt its behaviour as soon as the new $P(effect|action)$ model is known. Of course, the other agents may well modify their behaviour based on the new model; still, we hope that having this explicit model may give us a head start in keeping up with them. Likewise the first case is typically handled by treating the problem as a single agent problem, and the behaviour of the other agents as a part of the environment. However, we propose that explicitly modelling the behaviour may be advantageous, reaping the benefits, already discussed, of explicit models.

Finally, for all the complexity in the above models, we wish to add one more layer of uncertainty: uncertainty about the environmental dynamics. Consider again example 6 in section 2.3: In this example, neither the complete state of the building, nor the numbers or locations of other people in the building is known to each agent at any one time. Challenging problems of this nature draw on the work in learning and multi-agent learning, in POMDPs and in POSGs. We may take either multi-agent learning as a starting point, and extend it to the partially observable domain, or we can consider ways of integrating learning with our POSG solutions.

In fact, very little of the previous work has been extended to this difficult domain. However, card games have formed a testing ground for applying learning techniques to partially observable competitive games (Shi & Littman, 2002) (Matsuno,

Example 11 *Partially observable stochastic games in Tamptono*

Twin boys are standing near the old people's home when the rescue workers leave it with all the survivors. During the rescue, several parts of the building have collapsed, and more may go at any moment. The boys know that they can dart into the building to loot it for jewellery, but if they do, that will be the signal to the rest of the gang to join them. If the rest of the gang are not put off by the risk of building collapse, they will surely all want to join in and claim some of the spoils—greatly increasing the risk of building collapse for all. The twins must decide how likely the others are to follow them in their dash into the building before deciding to make it. A possible set of outcomes for the game is summarised below. In fact, each outcome is associated with some probability and the twins will make their decision by considering all probabilities.

Outcome for the twins:

		Twins	
		Loot	*Don't Loot*
Gang	*Loot*	*BC; -100*	*S; -20*
	Don't Loot	*J; 100*	*0*

Outcome for the gang:

		Twins	
		Loot	*Don't Loot*
Gang	*Loot*	*BC; -100*	*J; 15*
	Don't Loot	*0*	*0*

Outcome $Buildingcollapse$ $BC = -100$

Outcome $Jewellery$ $10 < J < 100$

Outcome $Loss of status$ $S = -20$

Yamazaki, Matsuda, & Ishii, 2002) (Amit & Markovitch, 2006). Although in card games such as hearts (Ishii et al., 2005) or poker (Gilpin & Sandholm, 2007), the environmental model is known to the agent, the behaviour of the other players is not necessarily known. Over many games, agents can estimate the behaviour of the other players, and use these estimates alongside POMDP solution techniques to learn to play effectively. We look at the issue of learning models of the other agents in more detail in the next section.

2.3.2 Learning about other agents

Given the aim of learning models of the environment, we have previously discussed reinforcement learning. However, learning about other agents' behaviour is typically a different kind of task from learning about the environment. In a fully observable domain with the Markov assumption, the optimal action will only ever depend on the current state. Therefore, agents can learn simple models of the strategies of the other agents, using multinomial distributions over actions (one for each state) and updating these distributions either using a simple frequency count or using Bayes' rule. In the situation in example 11, this may mean the twins observing the state *(Home empty, Gang in alley)* and deciding to loot, or observing the state *(Home empty, Gang at head of alley)* and deciding not to risk it. This is known as *fictitious play* (Fudenberg & Levine, 1998). Conversely, in scenarios where the full state is unknown to the agent, simple fictitious play is not appropriate. Each agent may have knowledge of the environment and a model of the current world state—but this is not sufficient to respond optimally to the other agents. To see this, consider a rescue scenario in which some rescue tasks require several agents: the agents must come to the same conclusions about when these tasks are approached. If agents have differing views of the situation, they may not make the same decisions about urgency, resulting in an ineffective dispersal of agents. Similarly, in example 11, the twins may believe mistakenly that the gang will realise how unstable the building is, and thus expect the gang to take more care than it does, or they may not know how desperate one of the gang is for cash.

Furthermore, as we have developed the theory of learning in MDPs through increasing layers of complexity, correct and complete solutions to the problem become ever more intractable. Algorithms use tricks such as factorisations (Sallans, 1999), assume independences (Kim et al., 2006), and repeatedly approximate (Roy & Gordon, 2002) (Chalkiadakis & Boutilier, 2003), leaving the original principled approaches some way behind. If these tricks are executed carefully—if the factorisations are correct, the assumptions not too far from reality, and the approximations directed by the problem structure, then the solutions

Example 12 *Higher-level views of the Tamptono earthquake*

Following the Tamptono earthquake, a fireman on watch observes a number of fires breaking out. Although he can guess that there will be a number of other fires across the city, he can also see immediately that large areas of the city are impassable where roads have collapsed or buildings have collapsed onto the roads. He therefore concentrates his rescue team only on the reachable parts of town. Once he has decided, in collaboration with other firefighters, which fire he will tackle first, and travelled there, his focus grows even narrower, taking in the details of this fire in as much detail as possible, but requiring no information about other fires in the area.

At the same time, a helicopter observes the scene from above. The pilot's task is to report on the state of the city. Unlike the firefighter on the scene, the pilot is uninterested in details of the fires, recording only an overview. In the helicopter, all fires are categorised into a small number of size classes (such as small, medium, large*) and approximate coordinates recorded.*

found may not fall too far behind the optimum. In the next section we discuss ways of reducing the state space in learning problems, in order to render such problems more tractable.

2.4 Extending MDP techniques to larger scale systems

In section 2.2.1, we briefly mentioned approximate methods such as sampling for computing value functions in continuous MDPs. Example 12 illustrates that frequently agents will either be interested only in a high-level view of the state space (as the helicopter pilot in the example), or in some particular region of the state space (as the allocated firefighter). It is therefore reasonable to reduce the state space in either of these ways, selecting a reduction approach appropriate to the particular problem. Reinforcement learning is a technique inspired by human behaviour (Rivest, Bengio, & Kalaska, 2005), and we look again to human behaviour to see how large state spaces may be managed. There are two related techniques which are important to decision problems with huge state spaces, *function approximation* and *abstraction*. We discuss both of these in section 2.4.1. Furthermore, when state spaces are large, agent policies are correspondingly large.

In a multi-agent system where an agent is trying to model other agents, it may become necessary for the agent to approximate their policies and we discuss policy approximation techniques in section 2.4.2.

2.4.1 State abstractions

In simple reinforcement learning, agents learn tables of states and values. **Function approximation** replaces these lookup tables for state values with functions (such as neural networks or radial basis functions (Sutton & Barto, 1998)) which take state variables as inputs and output a Q-value (in model-based learning) or an action (in model-free learning). Online supervised learning techniques such as gradient descent or simulated annealing learn the function parameters, given the function form, from the experience in (state, value-estimate) pairs (Mackay, 2003).

In more detail, such functions can be learned by assuming some form for the function, and then learning the appropriate parameters. Any standard learning method which is able to handle incremental learning (the accumulated experience forms the training data) and nonstationarity can be used; neural networks are common (Sutton & Barto, 1998). In particular, a neural network with two hidden layers can be used to approximate any function with an arbitrarily small error, given sufficient training data. Alternatives to neural networks include radial basis functions or linear approximations (Sutton & Barto, 1998). For most general problems we expect that neural networks will be sufficient. Therefore, we do not propose to extend the state of the art in this direction. Consequently, although function approximation could be used in the future to extend our work into larger scale or continuous problems (see sections 8.3.1 and 8.3.5), we will not implement it in our work.

Now, when carrying out function approximation in this way, the state variables themselves may be used as inputs to the function. However, if there are a large number of state variables it may be helpful to reduce them in some way. Furthermore, in any difficult problem, the learner may not find a good approximation within a reasonable time unless the input features are carefully selected to provide guidance (Fogel, 2002). To this end, reduction of the state variables can be achieved using abstraction.

Abstraction refers to taking a high-level or abstract view of the state space. For example, rather than considering one state for every size of fire, the helicopter pilot classifying all fires as small, medium or large is carrying out state abstraction. The simplest state *abstractions* are those in which the state space is just broken up into tiles or buckets, for example by laying a grid over the state space. A less crude method of dividing the state space is to form clusters using standard clustering techniques such as nearest neighbours or k-means (Hoar, 1996). A more sophisticated clustering technique makes use of a topological mapping (Smith, 2002) which exploits the form of the state space to provide more clusters in denser parts of the space.

In a high-dimensional input space, the fact that many combinations of input variables rarely or never occur can be exploited by mapping the input space into a higher level *feature space*. Such feature spaces can also be used to encode intuitive knowledge about the structure of the state space—for example, fires will never occur on rivers.

Automatic means of encoding feature spaces include coarse coding (Sutton & Barto, 1998), in which ellipses or other overlapping partitions in the state space represent binary features; and dimensionality reduction techniques such as PCA (Roy & Gordon, 2002) and sparse coding (Lee, Battle, Raina, & Ng, 2007). Alternatively, features can be learned through supervised learning, exploiting the intuition that humans can recognise abstractions intuitively, but not always easily define them (Tanner, Bulitko, Koop, & Paduraru, 2007). Finally, features can be manually defined (Fogel, 2002).

Now, in any of the above techniques, the agent must have or learn some notion of similarity between states. With abstraction, "similar" states are treated the same and given the same action. With generalisation, as well as similarity, the learner must have an idea about direction: whether the new, unseen state has a higher or lower value than the known state. In problems such as disaster response where the environment is new to the agent, this similarity and direction information is not necessarily known. Consequently, our focus will be on techniques where the agent can learn the abstractions online and unsupervised.

In order for an abstraction technique to be useful, it must be possible for us to predict the abstract state for a new state. The agent will not know at the outset how many abstract states will be appropriate, and so it must be possible to expand or contract the set of abstract states as the situation progresses—this is especially the case with a dynamic scenario, in which underlying states may need to be moved between abstract states.

In this context, *statistical clustering* describes a class of probabilistic clustering techniques which can be used to achieve this kind of abstraction. Each state can be identified by a set of binary features (at minimum, states can be identified by numbers and the binary features the bits in the state's number). A cluster is then described as probabilities over the states in the cluster: $C =< p_1, p_2, \ldots, p_n >$ where p_i is the probability that bit i is set, given that the state is in that cluster. The probability of a new state being assigned to any particular cluster can therefore be computed using Bayes' rule. These clusters may be associated with Q-values (Hoar, 1996) or action choices. The probabilistic nature of statistical clustering means that it will fit particularly well with our Bayesian modelling techniques, so will be our choice for this work. To this end, in section 3.4.2 we will outline an algorithm for using statistical clustering within our model.

Now, these state abstraction techniques can be applied in partially observable domains to reduce either the underlying state space or the belief space. In problems where the abstraction is carried out online, only the belief-space can be reduced since the underlying state is never known. In principle, any approach suited to continuous space can be used to abstract belief space. However, POMDP belief space has a particularly sparse structure. Typically, given a particular observation o_i, there will be a small set of states S_i which could have given rise to the observation. For a set of observations $O = o_1, ..., o_n$, the set of states which could have given rise to the observations is given by $S_O = \cap_{i=1..n} S_i$. Belief in any states other than these will be small or zero. A nice approach to exploiting this sparsity is to use some form of dimensionality reduction to map belief space into a lower-dimensional feature space (Roy & Gordon, 2002). However, a disadvantage of these approaches is that a certain amount of preprocessing is necessary. In the PCA-based algorithm outlined above, it is necessary to do some sampling of state and belief space before the reduction techniques can be applied. This is not always feasible in real, dynamic scenarios. Less principled approaches such as manually defining abstractions from human input may turn out to be more practical.

Finally, we note that in some kinds of problem, such as that in example 12, different levels of abstraction may be appropriate. This motivates the use of hierarchical learning techniques (Fischer, Rovatsos, & Weiss, 2004). Such techniques integrate action and state space abstractions, mapping high-level states to high-level actions. Depending on the particular action, its execution may require traversing the hierarchy to consider more details of the state or of a localised part of the state (Naeem & Bigham, 2008). In a large real-world problem such techniques may become necessary. As with function approximation, we do not propose to extend the state of the art and thus do not address them in this thesis, but identify a scheme for future work in section 8.3.1.

However, Bayesian multi-agent POMDPs have a further bottleneck. In principle, each agent can maintain and update a POMDP in which the unknown POMDP "state" includes the world state, the other agents' world models, and behavioural models for the other agents. In practice, it is not tractable either to update such a model or to determine a best response within it, without performing some approximations. In particular, it is intractable to maintain beliefs about all the other agents' beliefs about the state. In the next section we discuss approaches to approximation within problems of this form.

2.4.2 Policy approximations

In general, rather than maintaining beliefs about other agents' beliefs, and then beliefs about their strategies based on these beliefs, it is necessary to find some way to approximate our beliefs about strategies. For example, in one online algorithm (Emery-Montemerlo et al., 2004), rather than infer over the beliefs of other agents, all agents are initialised with the same information and the same random seed. Each agent can then compute an approximation to the joint optimal action for that step and executing its own part of this joint optimal action. In this algorithm, the joint optimal action is approximated by projecting just a small number of steps into the future (a *finite horizon* search), and using a domain-specific heuristic to estimate the values of those future states. Although this is theoretically sound, the algorithm is computationally intensive and has only been tested on small POSGs.

Such finite horizon searches can be refined, using heuristics to search further down more likely avenues (Ross, Pineau, Paquet, & Chaib-draa, 2008). Providing the heuristics are good, this allows for deeper searches and thus generally better action choices without increasing the costs prohibitively. Since our interest is in providing a general model, we do not propose to consider techniques which require domain-specific heuristics here.

An alternative is to extend the finite state machine approaches discussed in section 2.2.1 to find a joint approximate policy. For example, the technique of (Bernstein, Hansen, & Zilberstein, 2005), which uses a finite state machine to describe each agent, creating dependencies between the FSMs by exploiting shared information in the style of (Emery-Montemerlo et al., 2004).

Both the above techniques are dependent on shared information and thus not directly applicable to our decentralised scenario. More recent work has investigated offline algorithms for a special case of much larger POSGs, the networked POSG. In the case where the agents are networked according to a specific structure—such as a sensor network—it is possible to exploit this structure to develop more sophisticated strategies for agents located in critical parts of the network, and simpler strategies for agents located in less critical regions (Marecki et al., 2008). This is achieved by restricting the possible policies for all the agents, with more restrictions on some than others.

In particular, agent policies are restricted to the class of policies which can be described by finite state machines. By limiting the maximum number of nodes which the finite state machine can have, policies can be restricted further (Marecki et al., 2008). Although we are not considering networked structures, we propose to take inspiration from this work to develop an online multi-agent strategy in which an agent maintains explicit models of the system and of the other agents, but its models of the other agents are approximated as finite state machines.

In more detail, an agent controlled by a finite state machine has a number of internal states, each associated with an action (or a probability distribution over actions)—this tells the agent how to act when it reaches this internal state. After taking an action, the agent's observations determine its movement to a new internal state. The finite state machine captures the notion that an agent's beliefs can be approximated, for the purposes of decision making, by a variable but finite sequence of past observations, and examples such as (Vu, Powers, & Shoham, 2006) (Carmel & Markovitch, 1996) demonstrate that it can be very effective. Furthermore, approximate best responses to finite state machines can be computed efficiently (Marecki et al., 2008)—we explain how we will use this technique in section 3.4.1.

For any set of agent behaviours, there may be several possible FSMs. The least compact FSM for a finite time period has a distinct node for every time step. The *minimal* (most compact) FSM for an agent's behaviour has the smallest number of nodes necessary to describe the behaviour exactly. Now, finding the minimal FSM is an NP-complete problem and cannot be approximated by any polynomial-time algorithm (Carmel & Markovitch, 1996). However, it is possible to learn compact FSMs in polynomial time, for many practical problems. In particular, the US-L* algorithm (Carmel & Markovitch, 1996) has polynomial running time and has been shown to be effective at finding compact models of agent behaviour on small agent coordination problems. Given this, we propose to test it on larger problems. Section 3.4.3 outlines our FSM implementation, which is tested in chapter 6.

2.5 Summary

In this chapter we have emphasised uncertainty as a key requirement which we intend to address, and discussed agent decision making under uncertainty, highlighting a principled Bayesian approach for the single agent problem. Since coordinated multi-agent systems are our goal, we pursue the multi-agent extension to this approach, emphasising its correctness and its flexibility. However, in order to use the complete Bayesian approach on anything other than toy problems, and especially if we wish to scale it up to large problems, some compromises or approximations are necessary. To this end, we discuss ways in which the Bayesian model can be adapted to larger state spaces. However, in these larger state spaces, agents are unlikely to have full observability, which the models of (Dearden et al., 1999) and (Chalkiadakis & Boutilier, 2003) both assume.

We have therefore extended the discussion of learning techniques into partially observable domains, noting that there is very little work on learning within partially observable domains which explicitly takes into account the behaviour of the other players—which we believe would lead to better performance. We have considered other coordination mechanisms which could be used in such domains, such as conventions and negotiation, and we have argued that in large and uncertain domains, these mechanisms are likely to be integrated with learning techniques.

Given this, we propose to extend the model of (Chalkiadakis & Boutilier, 2003), developing a model for Bayesian learning which explicitly considers the other agents and which is appropriate to partially observable domains. In order to render such a model computationally feasible, especially in domains with large state spaces and many agents, a number of approximations will be necessary. In particular, it will be necessary to perform some kind of state abstraction. Keeping with our Bayesian model, we will use statistical clustering to gather states together. In multi-agent problems, with agents adapting to one another's strategies, it will also be necessary to model these strategies approximately. Thus we propose to use finite state machines for this strategy modelling.

Now, to date, previous work using finite state machines focuses on offline solutions to multi-agent problems, precomputing responses to every possible belief state. However, it is impossible for every belief state to be reached, since every belief state which is visited narrows the space of possible future beliefs (at least within a static environment). For offline solvers without tight time constraints, it may not be of concern spending time solving for unreachable belief states. Alternatively, it is possible to make use of the intuition that the belief space need only be divided into sufficient chunks to determine the next action, for example using principal components analysis on a discretised state space (Roy & Gordon, 2002). The alternative to such techniques is to search for solutions online. This is the only way of approaching very dynamic systems, or systems where the problem parameters may not be known in time to perform a comprehensive offline search—as is likely to be the case in our target domain. Online solutions will, of necessity, be approximate, since any accurate solution projects infinitely far into the future and thus is effectively an offline solution.

Thus, in the next chapter we describe in detail an algorithm for online cooperative action in partially observable multi-agent systems in which agent communication is limited to information-sharing (section 3.2). Our algorithm uses finite state machines to model the policies of the other agents and each agent computes online a best response to its beliefs about these finite state machines 3.4.3.

Chapter 3

A Bayesian model of partially observable multi-agent systems

As outlined in the previous chapter, we will describe a Bayesian model for coordinated decision making in partially observable multi-agent systems. In section 3.2 we give the Bayesian MDP model itself. This general, formal model is not practically useful without further modification: the rest of the chapter explains how we build from the general model to specific implementations (although the specific cases themselves will be described in later chapters). First, in section 3.3, we explain the use of belief networks to aid visualisation of a particular problem, and how this can be applied to the Bayesian MDP model. Then in section 3.4 we identify places in which the Bayesian MDP algorithm can be approximated and explain how to apply them within our model. First, however, section 3.1 defines the terms we will use throughout this chapter.

3.1 Definitions

We begin with our basic definitions. Throughout, we assume that there is some underlying world state, s, which changes in response to the joint actions of the agents. The progression of world states and joint actions forms an

MDP. We assume that agents are not able to perceive s completely, but make some observations o from which they make inferences about the state. These observations may include communications from other agents—we do not treat those distinctly in this work. More formally, we will make use of the following definitions:

- $I : \{I_1, \ldots I_k\}$, a set of k agents
- $S : \{s_1, \ldots s_{n_s}\}$, a set of n_s states. A state will generally be described by a set of state variables.
- $L \subset S = \{L_1, \ldots L_k\}$, a location variable for each agent. These determine the viewpoint from which agents make local observations.
- $A = \{a_1, \ldots a_{n_a}\}$, the set of individual actions. $\mathbf{A} = A^k$ is the set of joint actions. Thus, we differentiate between a single action a and a joint action $\mathbf{a}$ by using bold for the latter, to emphasise that it is a vector. We will also use $\mathbf{a_{-i}}$ to refer to the vector $\mathbf{a}$ with the element corresponding to i removed, and $\mathbf{a} \circ a'$ to refer to $\mathbf{a}$ with a' integrated.
- $O : \{o_o, \ldots o_{n_o}\}$, a set of n_o possible observations. Each agent's observations will be taken from this set.
- $T_f : T_f(s_{t+1}, s, \mathbf{a}_t) = P(s_{t+1}|s_t, \mathbf{a}_t)$, the transition function from the state at time t to the state at time $t + 1$, where $s_{t+1}, s_t \in S$ and $\mathbf{a} \in \mathbf{A}$. We use the subscript f here and below to distinguish the functions from sets.
- O_f: An n_o-dimensional function where $O_f(s_t, o_t)_i = P(o_t|i, s_t)$, the observation function for agent i, where $o_t \in O$ and $s_t \in S$.
- $R : \{r_1, ...r_{n_r}\}$, $n_r \leq n_s$, a set of possible rewards which an agent may receive
- $R_f : SxAxS \rightarrow R$, a reward function: each agent will have its own reward function. Typically, the reward will be associated with the immediate state, but for some problems it may be associated with the transition between states (for example, if actions have a cost).

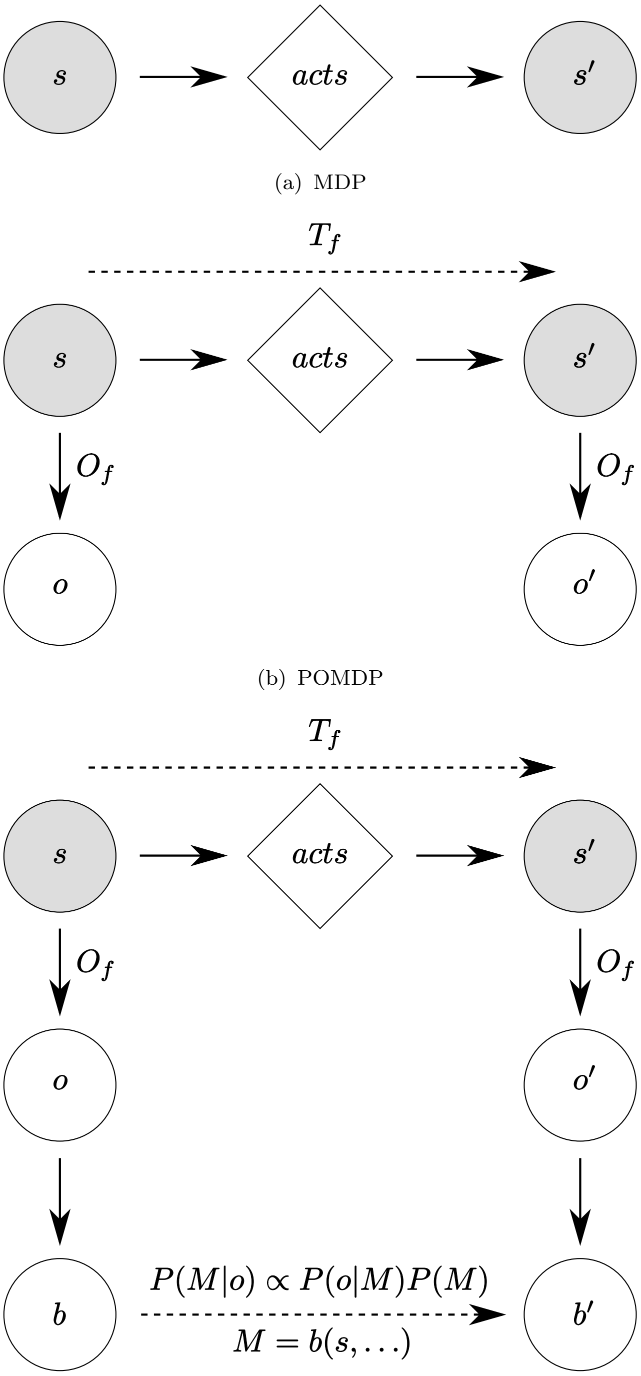
When taken together, T_f, R_f and O_f describe the dynamics of the environment. We will use $\theta = (T_f, R_f, O_f)$ to refer to these dynamics as a whole. An individual agent, A, may also have:

- A (deterministic) policy $\pi : (PxHxO) \rightarrow A$ where $h \in H$ defines all relevant historical information (observation sequences including communications from other agents), $p \in P$ any prior or domain knowledge, $o_t \in O$ is the current observation and $a \in A$ is a single agent action. Typically, (p, h) will be compressed to contain the sufficient statistics for a belief state (a probability distribution over states and unknown parameters).
- Beliefs over unknown parameters: for some variable X taking values $x_1, x_2, \ldots$, $b(x_i)$ is the probability that $X = x_i$, given the agent's prior information and subsequent observations.
- Models of the other agents' behaviour: $P(\pi_i|p, h)$ where π_i has the same form as π above and (p, h) refer to the prior and historical information of the agent A. To be clear, we assume that the other agents have deterministic policies, and our agent maintains *beliefs* over these deterministic policies.

Taking these definitions, the next section will describe a formal model of learning in multi-agent systems, extending the Markov decision process defined in section 2.2 to a continuous "Bayesian MDP" and explaining how this Bayesian MDP can be used to encapsulate partially observable multi-agent systems (section 3.2).

3.2 Bayesian MDPs

To recap, in section 2.2, we outlined the Markov decision process, or MDP, progressing from state to state (figure 3.1(a)), and the partially observable Markov decision process, progressing from world state to world state, emitting observations (figure 3.1(b)). We then showed how to turn a partially observable MDP into a continuous, fully-observable belief-MDP (figure 3.1(c)).



(a) MDP

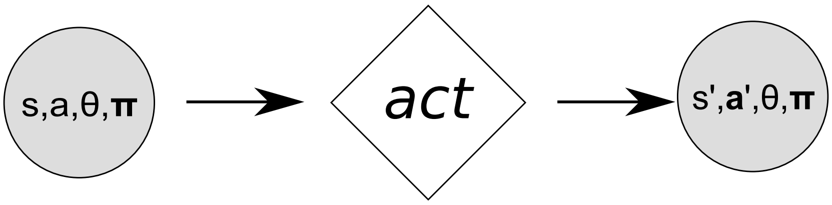
(b) POMDP

(c) belief MDP

Figure 3.1: MDP, POMDP, and belief-MDP

We then discussed a model which uses a belief-MDP formulation to carry out reinforcement learning in a fully-observable multi-agent system, by treating the environmental parameters as a part of a partially observable state (Chalkiadakis & Boutilier, 2003) (figure 2.5). A similar model achieves reinforcement learning in a partially observable single-agent system (Ross, Chaib-draa, & Pineau, 2008): figure 3.5. Finally, a related approach treats a partially observable multi-agent system as a series of *Bayesian Games* (Emery-Montemerlo et al., 2004): each step of this system is treated as a game, using Bayesian inference over the other agents' beliefs.

In fact, all these models can be considered to be special cases of one general model. Consider all the possible variables and parameters of a partially observable multi-agent MDP: the world state s, the agents' actions $\mathbf{a}$, the environmental models $\theta = (T_f, R_f, O_f)$, and the agents' strategies π. We can create a multi-dimensional "grand state", which contains all of these things, $g = \{s, \mathbf{a}, \theta, \pi\}$ and describe a "grand MDP" which proceeds through such states given the single agent action *act*:



In this MDP, provided the environment and agent behaviours are static, θ and π are unchanged between states, while the transitions for the state are determined by θ—the dynamics for the underlying MDP—and $\mathbf{a}$, and the transitions for the joint action are described by π—the agent behaviour models—and s. From this MDP, we can describe a POMDP in exactly the same way as any POMDP is described from an MDP with partially observable states. In order to define the POMDP from the MDP we must specify the observations. As in (Ross, Chaib-draa, & Pineau, 2008), (Emery-Montemerlo et al., 2004), the set of observations for agent i in this POMDP includes:

Example 13 *Tamptono rescue agent's observations*

A rescue agent arriving outside a damaged building observes:

- *The outside of the building: o_s (but not the internal state of the building)*
- *Rubble flying out of a window, from which she infers that an agent is digging near to the window: o_a*
- *A man hurrying from the building and handing some papers over to a man nearby, and receiving some money (she can't see how much): o_r*

- o_s: observations about the state
- o_{a_j}: observations about agent actions
- r_i: the individual agent's rewards
- o_{r_j}: (in a non-cooperative system) observations about the rewards of others
- $o_s \rightarrow a_i \circ o_a \rightarrow o_{s'}$: observations about the transitions[1].

These are illustrated in example 13. From this POMDP we can define a belief MDP, again in exactly the same way as we have before: the "belief states" of the belief MDP contain probability distributions over the variables in the grand state: we describe the belief state as $b(s, \{a_j\}, \theta, \{b_j\}, \{\pi_j\})$ where:

- s is the current world state
- $\{a_j\}$ are the immediate actions of the other agents
- $\theta = T_f, R_f, O_f$, the environmental dynamics
- $\{b_j\}$ are the beliefs of the other agents about the world state
- $\{\pi_j\}$ defines the behaviour of the other agents based on their beliefs

These definitions allow us to define a general belief MDP algorithm, algorithm 1 for an agent i in a partially observable multi-agent scenario. In practice, however, steps 2 and 3 in this procedure may involve complex calculations. In particular, the update step (step 2) must sum over all the unknowns in the environment, which is potentially a multi-dimensional summation or integral, while the best

[1] These observations combine the state and action observations from two consecutive timesteps. Here we use the → notation to indicate that the state leads to the action choice, and the actions lead to the state transition (similar to figure 3.1).

response step must evaluate over many states, with the evaluation cost increasing exponentially with each step projected into the future. We therefore look at ways of approximating algorithm 1 which will be suited to efficient online evaluation.

First, however, we show a different view of the MDPs using graphical models known as *belief networks.* The belief network visualisation provides a convenient way to think about the update step and we will be reusing it in sections 5.1 and 6.1 when we consider two particular instantiations of this model. Additionally, there are a number of efficient algorithms for calculating within graphical models and we will discuss in particular the junction tree algorithm for the update step in section 8.3.4.

Algorithm 1 The general belief MDP algorithm

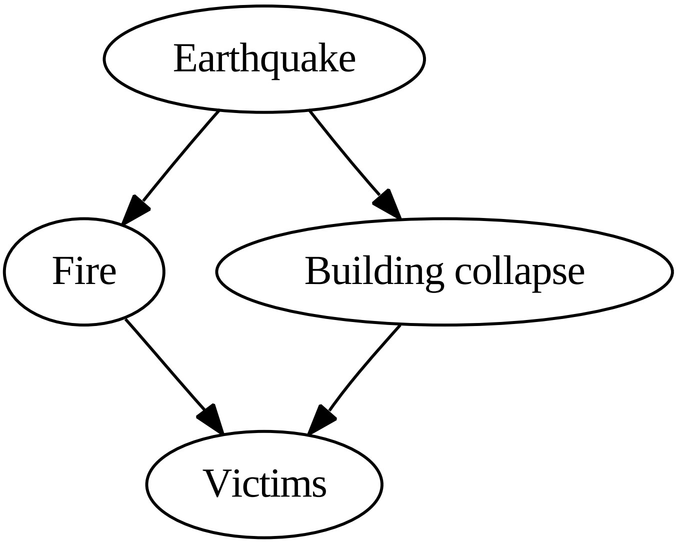
```
• The agent initialises its belief state $b_0 = b(M)$ where $M = (s, \{a_j\}, \theta, \{b_j\}, \{\pi_j\})$, either uniformly or based on domain knowledge.
• At each timestep the agent $i$:
    1. Makes its observations
    2. Updates its beliefs over $M$ using Bayes' rule, resulting in a new belief state $b_t$.
    3. Calculates the action $a_i$ which optimises $Q(b_t, a_i)$, where
           $Q(b_t, a_i) = \int_M P(M|b_t)Q(s, a_i|M)$
    4. Executes this action and makes new observations.
```

3.3 Belief networks

A Bayesian network, such as figure 3.2, shows dependency relations between variables, with directed arrows indicating the direction of causality: an arrow from a node A to a node B indicates “A directly affects B”. Figure 3.2 shows a simple model for an earthquake: earthquakes are assumed to occur independently (with some probability), so *Earthquake* forms a root node in the network. The likelihood of a fire is dependent on whether or not an earthquake has occurred;

this is represented by drawing an arrow "*Earthquake* affects *Fire*". The left-hand table, called a conditional probability table or CPT, shows the conditional probability of a fire given that an earthquake did occur (*Earthquake*=1) or did not occur (*Earthquake*=0). Similarly, we have an arrow "*Earthquake* affects *Buildingcollapse*" (the corresponding CPT is omitted). Finally, both fires and building collapse may result in victims and so we have arrows from each to *Victims*. The right-hand table shows the conditional probabilities for finding victims given all possible combinations of values for (*Fire*, *Earthquake*): the "parents" of the *Victims* node.

Such a diagram encapsulates the dependences between variables and thus the independence relations between them. Now, in general, in a probabilistic system, the probability of a particular variable assignment can be factorised using the product rule:



	Earthquake	
Fire	0	1
0	0.9	0.5
1	0.1	0.5

	Fire, Earthquake			
Victims	0,0	0,1	1,0	1,1
0	0.9	0.3	0.6	0.15
1	0.1	0.7	0.4	0.85

FIGURE 3.2: Example Bayesian network, with the CPTs for the fire and victim nodes

$$P(x_1, \ldots, x_n) = \Pi_{i=1}^{n} P(x_i | x_{i+1}, ..., x_n)$$

Where the conditional independence relationships in the system are known, this factorisation can be simplified given the fact that $P(A|B, C) = P(A|C)$ if A and B are conditionally independent given C. Now, the variables can be ordered in any way before applying the product rule, and some orderings permit more such simplifications than others. To see this, consider a particular variable assignment; $Fire = f, Earthquake = e, Buildingcollapse = b, Victims = v$ over the variables in figure 3.2. If we order the variables e, f, b, v, then the product rule gives us:

$$P(f, e, b, v) = P(e|f, b, v)P(f|b, v)P(b|v)P(v)$$

which permits only one simplification:

$$P(f, e, b, v) = P(e|f, b)P(f|b, v)P(b|v)P(v)$$

However, if we order the variables (v, e, b, f), then the product rule gives us:

$$P(v, e, b, f) = P(v|e, b, f)P(e|b, f)P(b|f)P(f)$$

which has two simplifications:

$$P(v, e, b, f) = P(v|b, f)P(e|f)P(b|f)P(f)$$

Furthermore, there is no general way to find an optimal ordering (Mitchell, 1997). However, the Bayesian network diagram supplies a way of ordering efficiently. This is to order the variables so that every variable has a lower number than all its parents, and then each node need be conditioned only on its parents (see for example (Mitchell, 1997), chapter 6 for more details).

Now, in any problem, we will have some variables which are observed, and others which are *hidden*. We care about the likelihood of some of the unknown variables, but not all. For example, if an earthquake occurs, a disaster response agency will have ambulances to dispatch to victims, and fire engines to dispatch to fires, but no services to dispatch to collapsed buildings: collapsed buildings are only of interest because of their effect on the likelihood of victims. Thus, if the buildings are not observed there is no need to compute the likelihood of their collapse. Thus, we marginalise out the unknown variables which do not interest us (summing over them), and normalise over the observed variables. In the belief network visualisation, as with the MDP progress diagrams in the previous section, we will distinguish between the observed variables and the unknown variables by shading unknown variables gray and leaving observed variables unshaded.

Bringing together these techniques gives rise to the following standard algorithm for calculating the probability of a hidden variable v in a system:

1. Identify all the variables and parameters in the system

2. Draw up a Bayesian network diagram with a node for each variable or parameter, indicating the "directly affects" relationship between nodes.

3. Shade the nodes which correspond to hidden variables or unknown parameters.

4. For the node corresponding to the variable of interest, write down its probability, conditioning on the observed (unshaded) nodes $\{obs_1, obs_2, ...\}$ and summing over all the other hidden (shaded) nodes $\{v_1, v_2, ...\}$:

$$P(v|obs_1, obs_2, ...) \propto \sum_{j=1,2...} P(v_1, v_2, \ldots, v, obs_1, obs_2, ...)$$

5. Using the Bayes network as a guide, factorise and simplify $P(v_1, v_2, \ldots, v, obs_1, obs_2, ...)$, removing any constant factors (i.e., factors independent of v) as these can be normalised over later. To achieve this factorisation, first, note that the term within the sum contains every node in the system. Begin at the leaves of the network and for each leaf l, write down the term $P(l|parents(l))$ where $parents(l)$ refers to those nodes which

> directly affect l. Then, for each node p in $parents(l)$, write down the term $P(p|parents(p))$. Continue in this way until you have included a term for every node in the system. This gives the factorisation. To simplify the equation, move any terms which contain no summed-over nodes outside the sum. Any terms which contain only observed variables may be removed as constant factors.

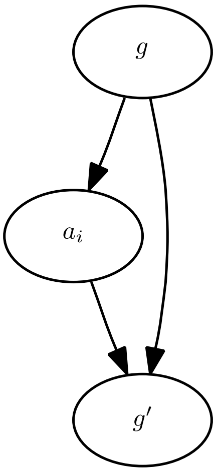
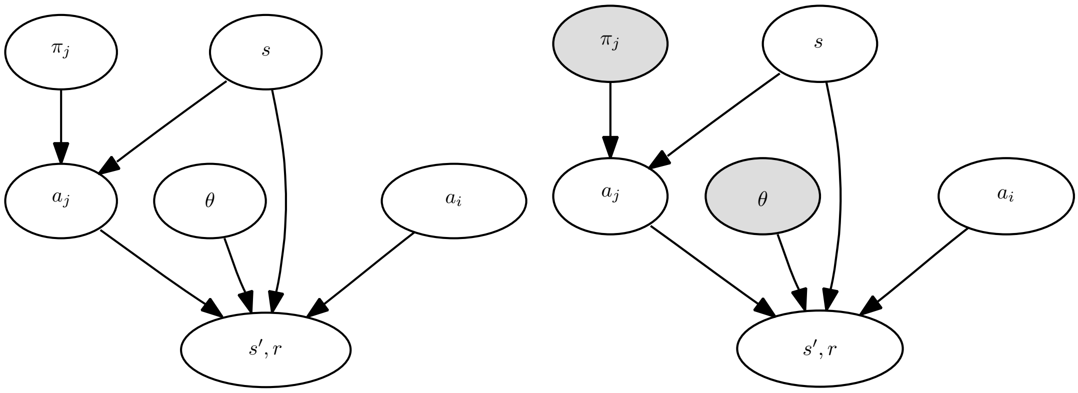


FIGURE 3.3: Simple MDP belief network

Now, we propose to apply this technique to our MDP system. To do this, we draw up one agent's belief network. This network will contain variables for the current state g, the agent's action, and the future state (3.3). In practice, rather than use a single variable g to represent the grand state, we can separate out each of the variables in the grand state, thus making explicit the dependencies between variables and allowing us to exploit any conditional independences.

Figure 3.4 shows the belief networks which correspond to a fully observable multi-agent MDP with known environmental and agent models (3.4(a)) and with unknown models (3.4(b)): the latter is the model of (Chalkiadakis & Boutilier, 2003). In both these networks, the single leaf is the (state, reward) observations for the new state s'. These depend on my action a_i, the actions of others a_j (which together form a joint action $a_i \circ a_j$), the previous state s, and the environment dynamics θ. My own action is assumed to be independent because its CPT is under my control, but the actions of others depend on the previous state s and on



(a) Fully observable multi-agent MDP

(b) Multi-agent MDP with unknown models

$$P(\theta|obs) \propto P(\theta)P(s', r|s, \theta, a_i \circ a_j)$$
$$P(\pi_j|obs) \propto P(a_j|s, \pi_j)P(\pi_j)$$

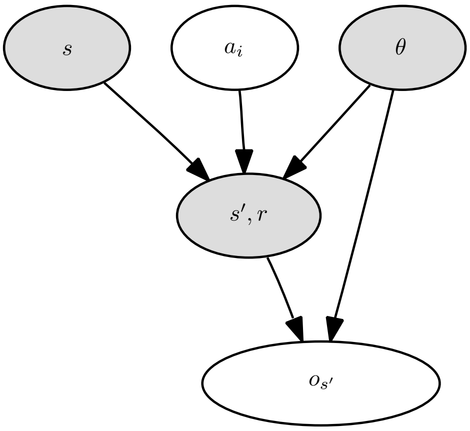
FIGURE 3.4: Bayes networks for the multi-agent MDP, with the equations for determining the likelihood of the two unknown nodes shown.

their behaviour models π_j[2]. The only difference between the two models in figure 3.4(a) and 3.4(b) is what is observed, indicated by the shading.

Given the network in 3.4(b), with its two hidden nodes, we can apply the algorithm described above during the update step of algorithm 1 to calculate separately the probability distribution for the unknown node π and the probability distribution for the unknown node θ. The factorised equations derived by the above algorithm are shown in the figure.

Similarly, figure 3.5 shows the POMDP with unknown models described in (Ross, Chaib-draa, & Pineau, 2008). Unlike the previous MDPs, this is a single-agent model and so does not have nodes for the actions or behaviour models of other agents. However, we have a new node o' representing the observations which arise from the current state s' and corresponding reward, which are now shaded as a hidden variable. We do not have a node representing the previous step's observation because once we have calculated $P(s)$ we consider the observation redundant. Below the figure are shown the associated updates. In this POMDP, we do not recalculate the probability for the hidden previous state s, which, being

[2] The network as shown is abbreviated: correctly there should be one node for each of the a_j and π_j. We use this abbreviation for all our network diagrams.



$$P(s', r|obs) \propto \sum_{s,\theta} P(o'_s|\theta, s')P(s', r|s, a_i, \theta)P(s)P(\theta)$$
$$P(\theta|obs) \propto P(\theta) \sum_{s',s} P(o'_s|\theta, s', r)P(s', r|s, a_i, \theta)P(s)$$

FIGURE 3.5: Single-agent POMDP, with update equations shown

a root node, would come out to $P_{t+1}(s) \propto P_t(s)$, but only for the current state s', as well as the environmental model θ.

Finally, figure 3.6 shows the "grand MDP" system described in the previous section, which combines the single agent POMDP of figure 3.5 and the multi-agent learning MDP of figure 3.4(b). To help draw together the ideas discussed, we explain this figure in detail. As before, we represent the others by the single agent j. In practice there may be several distinct "agent j"s, each requiring its own set of nodes.

- At the top, the "starting" nodes are the strategies and beliefs of the other agents, the π_j and b_j. In general, b_j refers to an agent's beliefs about the current state. These strategies and beliefs are hidden to us.
- Given beliefs about the state, and a strategy, the other agents decide on their actions, the a_j. We also decide an action, a_i. We may be able to make some observation about the others' actions, o_j.
- Given the true state s (hidden to us), the actions a_j and a_i, and the environmental dynamics θ (also hidden to us), a state transition takes place resulting in a new state s' and emitting a reward r.

- The new state is also hidden to us, but we can make some observations about the state, $o_{s'}$.
- Similarly, the other agents will make their own observations about the new state, $o_{s',j}$. From these observations they will update their belief states to give new belief states b'_j.

Thus, the core of the figure (outlined in green) contains almost the same structure as figure 3.4(b), but with the a_j and s', r nodes now hidden.

Looking at figures 3.5 and 3.6, we can see again that when there are hidden variables which cannot be normalised or marginalised away, as in the case of partially observable states, the update step of 1 will have to perform multi-dimensional summations, making timely updates a challenge in large state spaces. Similarly, the computation of the best response (the Q-values) involves summing over all the hidden nodes and thus the complexity increases with the number of unknowns. In the next section we detail how we will tackle the problem of carrying out these updates and best response calculations to find satisfactory solutions in online situations.

3.4 Improving efficiency

To deal with the bottlenecks we have identified, we propose to make use of the following techniques introduced in chapter 2, extending and adapting to our model where appropriate: finite horizon best response calculations (section 3.4.1), state abstraction using statistical clustering (section 3.4.2), policy abstraction using finite state machines (section 3.4.3) and efficient sampling techniques (section 3.4.4). The rest of this chapter explains these choices and gives the particulars of their integration with our general Bayesian model.

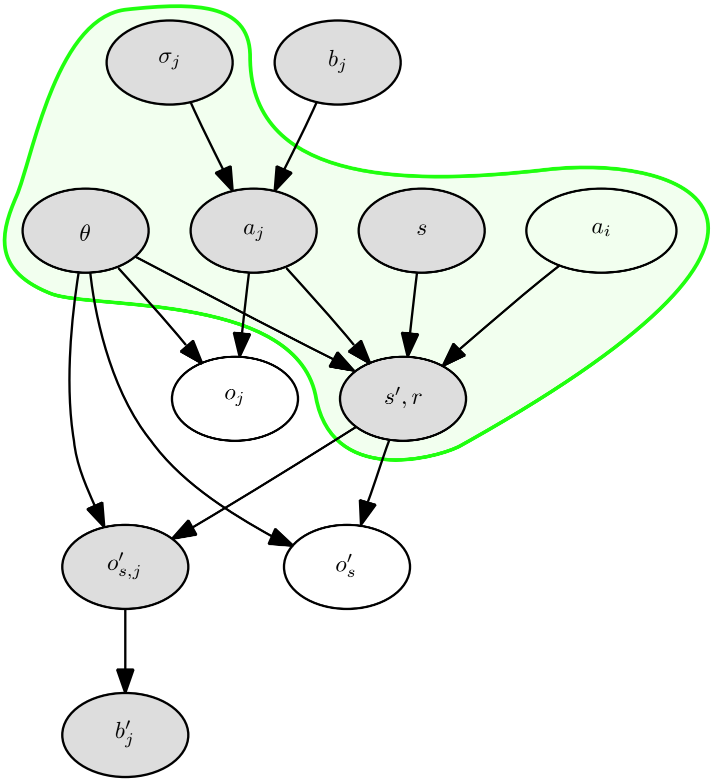


FIGURE 3.6: Complete partially observable multi-agent network

3.4.1 Finite-horizon Q-computation

Previous work (Vu et al., 2006), (Carmel & Markovitch, 1996) has considered agent coordination in fully-observable, but non-Markov, repeated games. In such scenarios finding a best response is straightforward, since the state and consequently the reward can be computed at every step. By contrast, in our work, the state is not known. We have discussed how this adds to the complexity of the belief state and associated best response calculation, by requiring us to maintain estimates about the other agents' observations.

We therefore consider an approximation technique which exploits the fact that states further into the future contribute less to the immediate Q-value. Considering an idealised case where the policies of the other agents are known, we revisit the Q-computation (equation 2.6) showing its expansion to n steps into the future. Here we treat a as a joint action, just summing over the future states, and treat the reward as deterministic. More correctly, r should refer to the expected reward.

$$\begin{aligned} Q(a,b) &= \sum_{s'} P(s'|a,b)[r+\gamma V] \\ &= \sum_{s'} P(s'|b,a)[r+(\sum_{s''} P(s''|b',a')[\gamma r+\gamma^2 V'])] \\ &= \ldots + z\gamma^n V' (\text{for some value } z) \end{aligned}$$

Providing $\gamma < 1$, for sufficiently large n (depending on the value of γ), $\gamma^n \approx 0$. Therefore, we will not lose much accuracy by cutting off the Q-computation after n steps: a *finite-horizon* approximation to the Q-value using the following recursion:

- $Q_k(b,a) = \sum_{s'} P(s'|b,a)[r(s') + \gamma V_{k-1}(b')]$
- $V_0(b,a) = 0$
- $V_k(b) = \max_a Q_k(b,a)$

These equations are finite and can in principle be solved. Such finite horizon algorithms have been used in related belief-state problems in many cases: in observable problems with unknown parameters (the base case is to assume that the current parameters are correct, and solve the corresponding MDP (Chalkiadakis & Boutilier, 2003)), in finding offline solutions for networked POMDPs (Marecki et al., 2008), and in online partially observable stochastic games (Emery-Montemerlo et al., 2004).

The base case of this recursion, $V_0 = 0$, is a heuristic, and a number of possible heuristics of varying complexity are available (Oliehoek & Vlassis, 2007). For example, we could (during an offline phase) apply the Bellman equations to the underlying joint-action MDP to obtain an optimal joint policy for the equivalent fully-observable system, and the corresponding values, which we refer to by $V_{MDP}(s)$. We could then use (Kaelbling et al., 1998):

$$V_0(b,a) = \sum_{s'} P(s'|b,a)[r(s') + \gamma V_{MDP}(s')]$$

However, for large problems this solution itself is likely to be intractable, while for problems in our disaster response domain the underlying MDP may not be available for an "offline phase" before the online solution is needed.

Another simple alternative is to use some domain-specific characteristics to estimate the value of the immediate belief state. For example, in a disaster response scenario where reward is measured in lives saved, we might make the estimate that half the people alive in a particular state will ultimately be saved. The heuristic value of the belief state b is then $\sum_{n=1}^{\infty} 0.5 * P(num_{alive} = n|b)$. Choosing good heuristics is a problem of its own, typically involving building up a body of experience with similar problems, thus we do not propose to use a domain-specific approach in our work. In principle, this work could be extended by replacing the $V_0 = 0$ base case with a domain-specific heuristic, potentially improving performance in domains where such heuristics are available.

In more detail, using the $V_0 = 0$ heuristic, we can make use of finite-horizon calculations using the following online algorithm, given knowledge of other agent policies:

- Initialise the belief state with $N = \{n_0\}$, $b(s)$ is uniform (or biased given domain-specific knowledge)
- At each step:
  - Compute a finite-horizon best-response to the current belief state
  - Carry out the prescribed action, observe the environment, and update the belief state accordingly.

In practice, however, such an algorithm is not really very much use since if the other agents' policies are known beforehand then we can compute an offline solution by adapting POMDP solution techniques to take the current node into account.

In our partially observable setting, where the agent does not in fact have knowledge of the policies of the other agents, but rather has beliefs over possible policies, our work will extend the finite-horizon approximation to use weighted samples from our belief state. In more detail, we sample from the possible policies to obtain a selection of sets of policies, $\Sigma = \{\Sigma_1, ...\Sigma_m\}$. For each sample policy set $\Sigma_s = \{\pi_1, \pi_2, ...\}$, containing a policy for each other agent, the agent computes the best response action assuming the other agents' policies are fixed to Σ: $BR_i(\Sigma_s, b)$. The action decision is then given by:

$$a = \max a_i \sum_{i=1}^{m} P_i.\delta(BR_i(\Sigma_s, b) = a_i)$$

(where $\delta(A = B) = 1$ if $A = B$ and 0 otherwise).

Now, when carrying out such finite-horizon approximations, for each step into the future we must calculate the Q-value of every state, along with its probability of occurring. Therefore, the complexity is exponential in the number of states. In our next section, we look at using statistical clustering to create abstract states, reducing the number of states.

3.4.2 State abstraction using statistical clustering

In section 2.4.1, we introduced the general technique of statistical clustering as a probability-based state abstraction technique which will dovetail well with our Bayesian learning model. The idea behind statistical clustering is to maintain an amorphous set of clusters based on the probability of the features of the states within them. Specifically, suppose that the state is represented by a set of binary features, then a cluster is represented by $C = \langle p_{bit1}, ...p_{bitn} \rangle$, where $p_{bit1} = P(bit1 = 1|s \in C)$. This last formula is obtained based on the prior probability of bit1 and the relative probability of bit1 among the states already in the cluster C. Then a new state's *match* to a particular cluster can be given by multiplying these probabilities. In more detail, the clustering algorithm we will use, based on that of (Hoar, 1996), will perform the following operations:

Updating: Each observed state is assigned to the cluster for which it has the highest probability. The bit-probabilities for that cluster are then updated according to the states. In order to do this it is necessary to maintain a count of the number of states in the cluster, then when the kth item is added to the cluster the new probabilities can be adjusted using the old probabilities and the state count.

Merging and Splitting: If a single cluster appears to be associated with an unbalanced state space, that cluster can be split into two. Ideally, this is done by dividing the states between the two new clusters so that they are as distant as possible (using a distance measure such as the Hamming distance (Sutton & Barto, 1998)). It is possible to achieve this in $O(n^2)$ by comparing every pair of states. For efficiency, we instead use a heuristic to do this, listing the states as rows of binary features and maintaining them in lexicographical order in the cluster. We then assign the top third of the states into one cluster and the bottom third into the other cluster. The remaining third of the states are allocated (top-down) to the nearest cluster. By doing this, we can greatly reduce the time required to split the cluster, while still having the states in the two new clusters fairly distinct: states with many ones in will be in one cluster and states with many zeros in will be in the other.

By contrast, if two clusters seem to be fairly similar—that is, if states are often a similar distance from both clusters (relative to their distance from other clusters—say, within 10%)—then the two clusters can be merged by placing all the states in the same cluster and recomputing the bit-probabilities.

Learning with merging and splitting: Each time we change the number of clusters, we have to update the relevant learned functions. We do this by adjusting the weightings:

$$P(action|cluster) = \sum_{cluster_{upd}} P(action|cluster_{upd})P(cluster_{upd}|cluster)$$

and similarly for the transition function.

Note that we can compute $P(cluster_{upd}|cluster)$ straightforwardly by considering for every state in *cluster* its likelihood of occurring in $cluster_{upd}$.

We can extend this model from binary clustering to n-ary features in one of two ways:

- Convert n-ary features to binary features, so that $v \;=>\; (x_1, ...x_n)$ is converted into the set of binary features $\{x_j\}$ corresponding to the bits in v.
- Use an n-ary model directly, so that instead of $P(bit_i)$ representing presence or absence of bit_i in the cluster, we have $P(bit_i = x_j)$ and for each cluster, $\sum_j P(bit_i = x_j|C) = 1$.

If n-ary features are converted to binary features, then there will be correlations between binary features: if a certain state is particularly common, then all of the binary features associated with that state will be common. For example, consider a 4-ary variable whose binary representation is (x, y) with the four values $0, 1, 2, 3$ represented as $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$. If this variable commonly has the value '3' $(1, 1)$ and rarely has the value '2' $(1, 0)$, then we would generally expect to see the binary variable x set to 1 only when the binary variable y is set to 1.

Furthermore, if the number of possible values for v is not a power of two, then some binary feature combinations will be impossible. However, the statistical clustering model does not exploit this correlation. This means that if we use binary features we lose some information from the clustering model. We therefore choose to use n-ary features, despite the slight additional complexity they bring to the model.

Now, when the state is fully observed, we can cluster over states straightforwardly as described above. If the state is not fully observed, then we can extend the model to cluster over states using the beliefs we have about the states. However, since we plan to use a policy approximation technique which will lead to policies over observations rather than belief states, it is of more use to us to cluster the observations: this will make the learned policies more compact. Below, we describe how our work will adapt the model to cluster observations.

3.4.2.1 The partially observable state model

To recap, in this model we are aiming to use the clusters: (1) to correctly decide an action for a new or rare observation based on the actions for similar observations; (2) to make our models of the other agents more compact. Now, as discussed in chapter 2, we will be using finite state machines to model the other agents. Thus, our FSM models of the other agents are based entirely on observations rather than underlying states, while our action decisions are based on these FSM models and our (known) model of the world, using our observations to predict future states. For the latter, using clusters rather than the observations would involve either computing state probabilities from clusters, or calculating a transition function from cluster to cluster. We believe that either of these would be unnecessary effort, and instead focus on using clusters to create more effective FSM models.

Now, creating a FSM relies on having foreknowledge of the *alphabet* of the machine. To this end, we propose to have a predetermined maximum number of clusters and to identify the alphabet with cluster-numbers. Given a fixed set of clusters, we proceed by assigning a new observation to a cluster and then adding to the FSM as an *example* according to the finite state machine algorithm (which is described in more detail in section 3.4.3). In more detail, we adjust the statistical cluster operations as follows:

Merging clusters means joining together two arcs from the FSM and their associated nodes. We create a new node whose action is the action associated with the larger cluster, and label it as an action which can be superseded (details of the FSM algorithm will be given in section 3.4.3), unless the actions in both nodes were fixed actions and are the same.

Splitting clusters means creating two arcs from one, each having an associated node. We duplicate the node from the original arc, and label the actions on each node as actions which can be superseded.

Now, a naïve clustering implementation will, for each new observation, compare every variable in the observation with every variable in every cluster in order to come up with a final choice of cluster. However, this naïve implementation can become very slow for large numbers of clusters (in the order 10^3 — much lower than the number of states).

However, we expect that some variables will be more important than others in determining clusters—i.e. variable probabilities will have a large variation between clusters. By investigating these variables first, we can immediately consider many clusters to be unlikely. We therefore propose that rather than storing a list of clusters with the details of each variable in each cluster, we store a list of variables, ordered from most informative (lowest entropy) to least informative (highest entropy), and for each variable we store the details of the clusters. Now, heuristically, we will drop the least likely clusters after checking each variable—all clusters which have a probability of less than cp% of the maximum. We have experimented with some values of cp and found that the exact choice does not make a big difference to the behaviour. Thus, we will (arbitrarily) set cp to 20%, since dropping a fifth of the clusters seems like a reasonable balance between not keeping irrelevant clusters and not disposing of too many clusters at once.

For this implementation, it becomes very convenient to use binary features. With binary features, each feature variable can maintain the list of clusters with the likelihood of observing the feature variable from high to low. We can convert an n-ary variable to ($log_2 n$) binary features [3]. Each of the binary features corresponds to one bit in the binary representation of the variable's value.

Now, given state clustering, the other major bottleneck in evaluating within the partially observable state problem is the double summation over agent policies and

[3] if n is not a power of 2, then let n' be the next power of 2 and we have $log_2 n'$ features.

agent beliefs (figure 6.1). We propose to address this bottleneck by approximating agent policies with finite state machines. The next section describes this in detail.

3.4.3 Policy abstraction using finite state machines

As outlined in chapter 2, we will use finite state machines to model individual agent policies in a multi-agent setting.

In this section we detail how to model agent policies using finite state automata (3.4.3.1 and 3.4.3.2). We then explain how these models fit with the multi-agent POMDP solution techniques described above, giving an algorithm for online learning (3.4.3.3) and explaining how this model extends previous work in the area. First, however, we begin with the definition of a finite state machine.

3.4.3.1 Definitions

A deterministic finite state machine has:

- A set of n nodes $N = \{n_1, \dots n_n\}$
- A set of m edges $E = \{e_1, \dots e_n\}$
- For each node, an associated action a from the set of actions
- For each edge, an associated observation o from the set of observations

One of the nodes is designated as a start node, n_0. We write $Act(n)$ to refer to the action associated with a node n.

An agent's policy is determined by such a state machine (algorithm 2): at each node (or agent state), the agent carries out the associated action. The resulting observations determine the agent's transition to a new node within the FSM.

Now, in order to use finite state machines as representations of agent policies in unknown multi-agent scenarios, we are proposing to do two things: (1) to learn

Algorithm 2 A finite state machine policy

- The agent begins at the start node n_0.
- The agent performs the action associated with the current node n.
- When all agents have performed their action, the system moves to a new state s, supplying agents with observations o.
- The agent moves along the edge associated with o, arriving at a new node n'.
- Repeat.

the finite state machine models over time, from the sequence of observed actions and state observations, and (2) to derive an online policy as a best response to a set of (beliefs over) FSM policies. We describe each of these in turn, bringing them together in section 3.4.3.3.

3.4.3.2 A polynomial FSM learning algorithm

In principle, learning a deterministic finite state machine from a set of observations can proceed as follows (Carmel & Markovitch, 1996):

- **Base case:** initialise the FSM with the single node n_0, setting the associated action to the first observed action
- **Recursion step:** given a FSM and an observation string (i.e. a sequence of observations), determine if the observation string is consistent with the FSM:

  1. Find a node whose action corresponds to the first action in the string: *if there are no untested nodes remaining, FAIL*
  2. Follow the FSM as prescribed by the observation sequence until (a) the action associated with a particular node does not match the action in the sequence: *FAIL, return to 1* or (b) the end of the sequence is reached: *CONSISTENT*

> If the observation string is consistent, then no further action need be taken. If the observation string is inconsistent, then we select a node from one of the failure points, and expand the FSM to include the new string.

Then, given a FSM and a particular (short-term) observation history, we can construct a list of possible current nodes for the corresponding agent by considering each of the starting points consistent with the observation history and following the FSM through to a current node from each (abandoning any inconsistent nodes *en route*). The probability of each resulting current node will be the total probability of all start nodes which reached it, with that probability having been computed in a previous step.

However, there are a number of difficulties in applying this algorithm within our Bayesian POMDP model. Firstly, observation histories can be of indefinite length, i.e. we may find ourselves storing the entire observation history in order to accurately build the FSM. Secondly, although the FSM is a deterministic model, the behaviour it is modelling may be neither deterministic nor static. Finally, we do not in fact know the observation histories of others, but rather have probabilities over them which are based on our own observations. We therefore wish to adjust our learning strategy to take these facts into account.

To this end, we propose the detailed algorithm, algorithm 3, for learning the FSM corresponding to agent j. The choices in this algorithm address the issues we have identified, maintaining compactness of FSMs and emphasising recent events in learning the FSMs, as follows:

Firstly, by defining a maximum number of nodes, we control the size of the learned FSM, thus ensuring compactness. We must decide the maximum number of nodes heuristically, in a domain-specific fashion, based on our beliefs about the problem. Intuitively, the more historical context is required to make a decision, the more nodes will be needed. Now, in a partially observable problem, this historical context will include observations made long ago which correspond to the last time we saw a particular agent or region. Therefore, in general, the larger the problem,

Algorithm 3 An algorithm for learning a finite state machine from beliefs

1. Initially, determine a maximum number of nodes which can occur in the FSM.
2. Run the environment for some time t
3. From our agent's observations and resulting belief states over the t steps, sample possible strings of t observations for agent j.
4. Using a sliding time window of length l, break these strings up into shorter overlapping strings.
5. Calculate the probability of each length l observation string by considering the probability of the individual observations:

   $$P(o_j|b_i) = \sum_s P(o_j|s)P(s|b_i)$$

   and the frequency of the strings among those sampled.
6. Adjust the probabilities to give more weight to more recently sampled observation strings
7. Add these observation strings to the finite state machine using the US-L* algorithm, which we describe shortly, except:
8. Rather than resolve inconsistencies by creating new nodes, resolve inconsistencies by appealing to the likelihood of each of the inconsistent strings, and discarding the least likely.

the more context we will need and consequently the more nodes in our FSM. In section 6.2.2.1 we experiment to find suitable FSM sizes for a problem from the disaster response domain.

In fact, the length of this context is the second issue addressed above. By fixing the length of possible observation strings we also limit the possible number of nodes in the FSM: every unique node in the FSM is the terminus of at least one observation string, and if there are o_l observation strings over n_{obs} possible observations, then there are at most $n_{obs}^{o_l}$ such strings. In practice we will still want far fewer nodes in the FSM than this theoretical maximum. In section 6.2.2.1 we experiment to find appropriate history lengths for the same example problem.

Having defined a maximum number of nodes, we then use probabilistic techniques to resolve the problem of the same observation string corresponding to two different actions. By weighting more recent observations more highly, we ensure that our FSMs will adapt to changing circumstances as the scenario progresses.

Given these adaptations, we now describe in more detail an algorithm for learning FSMs from observation strings. Now, finding the *minimal* FSM is NP-complete and cannot be approximated by any polynomial-time algorithm (Carmel & Markovitch, 1996). However, it is possible to learn reasonably compact FSMs (i.e., without large numbers of redundant nodes) in polynomial time, for many practical problems. For example, the US-L* algorithm (Carmel & Markovitch, 1996) has polynomial running time and has been shown to be effective at finding compact models of agent behaviour on small agent coordination problems—it is this algorithm which we modify as described above.

This algorithm models the FSM using a table, with rows corresponding to observation string prefixes os, columns corresponding to string suffixes o, and the table entries corresponding to actions a. The alphabet of possible observations is O and the set of observation strings is OS. The table is then partitioned into equivalence classes:

$$C(os) = row(os) | row(os') = row(os)$$

The table must be constructed in such a way that it describes a FSM: that is, it must be

- **consistent**: $\forall os_1, os_2 \in OS, [C(os_1) = C(os_2) \implies \forall o \in O, C(os_1 o) = C(os_2 o)]$.
- **closed** : $\forall os \in OS.O, \exists os' \in OS, os \in OS, C(os) \in C(os')$

From such a consistent and closed table a deterministic FSM can be described. Specifically, US-L* marks entries in the table as either *hole* entries or *permanent* entries. The former are those which can be reassigned as the algorithm tries to re-adjust the table for consistency. Only when no hole entries can be reassigned is a new test added to the table. Permanent entries correspond to a fixed action.

The algorithm proceeds by:

- Take a set of observation strings
- Initialise the table so that all the prefixes of the observation strings have an associated row in the table, and there is just one column with the empty string.
- Fill in the table entries using the observations, marking entries as *hole* entries if they are not supported by previous examples or *permanent* entries if they are supported by previous examples. In order to bound the size of the automaton, we specify a maximum number of times a *hole* entry can be changed, basing the maximum on domain knowledge if it is available: the maximum should depend on the dynamism in the system (since an entry will change if the system is changing) and on the uncertainty in the system. In our work, we may adjust the maximum over time using learned domain knowledge.
- Adjust the table to make it consistent, adding new columns to the table where necessary (adding a new column enables the separation of one equivalence class into two—this adds at least one new state to the corresponding automaton).
- Adjust the table to close it, adding new rows where necessary.
- Take the next set of observation strings and loop as appropriate

This algorithm is designed to be used as an online algorithm for an adaptive agent to learn models of opponent behaviour, although Carmel and Markovitch only apply it to repeated two-player games. We will be investigating its application in our domain, specifying in advance a maximum size for the automata. Now, in order to make use of these finite state machine models of agent behaviour,

our agent (maintaining these models) must be able to find an optimal response to what it believes to be the current situation. Referring back to our generic Bayesian model, this means evaluating $Q(b, a)$ for a belief state b which includes beliefs over finite state machines.

3.4.3.3 An online learning algorithm

This brings us to a complete description of our algorithm, which brings together several of the techniques described above. This is an algorithm implemented by a single agent that is aiming to adaptively find a best response to the behaviour of the other agents in the system. Our intent is that when all agents are implementing this algorithm, adapting to each other, they should converge on a "good" solution for the problem. This algorithm, as described below, maintains models of the other agents in the form of finite state machines. These models are held in a belief state which is updated using Bayesian learning. At each step, the agent computes an approximate best response to the current models.

- An agent maintains a current belief state, $b(X)$, with beliefs over the variables $X = (s, \{o, F, n\})$ where s is the current state, and $\{o, F, n\}$ describes a set of triples: in each triple, o is an observation history and (F, n) are the induced FSM and current node in the FSM. The belief state contains one such triple for each other agent in the system. The agent also maintains historical information about $b(s)$ over a fixed number of steps.
- Several parameters are fixed initially: F_{max} the maximum number of nodes in any FSM, γ the myopia of the agent, n_t the horizon length to use in computing an approximate best response and o_l the observation window length. n_t may be determined based on γ: roughly, for a state n steps into the future, s_n contributes $\gamma^n.r(s_n)$ towards the discounted future reward. Thus with $\gamma = 0.8$ (a common myopia value), after 10 steps less than 10% of the reward will be contributing towards the estimates of the future reward. This may be a small enough value to ignore. If γ is increased to 0.9, then

it will take 21 steps before the fraction of the reward under consideration is reduced below 10%.

- **initialise:**

  The belief state is initialised: $b(s)$ is initialised either to uniform beliefs or biased based on domain knowledge. The observation strings o are all empty, and the F have a single node with uniform probabilities over all actions[4]

- **at each step:**

  - The agent observes the actions of the others and makes observations about the state: these observations are used to update $b(s)$ using Bayes' rule.
  - The observation samples o are extended into the current time frame to obtain o', reweighting as appropriate. This is achieved by sampling from the expected observations of the other agents, given the current observation samples and $b(s)$. When the length of an observation string exceeds o_l, the earliest observations are dropped. If a sample's likelihood falls below probability threshold p_s, the sample is discarded, and a new string sampled using $b(s)$ and the stored history of $b(s)$ over o_l previous steps.
  - For each observation sample o', update the FSMs F associated with the sample with the new information in o' using US-L*. The weighting given to the FSM F is the probability of the associated observation sample.
  - For each sample FSM, compute an approximate best response, and thus decide the maximum likelihood best response action a from the FSM weightings as described previously.
  - Perform the action a
  - Repeat

[4] It would be possible to initialise with a more sophisticated set of F corresponding to shared conventions relating to the domain, for example encapsulating the knowledge that agents will run from a burning building. We leave that possibility to future work.

To reduce computational requirements, rather than doing all of this every step, we may prefer to collect behavioural samples over several steps and update our model less frequently. The best response is still computed every step.

In the final part of this section, we identify some other efficiency improvements which can be used in all models.

3.4.4 Efficient sampling techniques

In addition to the techniques described above which we have adapted for use in our model, we propose to incorporate some standard techniques to improve the efficiency of the algorithm. Although these techniques are not novel, we outline them in this section and explain how they are applied within the new context of our Bayesian POMDP model. Firstly, we explain the use of sparse priors to reduce the necessary calculations, particularly when projecting forward several states (section 3.4.4.1). Then, we describe two ways to reduce the amount of sampling necessary when sampling from a probability distribution: first by assuming that a point at time t will be similar to that point at time $t + 1$ (section 3.4.4.2), and secondly by focusing on sampling from regions of the space which are of particular interest to us (section 3.4.4.3).

3.4.4.1 Sparse priors

We incorporate an idea due to (Dearden et al., 1999). Typically, even if we do not know the transition function or the agent behaviours, we will have some idea that specific transitions or behaviours are infeasible. However, we may not wish to rule these transitions out completely. We can use a two-layered prior to describe this situation (Dearden et al., 1999) (Friedman & Singer, 1999). With the two layered prior, we (effectively):

1. Compute the probabilities as though the unlikely transitions were known to have no probability

2. Apply a scaling factor to all positive probabilities, and then distribute the remaining probability among the infeasible outcomes

In more detail, for the fully observable case (due to (Friedman & Singer, 1999)),

- Letting O be the observation space, of size L, define a random variable X that takes values from the power set of O, $x \in X = (o_1, o_2, \ldots)$.
- If $D = D_1 \ldots D_k$ is our Dirichlet prior, we intend that $D_i > 0 \iff i \in X$
- Now, for multinomial functions with Dirichlet priors,

$$P(D|X) \propto \prod_{i \in X} D_i^{\alpha - 1}$$

$$(\sum_i D_i = 1 \text{ and } \forall i \notin X, D_i = 0)$$

  and, for $i \in X$:

$$P(o_{k+1} = i | o_k \ldots o_1, X) = \frac{\alpha + n_i}{|X|\alpha + n}$$

- However, we may not be certain about what the set of feasible outcomes, X, should be. Let S_X be a variable describing the size of X, with some prior distribution $P(S_X = m)$. Then initialise all sets of the same cardinality with the same probability,

$$P(X|S_X = m) = \begin{pmatrix} L \\ m \end{pmatrix}^{-1}$$

- Then,

$$P(o_{k+1}|o) = \begin{cases} \frac{\alpha+n_i}{m^0\alpha+n} C(X, L) & \text{if } i \in O^0 \\ \frac{1}{n-m^0}(1 - C(X, L)) & \text{if } i \notin O^0 \end{cases}$$

  in which

$$C(X, L) = \sum_{m=m^0}^{L} \frac{m^0\alpha + n}{m\alpha + n} P(m|X)$$

- Finally, we use some kind of smoothing when sampling.

Specifically, in our work, D may be a parameter vector describing $P(s'|s,a)$, with D_i corresponding to $s' = s_i$, or it may describe $P(a|s)$ with D_i corresponding to $a = a_i$. Below, we also write $P(s, a \to s')$ and $P(s \to a)$, which we find intuitive for our system, in which we think of the state giving rise to the action, or the action causing the transition. (This is not the same as the logical $s \implies a$ arrow and is not associated with implication).

However, we are investigating scenarios in which *obs* does not directly include observing a, so that we cannot do the simple counting of occurrences described there. Instead, for our work, we must adapt the algorithm to sum over all possible counts, taking the likelihood of each count given previous data.

In more detail, letting *hidden* refer to either $(s, a \to s')$ or $(s \to a)$ and o refer to $(s \to s')$, we can say

$$P(o_{k+1} = i|o) = \sum_{hidden} P(o_{k+1}|hidden)P(hidden|o)$$

$$= \sum_j P(n_i = j|o) * \begin{cases} \frac{\alpha+j}{m^0\alpha+n} C(X,L) & \text{if } i \in O^0 \\ \frac{1}{n-m^0}(1 - C(X,L)) & \text{if } i \notin O^0 \end{cases}$$

Therefore, $P(n_i = j|o)$ simply refers to the appropriate multinomial induced by the observation o.

Now, although the use of sparse priors can simplify our transition updates considerably, we still need to sample when estimating integrals. The next two sections explain the techniques of weighted sampling and sampling with repair which make this sampling more efficient.

3.4.4.2 Weighted sampling

With the myopic-Q approximation, we have a recursive definition for the standard RL Q-value. If we are estimating the integral using sampling, we must recompute this definition for every sample we take, at every step. This can lead to a bottleneck in the computation.

However, if we already know Q for a given model θ, there is no need to recompute it in future steps. Rather, we can use the same set of samples for each step. When sampling, we associate a weight w with each sample value, corresponding to the probability of that value given the model. When the model changes, we can keep the same samples but adjusting these weightings according to the current distribution (as described in (Dearden et al., 1999)).

In more detail, suppose we have estimated $P(M|obs)$ for some model M and observations obs, but in fact have observations obs'. Then we adjust the weights associated with obs as

$$w_{obs'} = w_{obs}\frac{P(M|obs')}{P(M|obs)}$$

Specifically, suppose that $obs = obs_1 \dots obs_k$ and $obs' = obs_1 \dots obs_k \circ obs_{k+1}$. Then

$$\begin{aligned} w_{obs'} &= w_{obs}\frac{P(M|obs_1 \dots obs_{k+1})}{P(M|obs_1 \dots obs_k)} \\ &= w_{obs}\frac{P(obs_{k+1}|M)}{P(obs_{k+1}|obs)} \end{aligned}$$

However, as the distributions update, the selected samples become less representative of the update distributions. Thus it makes sense that every so often to re-sample from the new distributions and hence to compute new Q-values. We will define "every so often" using a heuristic based on the difference between the new and the old values. If the values are very different, then our model has changed significantly since we last sampled, and so we suppose that we should sample more frequently; if the values are similar, we can sample less frequently. There are several possibilities for selecting a heuristic corresponding to this insight. After some experimentation we have found selecting the number of steps between two samples using $\log \frac{1}{scaled\_difference}$ to be satisfactory, where $scaled\_difference$ refers to the difference between the new and old values, scaled according to the current expected reward. Other possibilities we tried included taking various powers of the difference, not using the log, and using multiples of the inverted difference.

Now, weighted updates reduce the frequency with which it is necessary to sample a particular value, using our beliefs about how much that value changes temporally. In the next section, we consider a technique which both reduces the frequency with which we sample particular values, using our beliefs about how interesting those values are, and reduces the frequency with which it is necessary to sample other values, using our beliefs about how the value compares with "nearby" values.

3.4.4.3 Sampling with repair

In more detail, consider that each time we want to find the Q-values for an MDP, we have to compute the complete Q-table (because the computation is recursive). We make two points:

- In practice, there may be many regions of the table that are not of much interest to us and which do not have much effect on the regions of interest. It is therefore not necessary to compute these parts of the table accurately.
- If two MDPs are fairly similar, then their Q-tables will also be similar. In some cases, regions may be almost identical. Consequently, if we have two MDPs built on estimated dynamics θ, and the estimates are similar, then these MDPs will be similar.

This leads us to the idea (Dearden et al., 1999) of patching up the old Q-tables rather than recomputing them from scratch every time we resample. The prioritised sweeping technique used in standard reinforcement learning techniques (Sutton & Barto, 1998) can be used for this. Prioritised sweeping is a variation on the iterative solution method for MDPs. Consider:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s,a \rightarrow s')[r(s') + \gamma V_k(s')$$

(with $V_k(s) = \max_a Q_k(s,a)$). A naïve iterative solution technique loops, performing this computation over each state s in turn until the change between Q_k and Q_{k+1} is sufficiently small for all k. When there are a lot of states, it may not be appropriate to compute accurate Q-values for every state, especially if many of them are rarely reached. Prioritised sweeping aims to estimate which states are most interesting, and update those most often.

For us, there are two points of interest: firstly, which Q-values have actually been changed by the change in the MDP, and secondly, which states are currently of interest to us. We propose to begin the patching up from the current state, which we can generally assume will be of interest.

3.5 Summary

In this chapter we have outlined a theoretical model for the online solution of partially observable multi-agent systems, based on the POMDP model. Our model generalises a number of existing models (Dearden et al., 1999) (Chalkiadakis & Boutilier, 2003) (Ross, Chaib-draa, & Pineau, 2008) (Emery-Montemerlo et al., 2004). We have shown how to make use of Bayesian networks to aid visualisation of various partially observable multi-agent systems, demonstrating their use as an aid to finding factorisations of probability distributions in this context; in this chapter we have not made use of the Bayesian networks beyond this visualisation tool, however in section 8.3.4 we show that formulating our problem with graphical models of this kind can allow us to lever powerful approximation tools for calculating probability distributions.

We have then identified three approximation techniques which we can add to this Bayesian POMDP model, and explained how we extend these techniques into our model. The first technique we identified was finite horizon best response, a standard technique which we extended to calculate best responses over belief states in a similar fashion to (Marecki et al., 2008). The second approximation technique described is state abstraction using statistical clustering. In this vein, we have (1) adapted this technique to perform efficient merging and splitting, (2) extended it to learning scenarios, (3) extended it to cluster observations in partially observable systems with finite state machine policies. Further, we suggest a technique for efficient implementation of this clustering method. The final approximation technique was policy abstraction using finite state machines. Based on the offline algorithm of (Marecki et al., 2008), we have provided the first online algorithm incorporating the learning of finite state machines in partially observable scenarios, and described in detail how to learn compact FSMs in dynamic online situations.

Finally, we have outlined three techniques due to (Dearden et al., 1999) which improve the efficiency of sampling from probability distributions when calculating belief updates or projecting forward during the best response calculation, and detailed the adjustments which we have designed in order to apply the techniques in our partially observable state model.

The sum of these contributions is to give a theoretical model for the online solution of partially observable multi-agent systems and then show how we can approximate this model in order to apply it to real-world problems. In order to demonstrate the effectiveness of this model, we have implemented three specific subcases. In the next chapter (chapter 4) we outline the problem we will use to test our models, before going on in chapters 5, 6 and 7 to describe our results and how they compare with the state of the art.

Chapter 4

The ambulance rescue problem

In order to test the algorithms outlined in the previous chapter on a challenging problem from the disaster response domain, we implemented a rescue scenario involving coordinating ambulances. Specifically, we consider the scenario from Robocup Rescue[1] in which there has been an earthquake in a region, causing civilians who were in that region to be hurt and buried under rubble. Ambulance teams enter the region and must coordinate to find and dig out the victims before they die, taking into account the depth at which victims are buried (more deeply buried victims will need more digging out) and the extent to which the victims have been hurt (more badly hurt victims will die sooner if they are not rescued).

In the body of this chapter (section 4.1), we specify a simplified version of this scenario as a multi-agent POMDP in the form of 3.2 and explain how we simplify the observation space. We also explain some variations on the problem which we used to investigate specific cases of our algorithm for which the initial formulation problem was not appropriate.

[1] `http://www.robocuprescue.org`

4.1 Model instantiation

In more detail, in the rescue problem we have an n by m gridworld. k agents can move left, right, up or down (constrained, of course, at the edges of the grid), or they can dig in their current location. In the gridworld are buried victims, described by two parameters: D and R. D ('deadness') is a measure of the proximity of the victim to death. When it reaches a maximum level the victim is dead and subsequently ignored for the purposes of the rescue problem. R ('rescue needed') is a measure of the depth at which the victim is believed to be buried. Agents digging can reduce R. If R reaches 0 before the victim dies, then the victim is assumed to be safe. The urgency of the victim therefore increases with increased D and with increased R, unless R is sufficiently large compared with D that the victim can be considered a lost cause. Figure 4.1 shows an example timestep on a 4x4 grid with three agents. In the figure, the state at the current time t is described by the 16 $< D, R >$ pairs and by the square numbers corresponding to the three agents, $a0, a1, a2$. The agent actions at time t are:

a0: Dig, **a1**: Move left, **a2**: Move right

After the actions have been carried out, the grid can be updated to show a new state, adjusting the agent locations and the $< D, R >$ values. The observations of individual agents are not shown on this grid. Keeping this example in mind, we now describe more precisely the instantiation of the model in section 3.2:

Agents: We assume that the number of agents, k, is fixed throughout each problem (although we will modify this assumption in chapter 7) and known to each agent. In figure 4.1, the set of agents is

$$\{a0, a1, a2\}$$

States: A state of this world is described by using a pair of variables $< D, R >$ for each of the grid squares, characterising the D and R values in the square (we

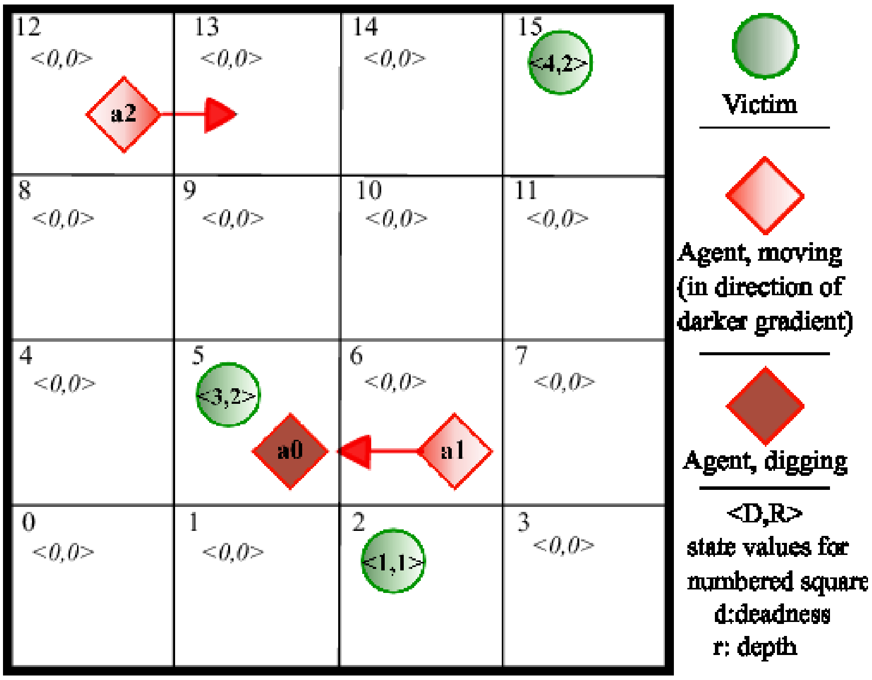


FIGURE 4.1: One step of the rescue problem on a 4x4 grid with three agents

make the simplifying assumption that there can be at most one victim in the square), and a variable for each agent, identifying its current square. We use l_d and l_r discrete levels to describe D and R, so for each square there are l_dl_r possible states, and for each agent there are mn possible states. This means that the state can be described by a total of $mn + k$ characters, where the first mn characters have l_dl_r possible values and the last k characters have mn possible values, making a total of $(l_dl_r)^{mn} * (mn)^k$ possible states. Thus, the number of states is exponential in the size of the grid and in the number of agents. In the example in figure 4.1, there are $16^{16} * 16^3 = 7.55578637 \times 10^{22}$ possible states, and the current state is

$$[0 :< 0, 0 >, 1 :< 0, 0 >, 2 :< 0, 0 >, 3 :< 0, 0 >, 4 :< 0, 0 >, 5 :< 3, 2 > \ldots$$
$$\ldots a0 : 5, a1 : 6, a2 : 12]$$

Locations: The location variable for each agent is its current square. Thus, for our example, the subset of the state describing the locations is $L = [a0 : 5, a1 : 6, a2 : 12]$.

Actions: Agents may take Move actions (left, right, up or down), or Dig actions in their current square. This results in five possible actions per agent (we do not admit “null” actions):

$$A = \{\text{Dig}, \text{Move left}, \text{Move right}, \text{Move up}, \text{Move down}\}$$

Consequently, there are k^5 joint actions: 125 joint actions in the example. Assuming the agent ordering $[a0, a1, a2]$, the immediate joint action is $\mathbf{a} = [\text{Dig}, \text{Move left}, \text{Move right}]$.

Observations: An agent observes some subset of the state variables, so there is one observation variable for each state variable. The values taken on by observation variables are those of the corresponding state variable, plus “null”, when no observation has been made. Consequently, there are $((l_d + 1)(l_r + 1))^{(mn)} * (mn + 1)^k$ possible observations, $25^{16} * 3^{17} = 1.14389695 \times 10^{26}$ in our 4x4 example.

Transition function: We can consider each of the independent state variables in turn.

- **Agent location** Each agent’s location depends only on its own action, and only on its previous location: $P(L_{i,t+1} = x|\mathbf{a}, s_t) = P(L_{i,t+1} = x|a_i \in \mathbf{a}, L_{i,t})$. In this problem, Move actions are deterministic, and move the agent one square in the requested direction. If this is impossible because the agent is at the edge of the grid, the action has no effect. Dig actions leave the location unchanged.
- **Deadness in a square with a victim** Each square j transitions (D_j, R_j) independently of other squares, so it is sufficient to define the transition function for one square. We use a global probability, p_d, to specify the probability of D increasing: this is a constant probability independent of the action: $P(D_{j,t+1} = x + 1|D_{j,t} = x) = p_d$.
- **Depth in a square with a victim** The R level reduces only if there is a Dig action. We assume that if there are $n_d(j)$ digs in square j in

the joint action, they are carried out one after another. n_d is a vector function of the state and the joint action (the action specifies which, if any, of the agents are digging, and the state specifies which square these are agents are in). With each of the $n_d(j)$ digs, the square depth (R_j) is reduced by 1 with probability p_r, with a minimum R_j value of 0.

$$\begin{aligned}
&P(R_{j,t+1} = x - 1|R_{j,t} = x, n_d(j) = 1) = p_r\\
&P(R_{j,t+1} = x - x'|R_{j,t} = x, n_d(j) = r) =\\
&\qquad (1 - p_r) * P(R_{j,t+1} = x - x'|R_{j,t} = x, n_d(j) = r - 1)\\
&\qquad + p_r * P(R_{j,t+1} = x - x' + 1|R_{j,t} = x, n_d(j) = r - 1) \text{ (where } n_d(j) > 1)
\end{aligned}$$

- **Deadness and depth when an agent dies or is rescued** After the joint action has been applied in a square j, we carry out a "tidying up" operation on the (D_j, R_j) settings. If the square's R_j value has reached zero, then it is assumed that a victim has been rescued from the square. This victim is no longer of interest to us and D_j and R_j are reset to 0. Otherwise, if the square's D_j value has exceeded the maximum health level then it is assumed a victim has died in the square. This victim is again no longer of interest to us and D_j and R_j are reset to 0. This means that the equations in the above two items must be adjusted slightly. Let $P(reset_{j,t})$ be the probability that square j is reset during this "tidying" phase at time t, with

$$P(reset_{j,t+1}) = P(R_{j,t+1} = 0) + P(D_{j,t+1} > l_d)$$

then

$$\begin{aligned}
&P(D_{j,t+1} = x) = (1 - P(reset_{j,t+1}))P(D_{j,t+1} = x) \quad \text{where } x \neq l_d\\
&P(D_{j,t+1} = 0) = P(reset_{j,t+1})\\
&P(R_{j,t+1} = x) = (1 - P(reset_{j,t+1}))P(R_{j,t+1} = x) \quad \text{where } x \neq 0\\
&P(R_{j,t+1} = 0) = P(reset_{j,t+1})
\end{aligned}$$

- **Deadness and depth in an empty square** Finally, if a square j is empty at the beginning of the time step, we use a further parameter, p_a, to define the probability that a victim will appear in that square. If a victim does appear, the (D_j, R_j) levels it has are determined with uniform probability (greater than 0) .We define a temporary binary variable, a_j, to determine whether or not a new victim appears in the square j. Then,

$$P(D_{j,t+1} = x | D_{j,t} = R_{j,t} = 0, a_j = 1) = \tfrac{1}{(l_d - 1)} \qquad \text{(Where } x = 1, 2, \ldots l_d)$$
$$P(R_{j,t+1} = x | D_{j,t} = R_{j,t} = 0, a_j = 1) = \tfrac{1}{(l_r - 1)} \qquad \text{(Where } x = 1, 2, \ldots l_r)$$
$$P(D_{j,t+1} = 0 | D_{j,t} = R_{j,t} = 0, a_j = 0) = P(R_{j,t+1} = 0 | D_{j,t} = R_{j,t} = 0, a_j = 0) = 1$$

We can apply these functions to the example in figure 4.1, with victims in three squares, $[2 :< 1, 1 >, 5 :< 3, 2 >, 15 :< 4, 1 >]$ and the joint action $[a0 : \text{Dig}, a1 : \text{Move left}, a2 : \text{Move right}]$, as follows:

- **Agent locations** Agent $a0$ will remain in place: $P(L_{0,t+1} = 5) = 1$. Agents $a1$ and $a2$ will each move one square: $P(L_{1,t+1} = 5) = 1$, $P(L_{2,t+1} = 13) = 1$.

- **Deadness in squares with victims** Squares 2, 5 and 15 contain victims. Note that if the deadness in square 15, D_{15}, increases, both D_{15} and R_{15} will be set to 0. Also, if the depth in square 2 decreases (impossible as there is no agent digging there) then D_2 and R_2 will be set to 0. (We do not show $P(reset_j)$ for the squares where it neither D_j nor R_j is one step away from resetting, making $P(reset_j)$ trivially zero):

$$P(D_{2,t+1} = 2) = (1 - P(reset_{2,t+1}))p_d, \quad P(D_{2,t+1} = 1) = (1 - P(reset_{2,t+1}))(1 - p_d)$$
$$P(D_{5,t+1} = 3) = p_d, \quad P(D_{5,t+1} = 2) = (1 - p_d)$$
$$P(D_{15,t+1}) = 0) = P(D_{15,t+1} = 5) = p_d, \quad P(D_{15} = 4) = (1 - p_d)$$

(where $P(reset_{2,t+1}|\mathbf{a}) = P(R_{2,t+1} = 0|\mathbf{a})$)

- **Depth in squares with victims** We extract the Dig information from the actions for each square: $n_d(2) = 0, n_d(5) = 1, n_d(15) = 0$. Then,

$P(R_{2,t+1} = 0|n_d(2) = 0) = 0,$ $\quad P(R_{2,t+1} = 1|n_d(2) = 0) = 1)$

$P(R_{5,t+1} = 2|n_d(5) = 1) = 1 - p_r,$ $\quad P(R_{5,t+1} = 1|n_d(5) = 1) = p_r$

$P(R_{15,t+1} = 0|n_d(2) = 0) = P(reset_{15,t+1}),$ $\quad P(R_{15,t+1} = 1|n_d(2) = 0) = 1 - P(reset_{15,t+1})$

(where $P(reset_{15,t+1}) = P(D_{15,t+1} = 0)$)

- **Deadness and depth in squares with no victims** All the remaining squares with state values $< 0, 0 >$ have the same functions. We define a temporary binary variable, a_j, to determine whether or not a new victim appears in the square j. Then, using our settings of $l_d = l_r = 4$,

$$\begin{array}{lll} P(a_{j,t+1} = 1) = p_a, & P(a_{t+1} = 0) = 1 - p_a & \\ P(R_{j,t+1} = 0|a_{j,t+1} = 0) = & P(D_{j,t+1} = 0|a_{j,t+1} = 0) & = 1 \\ P(R_{j,t+1} = x|a_{j,t+1} = 1) = & P(D_{j,t+1} = x|a_{j,t+1} = 1) & = 1/4 \quad \text{where } x = 1, 2, 3, 4 \end{array}$$

Observation function: Agents are able to see the squares (deadness, depth, and any other agents in the square) to the left, the right, above and below them, as well as their own square. Since all agent actions are fully observable, we assume that we can also observe all agent locations. We can consider this analogous to supposing that all the agents have radios, but no time to communicate more than their own position. Additionally, we define a problem-specific parameter, v, for the visibility. For every other square, the agent will be able to see the agent-deadness D in that square with probability v and the depth R in the square with independent probability v. This 'visibility' parameter could be justified as some level of communication with a centralised observer, say a helicopter viewing the scene. We assume no error in the observation: either a variable is completely and correctly observed or it is not observed at all. In section 8.3.5 we will explain how the model can be extended to permit error-prone observations.

Thus, in example 4.1, consider agent **a0**: he observes the positions of every other agent: $[a0 : 5, a1 : 6, a2 : 12]$. $a0$ also observes completely squares 1, 4, 5, 6, 9: $[1 :< 0, 0 >, 4 :< 0, 0 >, 5 :< 3, 2 >, 6 :< 0, 0 >, 9 :< 0, 0 >]$. For any other square j with values $< D_j, R_j >$, $a0$ observes $j :< null, null >$

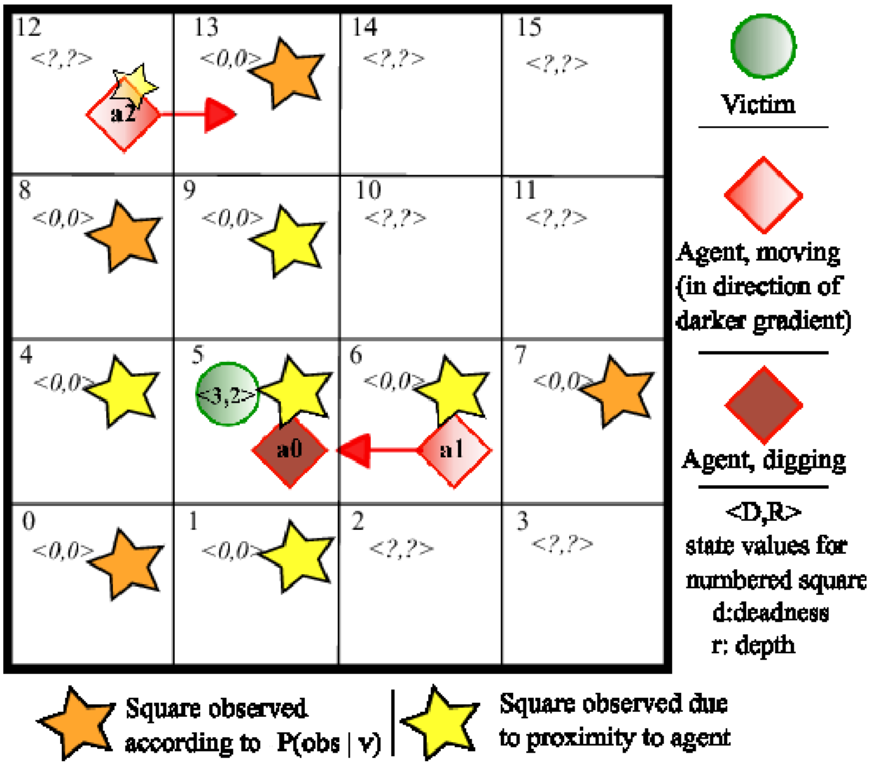

12
<?,?>
a2
13
<0,0>
14
<?,?>
15
<?,?>
8
<0,0>
9
<0,0>
10
<?,?>
11
<?,?>
4
<0,0>
5
<3,2>
a0
6
<0,0>
a1
7
<0,0>
0
<0,0>
1
<0,0>
2
<?,?>
3
<?,?>
Victim
Agent, moving (in direction of darker gradient)
Agent, digging
<D,R> state values for numbered square d:deadness r: depth
Square observed according to P(obs | v)
Square observed due to proximity to agent

FIGURE 4.2: One agent's view of the situation shown in figure 4.1

with probability $1 - v$ and $j :< D_j, R_j >$ with probability v. Similarly, $a1$ observes $[a0 : 5, a1 : 6, a2 : 12, 2 :< 1, 1 >, 5 :< 3, 2 >, 6 :< 0, 0 >, 7 :< 0, 0 >, 10 :< 0, 0 >]$ and all other squares with probability v. Finally, $a2$ observes $[a0 : 5, a1 : 6, a2 : 12, 8 :< 0, 0 >, 12 :< 0, 0 >, 13 :< 0, 0 >]$.

Figure 4.2 shows a possible view for agent $a0$ when $v = 0.3$. The yellow stars represent the squares which are observed completely, because they are near to the agent. The orange stars represent squares which on this occasion have become visible as a result of the global visibility parameter: note in particular that neither square 2 nor square 15 and their victims are visible to the agent. Finally, the small yellow star by agent $a2$ indicates that it is visible as all agent locations are visible.

Reward function: The reward function is a function of both the previous state and the current state. For each square, if a victim disappears because they have died, then the reward is decremented by one point. If a victim disappears because they have been saved, then there is no change to the reward. Consequently, for this problem rewards will always be less than or equal to 0.

In example 4.1, there are two victims which may change status between the timestep shown and the next timestep: the victim in square 2 may be rescued (although as we have shown this is actually impossible because there is no agent present), and the victim in square 15 may die. Of these, only the victim in square 15 can affect the reward. We have stated that this victim will die with probability p_d. If it does, then the reward for the timestep will be -1; otherwise, the reward will be zero.

The above definitions allow us to define beliefs over the values (D, R) of a square (and thus over the state, since locations are observable), and beliefs over the observations of other agents, given their locations:

Agent locations: We are certain for all squares how many rescue agents they contain / for all agents where they are located

The square is observed: We are certain of both its parameters

The square is not observed and has not been observed for t_i timesteps:

$$P(x_t = v_t | x_{t-i} = v_{t-i}) = \sum_{v} P(x_t = v_t | x_{t-1} = v) P(x_{t-1} = v | x_{t-1} = v_{t-i})$$

where the 1-timestep probabilities depend on p_d, p_r, p_a as appropriate, and the dig observations in that square.

The square has never been observed: This is just as above, but with $P(x_0 = v_0)$ set to the problem-specific prior probabilities. Here, we assume that all squares are empty to begin with.

Given these equations, and using its environment model, our agent can calculate probabilities (beliefs) for each of the state variables. In all of our experiments, we assume that the agent knows the environment model, which is the model described on the previous pages. These probabilities form our agent's belief state: that is, the beliefs about the D and R values of the squares and the locations of the other agents. Similarly, we must define our beliefs about the observations of the other agents. Just as our beliefs about the state of each square are multinomial, the other agents' beliefs about the state of the square will be multinomial. Therefore, in the full POMDP model, our beliefs about other agents' beliefs over the state of the square would take on corresponding Dirichlet distributions. However, we are not trying to maintain beliefs about the other agents' belief states, only about their observations. Now, our own beliefs about the state of the square define exactly what we believe other agents will see if they see that square, as the observation function is deterministic and consistent for all agents. Because we know the location of the agent, we know of the (up to) four surrounding squares it definitely sees. Finally, we know that there is a v probability it will see any other square.

Now, the problem described above is used as given for experiments involving partially observable states. However, we modify the problem slightly in order to investigate specific aspects of our general model. Specifically, problems where it is the actions rather than the states which are partially observed, which are investigated in chapter 5, and problems which are dynamic or open, which are investigated in chapter 7. In order to handle these cases, we make slight modifications to the problem.

Firstly, three modifications apply when actions are partially observable:

Removing agent locations from the state. Although the agent locations are a part of the state, in one subproblem we choose to treat them separately, making them only partially observable while the state remains full observable. Technically, this is a partially observable state problem. However, providing the actions are deterministic, deductions about the agent locations, depend only on the agents' choice of actions.

Uncertain actions. As described in example 10, we add some uncertainty to the outcome of an action: with probability mp, the intended action is carried out, while with probability $(1 - mp)/4$, a random action (selected from all five possibilities including the intended action) is carried out. That is, an agent intending to move (such as $a1$ and $a2$ in the example in figure 4.1) may find himself moving in the wrong direction, or shuffling on the spot and inadvertently digging. Similarly, an agent intending to dig (such as $a0$ in the example in figure 4.1) may unexpectedly slip into a nearby square without digging.

Penalising actions. In order to investigate how agents can use the reward function to make inferences about actions, we adapt the problem in such a way that the choice of action affects the reward directly. In this adapted problem, we give moves a cost of 0.1 points, representing the fact that each move uses up some of an agent's resources. We also penalise Digs which take place in an empty square, giving them a cost of 0.5 (Digs in a square with a victim incur no cost). These penalties are applied to the intended action, not the outcome, forming an analogy with the effort the agent must put into the action.

Consider the example in figure 4.1 in the light of such a penalising function, with a visibility parameter of 0: agents $a1$ and $a2$ will each incur a cost of 0.1, so that the total reward for the step is -0.2 if the victim in square 15 survives, and -1.2 if the victim in square 15 dies. Every agent will observe this reward and consequently deduce that two move actions were made (since no other combination of actions could cause this fractional reward) and one Dig action. $a0$ and $a1$ can observe each other and so will both know that $a0$ carried out the Dig action and therefore $a2$ made a move action (although not in which direction). $a2$ which cannot observe either of the other agents can update beliefs about the agents—for example, if $a2$ has observed square 5 recently and seen the victim there, then they may believe that an agent in square 5 is likely to dig. Furthermore, if $a2$ can see square 6, they will know that the agent in square 6 did not Dig, or they would have incurred the penalty.

> Now, the above discussion assumed that all actions were successful, but if the action penalties are combined with action uncertainties, then $a0$, who performed the Dig, will know that $a1$ and $a2$ both intended move actions, and will know whether $a1$ achieved a move. However, $a1$ will no longer be certain whether $a0$ intended a Dig or a Move. Consequently, $a1$ will have to update his behaviour models for each agent assuming that $a0$ intended a Dig with probability mp and a Move with probability $(1-mp)$, and vice versa for $a2$. In general, agents will not be able to make such precise deductions about the intended actions of others and will have to apply similar probabilistic rules for many of the others when learning about their behaviour.

Secondly, as well as partially observable actions, we will be investigating our model in the context of dynamic environments, and open environments. In the problem described above and depicted in figure 4.1, we have not explained how such environments are included.

In more detail, **dynamism** occurs when the environment changes during the course of the problem. In the ambulance problem, the change may be to the value of any one of the parameters. Here, however, we will investigate dynamism in the arrival rate and death rate parameters; and in the move penalty value. **Openness** refers to agents appearing or disappearing during the course of the problem. In chapter 7 where we experiment in such environments, we explain how to implement dynamism and openness without changing the problem structure. To investigate dynamism, we will change problem parameters such as p_r or p_a at a timestep t. We assume the other agents know about all changes. The belief state at time t forms the prior for the belief state at time $t+1$, but the belief updates at $t+1$ use the new parameters. No further changes are necessary. To investigate openness, we will introduce or remove agents after some number of steps. Again, all agents in the system know about the changes instantly. When new agents are added to the system, they will be placed in the same initial location at square 0. New state and observation variables must be added describing the locations of every new agent. When agents are removed from the system, the corresponding state and observation variables must be removed from the state.

4.2 Summary

In this chapter we have introduced a scenario from the disaster response domain and shown in detail how to describe this problem in terms of the model in section 3.2. We have illustrated the description with a specific example. Finally, we have explained how to adjust the problem in order to experiment on some specific variants of our model. In the following chapters, we use the problem we have described to evaluate our model, beginning with the simplest case: the case in which the state is fully observed, so that no inferences about other agents' beliefs are necessary, but actions are not fully observed.

Chapter 5

Coordination in the presence of partially observable actions

In this chapter, we evaluate the model of chapter 3 for the specific case in which the state and the transition function are known, but the actions of other agents are only partially observable and their behaviour policies are unknown. By so doing, we substantiate the claim in section 1.4 that we are the first to consider explicit models of the other agents in a partially observable Bayesian environment. We also show that doing so is better than a handwritten state-of-the-art strategy for the same problem.

In more detail, the first part of the chapter (section 5.1) explains how the model in chapter 3 specialises to this case, detailing the update and best response equations. The second and third parts of the chapter evaluate the model on the ambulance problem of chapter 4, first for the problem in which all agents share a global reward based on the number of lives saved (section 5.2), and second on the problem variant in which the global reward includes an aggregated cost of the agents' moves (section 5.3). The first, simpler case will substantiate our claim, showing that explicitly considering the other agents results in a successful strategy. The second allows us to investigate our model in more depth, substantiating the assertion in section 2.2.2 that such model-based learning algorithms provide flexibility over model-free learning algorithms—model-free algorithms would not be able to make inferences from the reward at all.

5.1 Evaluating the general model with partially observable actions

We consider the case where an agent knows the environmental dynamics (the transition and reward functions), and is able to observe the state, but cannot see the actions of every other agent. He may be able to see the actions of some agents locally (as in the Tamptono example 1 when two ambulances pass one another or arrive at the same location). For the other agents in the system, he will have to make deductions about the actions based on the state changes (for example, in Tamptono, when the agent hears that the body count has increased, he may assume that one of the other agents has been searching). To keep the model straightforward, we do not consider the case where the agent is able to make direct observations about the actions of those he can't see (such as the flying rubble in example 13). Adding this case would not change our model substantially, but would add an extra layer of evaluation complexity.

In terms of the model described in section 3.2—specifically, the nodes in the grand MDP diagram (figure 3.6), we assume that: s is observed, the a_j are not directly observed, θ is known and the π_j are not known (although the number of agents is known). Figure 5.1 shows the Bayes network for this case.

The best response for this network is calculated using the same Q-calculation as before (equation 2.6 in section 2.3):

$$Q(a_i, b) = \sum_{\mathbf{a_{-i}}} P(\mathbf{a_{-i}}|b) \sum_{s'} P(s'|a_i \circ \mathbf{a_{-i}}, b) \sum_{r} P(r|s', a_i \circ \mathbf{a_{-i}}, b)[r + \gamma V(b < s, \mathbf{a}, r, s' >)] \tag{5.1}$$

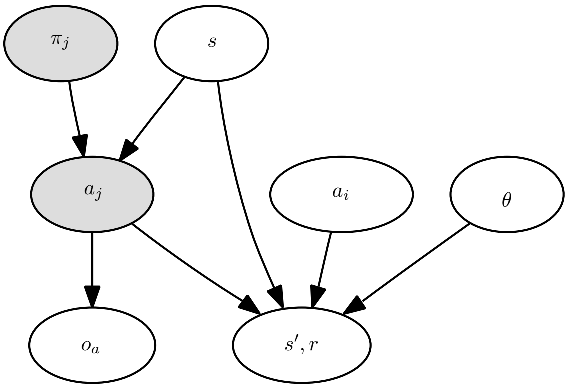


FIGURE 5.1: Partially observable actions: Bayes network

This is the basic Bellman equation, extended with the explicit joint action probabilities. $P(s|\cdot)$ and $P(r|\cdot)$ are taken straight from the transition and reward functions. $P(\mathbf{a_{-i}}|b)$ refers to the joint strategy of the other agents. Since the agents make their decisions independently, we can factorise this probability into

$$\Pi_{j\neq i}P(a_j|b)$$

where

$$P(a_j|b) = \int_{\pi_j} P(a_j|\pi_j)P(\pi_j|b)$$

Finally, we use figure 5.1 as described in section 3.3, to obtain the belief update equations:

$$P(\pi_j|obs) \propto P(\pi_j)\sum_{a_j} P(s', r|s, \theta, a_i \circ a_j)P(o_a|a_j)P(a_j|s, \pi_j)$$
$$P(a_j|obs) \propto P(s', r|s, \theta, a_i \circ a_j)P(o_a|a_j)\int_{\pi_j} P(a_j|s, \pi_j)P(\pi_j)$$

In the next section, we discuss how these various values can be computed.

5.1.1 Performing the updates

This section discusses the implementation of the equations given above. Now, in the models we have been considering, we assume that states and actions are all discrete[1]. Consequently, the distributions can be described using a multinomial distribution; the multinomial distribution has the Dirichlet distribution as its conjugate prior.

Specifically, letting **a** be a joint action, consider $P(\mathbf{a}|s)$[2] to be a weighted n-sided die (with a different die corresponding to each state s). A particular weighting is defined by a multinomial distribution with parameter vector $W = w_1 \dots w_n$. We aim to learn the weightings of the die, maintaining at all times the likelihood of every possible weighting W. The probability of a particular weighting W is described by an n-dimensional Dirichlet distribution with parameter vector $\alpha = \alpha_1 \dots \alpha_n$. After each throw of the die, the Dirichlet distribution is updated based on the observed outcome, using Bayes' rule:

$$P(W = w|obs) = zP(obs|W = w)P(W = w)$$

(z is a normalising constant), where $P(W = v)$ has distribution $Dir(\alpha)$. Letting n_i be the number of times i was observed in a sequence of observations $obs_1 \dots obs_k$,

$$\begin{aligned} P(W = v|obs_1 \dots obs_k) &\propto P(obs_1 \dots obs_k|W = v)P(W = v) \\ &= z_1 \prod_i v_i^{n_i} \ z_2 \prod_i v_i^{\alpha_i - 1} \\ &= z_1 z_2 \prod_i v_i^{n_i + \alpha_i - 1} \end{aligned}$$

[1] In general, problems with continuous states or actions could be approximated using discretization techniques (such as those discussed in section 2.4.1), and thus the same model used. However, depending on the specific continuous probability distributions controlling the transition functions, using the continuous distribution directly may be more appropriate. We do not discuss such an implementation in this thesis, although some pointers to recent work with learning in continuous environments are provided in section 8.3.5

[2] As before, we may write this as $P(s \rightarrow \mathbf{a})$.

i.e. the posterior probability of a particular weighting has a Dirichlet distribution with parameters $\alpha'_i = \alpha_i + n_i$ where n_i is the number of times i has been observed.

Similarly, we can consider $P(s'|s,\mathbf{a})$ to be a particular weighted die and estimate its weighting accordingly. In exactly the same way, we can use a multinomial distribution with Dirichlet priors and posteriors over the multinomial to learn the weightings on the dies.

Now, when we cannot observe specific actions but we know the weightings on the $P(s,\mathbf{a} \to s')$ die, and have some estimator for the distribution of $P(s \to \mathbf{a})$, we aim to gradually update the latter estimator through observation. Since $P(s \to \mathbf{a})$ and $P(s,\mathbf{a} \to s')$ remain multinomial, given Dirichlet priors we should still have Dirichlet posteriors. We return to the dice analogy to explore this in more depth. Consider again $P(s \to \mathbf{a})$. Rather than trying to estimate the die weightings from observing the faces, suppose that there is a second row of dice, each of which has an unknown weighting. The first roll $(s \to \mathbf{a})$ triggers one of the second dice to be rolled $(s,\mathbf{a} \to s')$, and it is the face on this (s') that we observe. From our knowledge of the weightings on the second row of dice, we try and determine which one was rolled, and hence which face the first die landed on.

In detail, let $\pi = \pi_1, \pi_2, \ldots$ be the weight vector for the multinomial distribution $P(s \to \mathbf{a})$ and T_f the weight vector for the multinomial $P(s,\mathbf{a} \to s')$. We observe the system transitioning from s to s' (or we observe the result of the second die roll), and from this observation $(s \to s')$ we try and infer the action a (or the first die roll), and the weight vector π determining this action choice (or die). We can apply Bayes' rule :

$$P(\pi|s \to s') \propto P(s \to s'|\pi)P(\pi) \tag{5.2}$$

$$P(\mathbf{a}|s \to s') \propto P(s \to s'|\mathbf{a})P(\mathbf{a}) \tag{5.3}$$

$$\begin{array}{lrcl} \text{where} & P(s \to s'|\pi) & = & \sum_a P(s \to \mathbf{a}|\pi)P(s,\mathbf{a} \to s') \\ \text{and} & P(\mathbf{a}) & = & \int_\pi P(s \to \mathbf{a}|\pi)P(\pi) \end{array}$$

This completes the update step, providing the agent with the belief states which it will use in evaluating equation 5.1 to decide the best response to the immediate state, given its beliefs about the other agents' strategies. We now turn to the evaluation of this equation.

5.1.2 Best response computation

The second part of the stepwise algorithm is the computation of the Q-values at each step. We compute the value $Q(a|b)$ for every single-agent action a, given our current belief state: the belief state describes our beliefs about the action weightings π, as well as holding our state and action observations.

Given a specific, known model, $\theta = (\pi, T_f)$, we can estimate the value of each action as:

$$\begin{aligned} Q(a|s,\theta) &= \sum_{a_j} P(a_j|\theta) Q(a \circ a_j|s,\theta) & (5.4) \\ &= \sum_{a_j} P(a_j|\theta) \sum_{s'} P(s'|a \circ a_j, s, \theta)[r(s') + \gamma V(s')] & (5.5) \end{aligned}$$

where a_j refer to the actions of the other participants in the system, and $V(s) = \max_a Q(s,a)$

For the Bayesian system, referring to equation 5.1, we modify this as:

$$\begin{aligned} Q(a|s,b) &= \sum_{a_j} P(a_j|b) Q(a \circ a_j|s,b) & (5.6) \\ &= \sum_{a_j} P(a_j|b) \sum_{s'} P(s'|a \circ a_j, s, b)[r(s') + \gamma V(b')] & (5.7) \end{aligned}$$

In which,

$$P(a_j|b) = \int_\pi P(a_j|s,\pi)P(\pi)$$
$$V(b') = \max_a Q(a|b') \text{ (note the recursion in this definition)}$$

where b' takes into account how the beliefs for θ would be updated if a_j and s' were observed. Now, neither $P(a_j|b)$ nor $Q(a|b')$ can generally be calculated precisely, the former because it is an infinite sum and the latter because it may project infinitely many steps into the future: with each step, the agent can reach a new belief state. Now, $P(a_j|b)$ can be estimated to any required accuracy by sampling, using the techniques described in section 3.4.4. Similarly, providing $\gamma < 1$, with each step projected into the future, the contribution of $Q(a|b')$ from that step to the current Q calculation is reduced. Consequently, we can estimate $Q(a|b)$ to the required accuracy using the finite horizon technique described in section 3.4.1.

In conclusion, considering the three-step algorithm in section 3.2 (algorithm 1), the agent i initialises its belief state with uniform beliefs over the other agents' strategies, and then at each timestep carries out the following three steps:

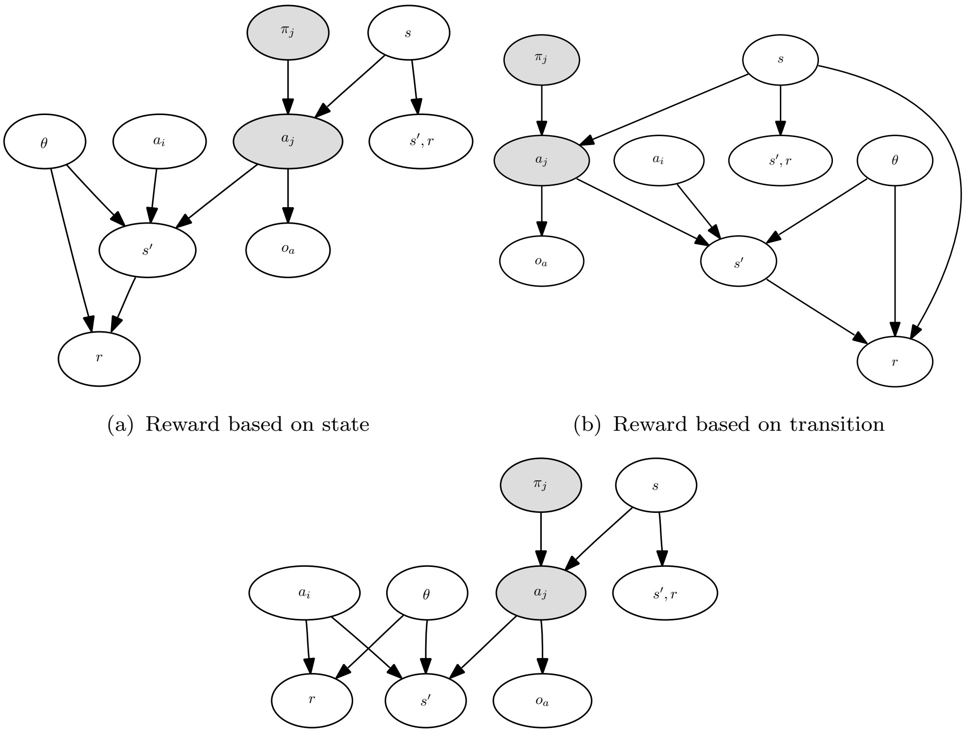
1. Observe: the state transition $s \rightarrow s'$ [3]
2. Update (z is a normalising constant):

$$P(\pi|s \rightarrow s') = z \sum_a P(s \rightarrow \mathbf{a}|\pi)P(\pi)$$
$$P(\mathbf{a}|s \rightarrow s') z P(s \rightarrow s'|\mathbf{a}) \int_\pi P(s \rightarrow \mathbf{a}|\pi)P(\pi)$$

3. For some k compute a k-step best response, $\max_a Q(a|b_k)$, using

$$Q(a|b_0) = 0$$
$$Q(a|b_k) = \sum_{\mathbf{a_{-i}}} P(\mathbf{a}|b_k) \sum_{s'} P(s'|\mathbf{a}, s, b_k)[r(s') + \gamma V(b_{k-1})$$

[3] This can also be seen as observing the current state s' while maintaining a history of one previous state.



(a) Reward based on state

(b) Reward based on transition

(c) Reward based on action

Figure 5.2: Bayesian network diagrams for different reward cases of the same transition function, in a scenario with observable states and partially observable actions

(where $V(b_{k-1}) = \max_a Q(a|b_{k-1})$)

Now, in this description, we have not mentioned the rewards. If the system is cooperative, so that all agents see the same rewards, then the rewards can be considered part of the global state. However, in some scenarios, an agent's individual reward may be unique to that agent—for example, in problems where moving vehicles are penalised as they use up fuel. In the next section we discuss such scenarios in more detail.

5.1.3 Exploiting reward structure

It is possible to think of rewards as being a consequence of:

- The immediate state: for example, the number of rescued civilians in a disaster (figure 5.2(a))

- The transition from state to state: for example, a reward may be assigned based on the change in the number of rescued or dead civilians in a disaster, rather than the total figure, so as to permit comparisons between disasters with different initial death tolls (figure 5.2(b)).
- The action performed by an agent: for example, a movement may be penalised as having a cost, or an agent may be rewarded for effort even if nothing comes of it (figure 5.2(c)). A common use for this style of rewards is to simplify the necessary agent inference. For example, an agent's need for fuel could be built into the first reward type by including the agent's fuel levels in the state, adjusting the transition function so that a "move" action failed if fuel levels were low, and providing a "refill" action available at certain "pump" locations. However, to maintain a simpler model we can penalise each of the agent's "move" actions a small amount, indicating to the agent that moving has some cost without requiring the agent to include a refuelling strategy. In this case, either each agent's actions may contribute to a global reward, or each agent may receive a distinct individual reward.

or of any combination of the above. In the problem described in the bulk of section 4.1, the reward is notionally based on the current state (number of bodies, number of survivors). However, in the implementation given, dead civilians and rescued civilians are "tidied away". Therefore, the implementation must assign its rewards based on the changes between two states. In the variant problem described at the end of section 4.1, some reward is assigned based on the changes between two states. Additionally, each agent's actions contribute to a globally observed reward.

In the next two sections, we will evaluate our algorithm on each of the two variants of the problem: firstly, we use the problem without actions contributing to rewards, which is both simpler and more intuitively related to the real scenario. This is the variant of the problem which will be investigated in later chapters. Secondly, we evaluate the algorithm using the variant of the problem with actions contributing to rewards, in order to demonstrate that by making use of the extra information which can be inferred about the others' actions, the agent is able to improve its own strategy, thus providing further insight into the Bayesian model.

5.2 Ambulance rescue with partially observable actions

Many disaster scenarios have the property that any one agent has limited visibility of the other agents, but still needs to coordinate with them. Here, we investigate the ambulance rescue problem described in chapter 4. In this instantiation of the ambulance rescue problem, the visibility parameter v is used to describe the visibility of the other agents: our agent can see any agents on neighbouring squares, and on average a fraction v of any more distant agents. Furthermore, agents have some uncertainty over *Move* actions as well as *Dig* actions—although an agent will move in the direction it intended with probability p_m, it may move in another direction or in no direction at all with probability $(1 - p_m)/4$. This means that even when we observe the locations of other agents, we cannot be certain about what action they have taken. We begin by describing our experimental setup in detail in section 5.2.1. Section 5.2.2 then gives our results.

5.2.1 Experimental setup

In particular, we pay attention to the following system parameters:

Sample size In our partially observable action model, it is necessary to estimate an integral (over all models for the strategies). We chose to do this by taking *sample_size* samples from possible strategies, and evaluating best responses over all possible combinations of these.

Number of agents We believe that the effects of using our explicitly multi-agent model over a single agent model should become more marked as the number of agents increases. However, for this problem the number of states, hence the computational complexity, increases exponentially with the number of agents, making it infeasible for us to test on many agents.

Move randomness We investigated different values for p_m between 0.1 and 1.0. When varying the sample size, we fixed p_m at 0.7, as a middle ground with some randomness but not so much that nothing can be learned.

Other parameters Finally, we set γ, the agents' myopia, to be 0.75. This means that they allocate only a quarter of the original importance to states five steps in the future, and by ten steps the contribution of new states is negligible.

Model parameters: Following experimentation, we fix the following parameters: $l_d = l_r = 5$, $p_d = 0.15$, $p_r = 0.4$, $p_a = 0.05$, $v = 1$. In particular, we felt that the choice of five health and burial levels was sufficient to make the problem interesting without making the state space too huge. The other parameters were selected to generate scenarios requiring cooperation: victims were not arriving so fast that simply digging out the nearest was appropriate, victims might require more than one agent for rescue, and victims could survive long enough to be reached by agents some distance away.

We vary m, n as specified. By default, we work with problems involving three agents on a 7x7 grid, as this provides a reasonably challenging medium-size problem. Finally, in the belief-state based algorithms, we must take samples from the belief state (in this problem, our beliefs about the likely actions of other agents). We define the *sample size* as the number of samples taken for each variable, initialising it at a size of 20 (for comparison, previous work on a single agent problem found that 20 samples was sufficient for good solutions (Dearden et al., 1999)).

Previous work on large dynamic rescue problems of a similar form (Paquet et al., 2005) compares with a handwritten strategy (`smart`) tailored to the problem, and we do the same thing. Our handwritten strategy is the strategy that was used by the AladdinRescue team for ambulance distribution in the Robocup Rescue competition, which inspired this problem. The algorithm uses a greedy strategy to allocate ambulances to victims and is optimal in scenarios where (1) no new victims are arriving and (2) visibility is perfect (Ramamritham, Stankovic, & Zhao, 1989). It is therefore not an optimal strategy for the problem as we have stated it, but is a good state of the art approximation.

In every experiment, we carried out several runs of the problem, varying the initial placement of civilians and randomising their arrival and visibility. The same random seed was used to initialise each of the test algorithms in each run. The error bars included in the results show the 95% confidence intervals around each point. The rest of this section discusses our key results.

5.2.2 Experimental evaluation

We begin this section with a discussion of several related algorithms, based on the best response algorithm described above, but faster to evaluate. Section 5.2.2.1 outlines these algorithms and section 5.2.2.2 evaluates them over different sampling rates. The rest of the section focuses on the `full best response` policy, evaluating the policy across a number of different parameters.

5.2.2.1 Alternative implementations

Initially, we implemented the "full" best response algorithm, evaluating the finite horizon best response as described in section 5.1.2 above. However, this full best response algorithm is considerably slower than the `smart` policy to evaluate, particularly as the number of agents increases. As we showed in chapter 4, the number of joint actions increases exponentially with the number of agents. Consequently, the Q-evaluation (equation 5.1) $Q(a_i, b) = \sum_{\mathbf{a_{-i}}} P(a_i|b) \cdot \ldots$ requires exponentially more computation to complete. We therefore investigated two variants on this best response algorithm which use approximations to reduce the number of actions considered.

- **Maximum likelihood (ml) best response**: In this variant of the best response algorithm, our agent does not project over all possible joint actions, but rather calculates the maximum likelihood action for each other agent and calculates the best response assuming that the other agents do perform the maximum likelihood action. Formally, the Q-calculation (equation 5.7) is replaced with

$$Q(a|b) = \max_{a_j} P(a_j|b)Q(a \circ a_j|s, b)$$

- **Sampling best response**: Already, rather than project forward over millions of states in the summation $\sum_s'$ in the Q-calculation (equation 5.7), we project forward over a sample of states, as described in section 3.4.4. In the "sampling best response" variant of the full best response, our agent also does not project over all possible joint actions, but rather samples from the possible joint actions to estimate the best response. Note that unlike sampling from the models, there is a relatively small number of possible joint actions ($5^{numagents}$). Although this means that it is easier to sample all or most of the actions, it may also mean that any samples taken are less likely to be generally representative.

Since these two algorithms are less thorough than the full best response, we do not expect them to be as good. However, they scale much better with the number of agents: `maximum likelihood best response` carries out one computation per agent at each step; `sampling best response` carries out *samples* computation per agent, while the `full best response` carries out $5^{numagents}$ computation at each step. Figure 5.4 demonstrates the time savings from the simpler algorithms[4]. Therefore, we investigated how much the simplified algorithms underperform the full algorithm in order to see if they might make feasible replacements for larger problems.

In more detail, figure 5.3 compares the three best response algorithms, and the `smart` policy, for an average over ten runs, with the parameters set as above and a

[4] These experiments, and all experiments described below, were run on a dual-core machine in the university's Beowulf cluster.

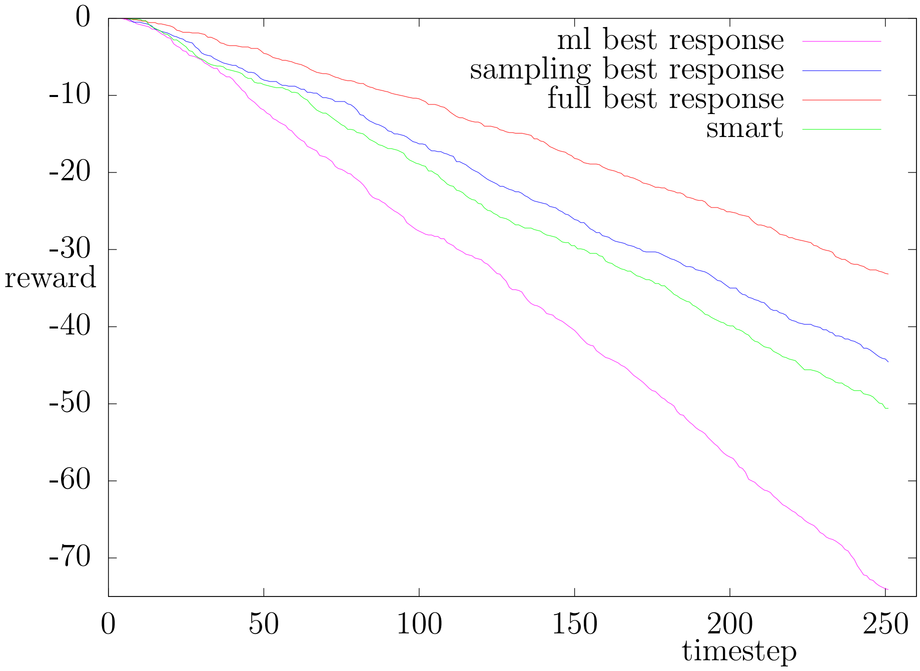
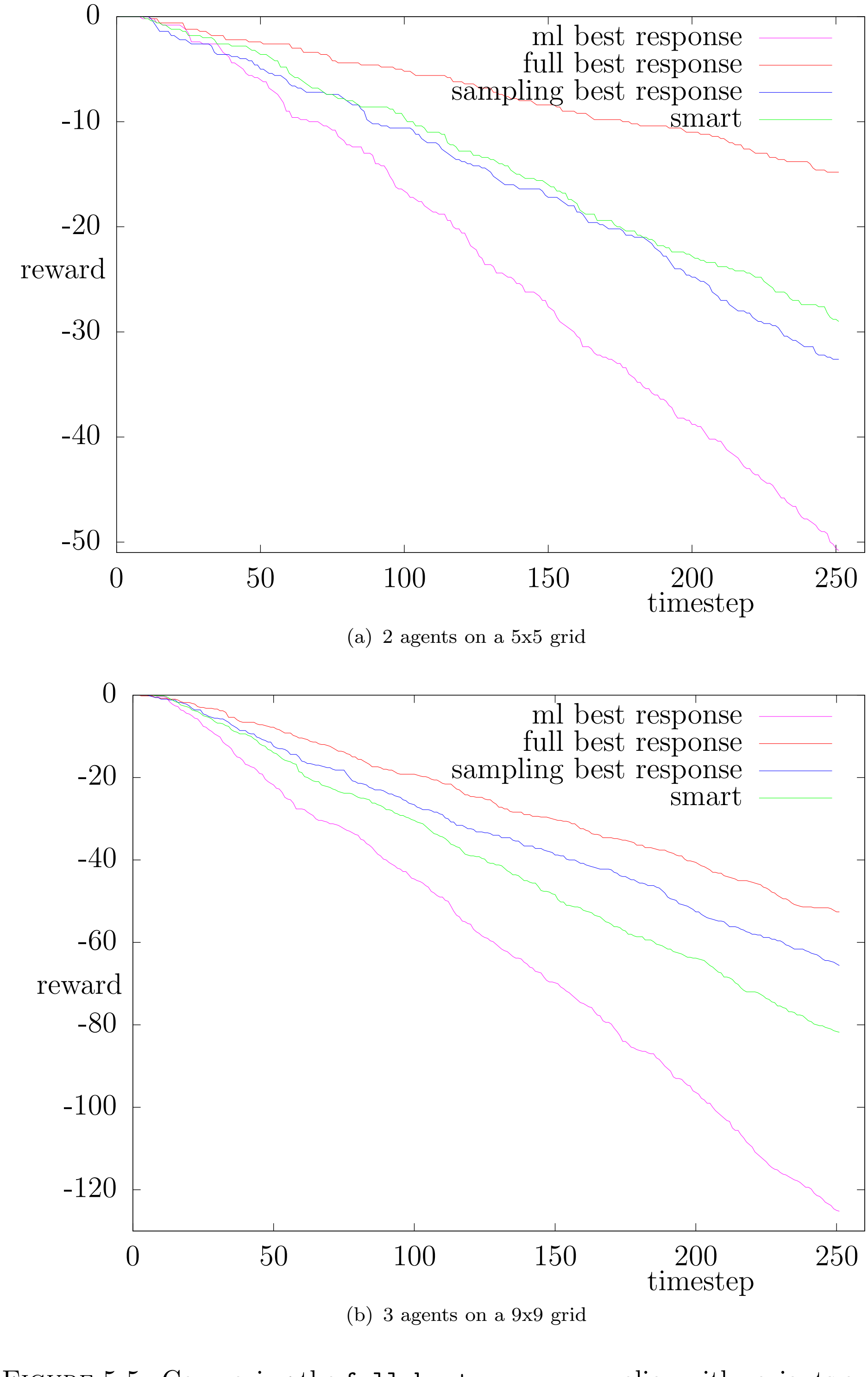


FIGURE 5.3: Comparing the `best response` policy with variants and the smart policy

7x7 grid, 3 agents	
Algorithm	Time
`max. likelihood best response`	11 seconds
`sampling best response`, sample rate 5	7 seconds
`sampling best response`, sample rate 120	105 seconds
`sampling best response`, sample rate 300	127 seconds
`full best response`, sample rate 5	101 seconds
`full best response`, sample rate 120	121 seconds

FIGURE 5.4: Time taken to complete one run of 400 steps

sample size of 15 for the `sampling best response` (just under a quarter of the 125 possible joint actions). To avoid confusing overlap, we do not show the errorbars on this graph: the error range for the reward is approximately +/-2. From figure 5.3 it is clear that the `full best response` policy is superior to either of the approximations for these parameters. Furthermore, the `maximum likelihood` policy is significantly worse than the `sampling best response` policy, with the latter an improvement on the `smart` policy, close to the `full best response`. We therefore did not experiment further with the `maximum likelihood best response` policy. In the next section we will investigate the effects of sample size in more detail.



(a) 2 agents on a 5x5 grid

(b) 3 agents on a 9x9 grid

Figure 5.5: Comparing the `full best response` policy with variants and the `smart` policy, with a maximum step time of 1 second

Secondly, figure 5.5 compares the two simplified best response variants, and the `smart` policy, for two different problems: two agents on a 5x5 grid, and three agents on a 9x9 grid, with a maximum step time of 1 second. Again, we leave off the error bars to enhance readability. We see that in both these cases the `sampling best response` policy actually does less well than the `smart` policy, presumably as it is able to take fewer samples. However, the `full best response` policy even with timeout does nearly as well as without a timeout. We can conclude from this that the consideration of possible actions is more important than the projection over future states. This is because there are so many possible states that changing the number of samples barely changes the fraction evaluated, while going from ten to twenty samples when there are 25 actions makes the difference between only considering half the actions and considering most of them.

5.2.2.2 Varying the sample size

For these experiments, the same "sample size" parameter was used for both sampling from policies, and sampling from actions in the `sampling best response` variant. Figure 5.6 shows the effects of varying the sample size for the 3 agents on a 7x7 grid. We expected that increasing the sample size would improve both the best response policies, since more samples provide more information and thus more accurate estimation of the belief update in all cases. Moreover, the effect should be more noticeable for the `sampling best response` policy, which is also sampling in the best response calculation, up to a sample size of around $5^{numagents} = 125$, where the `sampling best response` policy should approach the full policy. In fact, we see (i) that the `sampling best response` policy significantly underperforms the `full best response` policy and (ii) that although the `full best response` policy improves slightly as the number of samples increases, the `sampling best response` policy actually performs better with a small number of samples. This is because a small number of examples is effectively a best response to "random", while with more samples the agent has enough information to form erroneous conclusions.

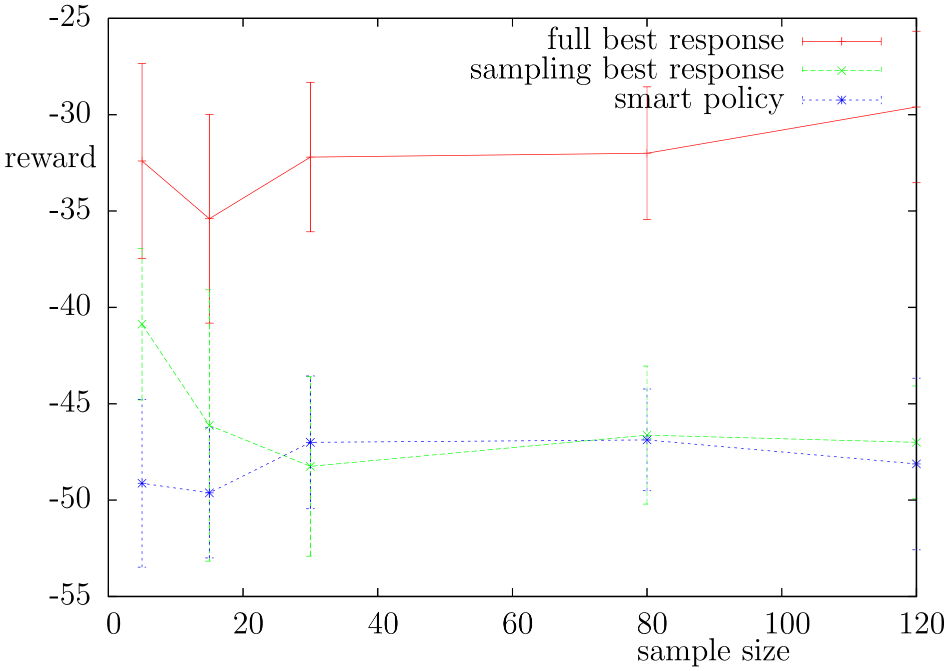


FIGURE 5.6: Effect of varying the sampling size

Given this, figure 5.7 investigates the `sampling best response` policy in more detail. The initial drop as the number of samples increases is less obvious. There appears to be a hump early in each of the lines. Although these could be easily swallowed by the error bars, it is noticeable that each of these apparent humps occurs at fairly low sample rates. The reason for this is low sample rates resulting in more fluctuations in the outputs. Ultimately, we conclude from these results that although it is more efficient timewise, particularly for larger numbers of agents, the `sampling best response` policy is continually less effective than the `full best response` policy, even as the number of samples increases. The remainder of our experiments therefore use the `full best response` policy and we can refer to it as `best response`.

5.2.2.3 Varying the move predictability

These experiments reference particularly the case in which our agent tries to model the intended actions of other agents in a world where the actions are unpredictable. In this case, it is natural that the performance will improve as the move predictability improves, for the `smart` policy because agents are better

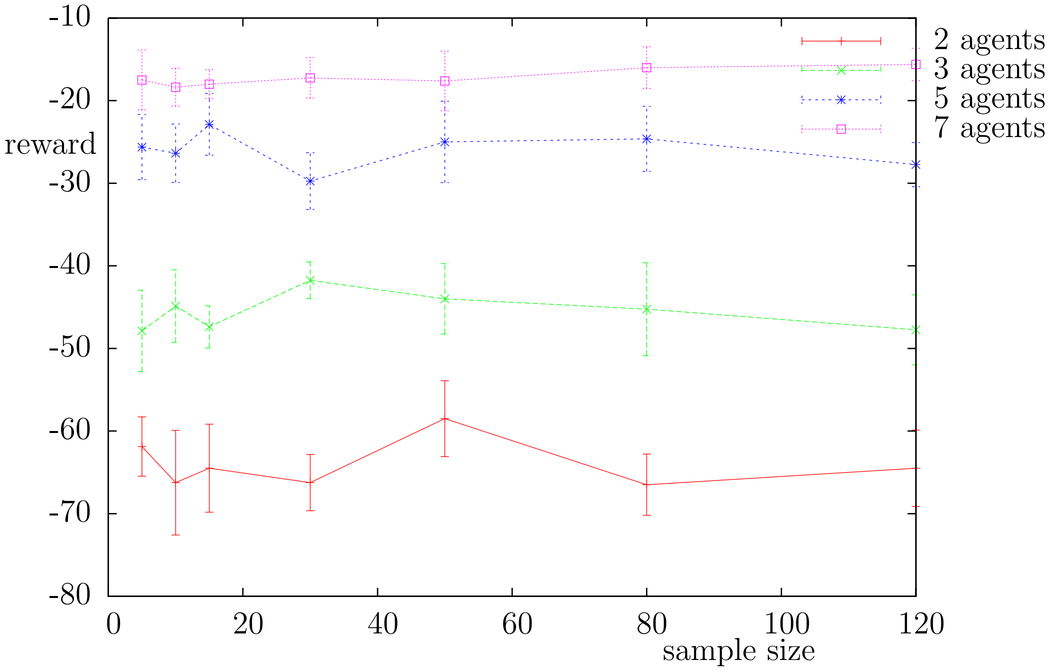


FIGURE 5.7: Effect of varying the sample size for several sizes of problem

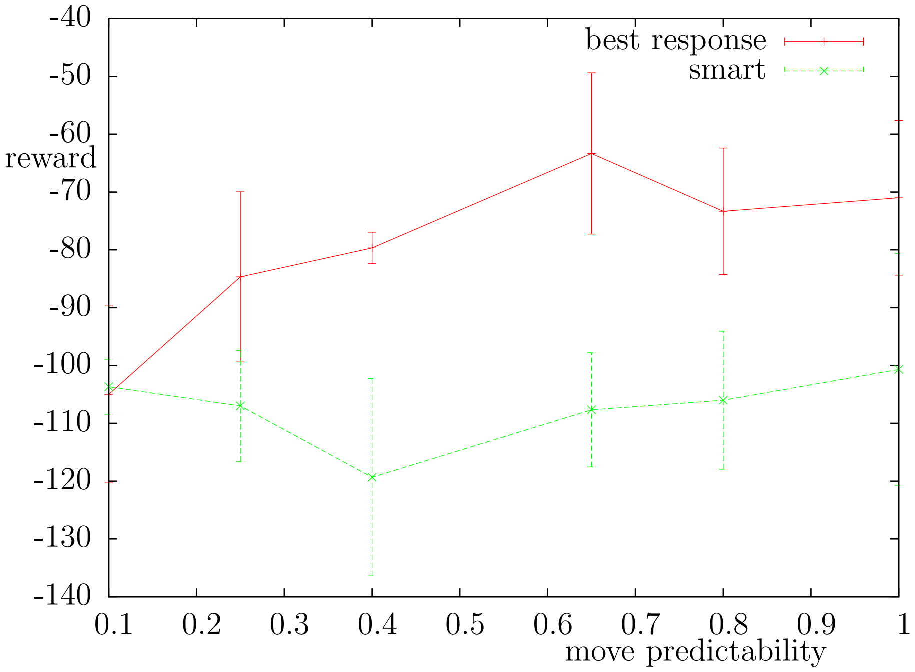


FIGURE 5.8: Effect of varying move predictability

able to do what they intended, for the `best response` policy both because agents are able to carry out their intended moves and because they can model other agents' intentions more accurately. Although the increased level of randomness when moves are unpredictable means that neither policy will be able to do as well as with fully predictable moves, we expect that the `best response` policy will adapt better than the `smart` policy.

In more detail, figure 5.8 shows how the algorithms perform as the actions become more predictable, and we see that both the `smart` algorithm and the `best response` algorithm improve their performance with better move predictability, as expected. The `best response` algorithm improves more rapidly than the `smart` policy and in fact appears to be making little improvement by about 0.8 predictability after rapid improvement initially. This demonstrates that as we gain more information, additional information gives us an increasingly small edge. The sample size, which limits the precision with the `best response` policy can work, is also a contributor to the graph topping out at around 0.8.

5.2.2.4 Varying the number of agents

Since the full `best response` policy scales exponentially with the number of agents, our algorithm does not scale well to many agents and we did not try and investigate beyond 7 agents (figure 5.9 indicates the running times, demonstrating the poor scaling and impracticality of even 7 agents). Section 5.2.2.2 discussed the behaviour of the `sampling best response` policy as the number of agents is increased. Here, we show that, as expected, with more agents on the same board, both `best response` and the `smart` policy are able to increase their rewards (figure 5.10), with the improvements still increasingly almost linearly up to seven agents on the 7x7 board: we expect that as the number of agents is increased, the improvements will continue to the point where all the victims are being saved, and then level out.

We also see that the `best response` policy is not making better use of the additional agents than the `smart` policy, the curves have very much the same

7x7 grid			

Number of agents	`full`
	`best response`
		`sampling`
		`best response`
		(120 samples)
			`sampling`
			`best response`
			(10 samples)

2	3.5s	42 s	2s
3	39s	100s	12s
5	2760s	320s	35s
7	173000s	825s	84s

Figure 5.9: Time taken to complete one run of 400 steps

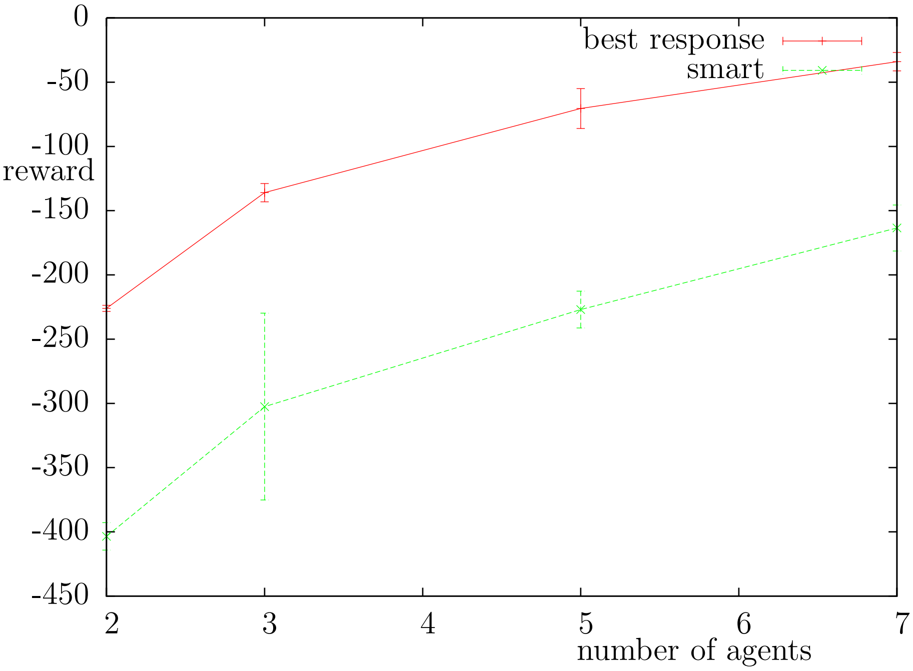


Figure 5.10: Effect of increasing the number of agents

shape, with the difference in reward between the two much the same for seven agents and for two agents. To emphasise this, figure 5.11 demonstrates this effect for a 3x3 grid which is saturated with agents: on this smaller grid the `smart` policy is very effective and we see it making good use of the increased agents. By contrast, the `best response` policy improves more slowly as the number of agents increases, struggling to learn how to make the best use of them. In the next section, we compare the policies on increasingly large boards and show how the `smart` policy loses its edge.

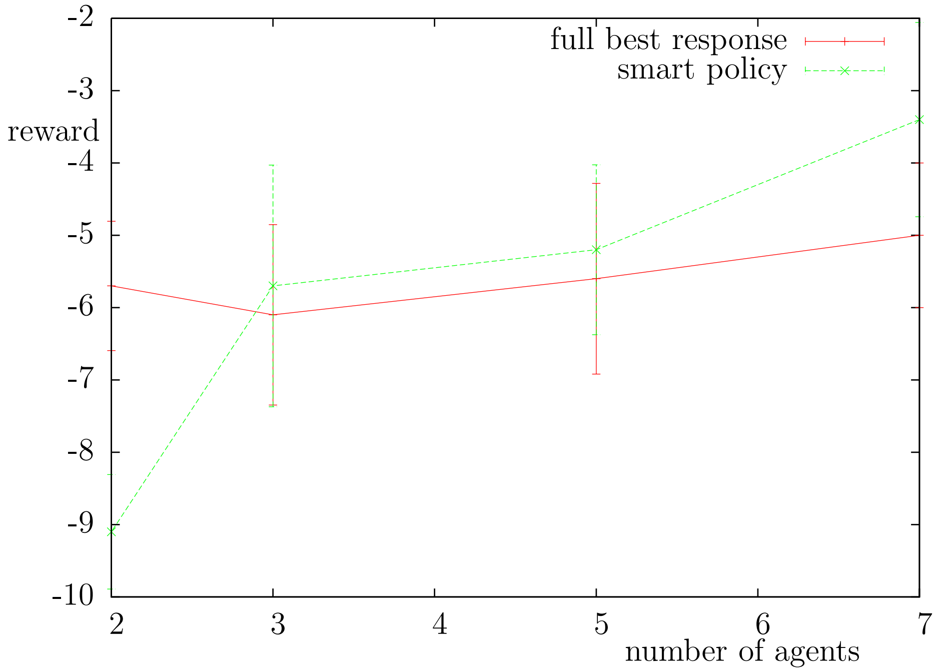


FIGURE 5.11: Effect of increasing the number of agents: 3x3 grid

3 agents	
Board edge size	Time
3	39s
5	1 minute 9s
7	1 minute 45s
9	2 minutes 28s
15	7 minutes 18s
25	34 minutes

FIGURE 5.12: Time taken to complete one run of 400 steps

5.2.2.5 Varying the board size

The other scale parameter which interests us is the number of states, in this problem defined by the size of the board. The running time for this problem scales well as the board size increases (figure 5.12): even on a small board there are millions of states and we must sample future states rather than considering all possibilities. Unlike sampling from actions, choosing a representative sample of states can provide a good estimate for future Q-values as many sets of states will have very similar Q-values.

As we increase the size of the board, and thus the number of states, we find

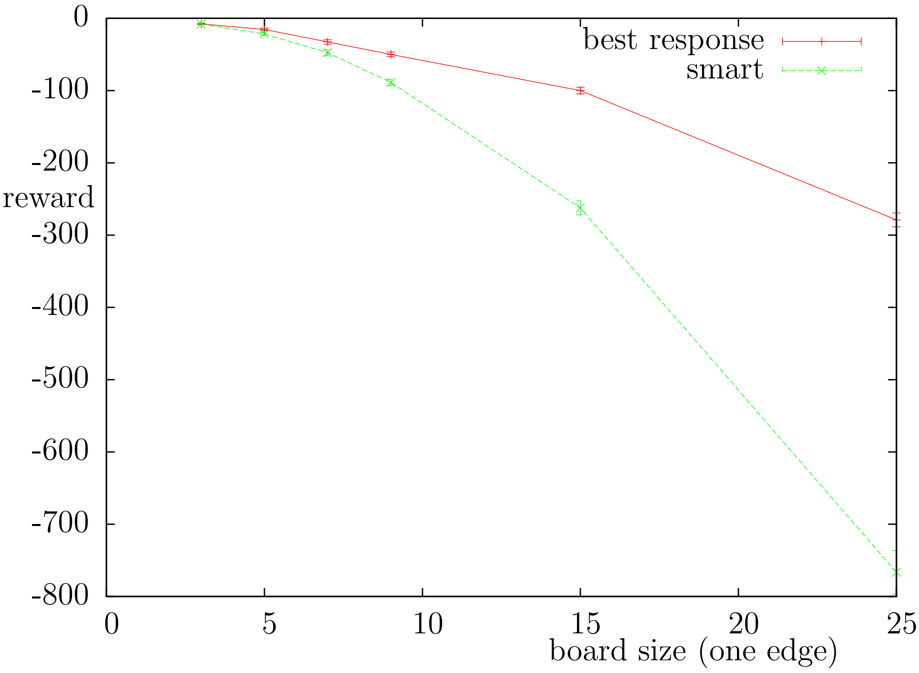


Figure 5.13: Effect of increasing the size of the board

(figure 5.13) that the reward drops as the problem gets harder. However, the `best response` policy is able to maintain higher scores than the `smart` policy. This is because, as the size of the problem increases, the `smart` policy is less able to make good judgements about travelling across the board; furthermore, as new victims appear at every step the `smart` policy can result in agents setting off to carry out a rescue and then doubling back for a new victim, wasting time. The `best response` policy is more flexible, which pays off on the larger boards.

5.3 Ambulance rescue making use of reward information

In this section, we investigate the extent to which the `best response` policy is able to make use of reward information. For this purpose, we used the version of the ambulance problem in which actions have penalties. For implementation purposes, it was convenient to continue to use integer rewards and so the rewards have been scaled up by a factor of ten. As well as the scale factor, it is noticeable that the problem has changed character slightly with the additional information. In particular, our `best response` policy no longer does better than the smart policy on smaller problems, a mark of the additional difficulty of judging rewards given their dual source. The rest of this section describes these experiments in detail, first giving our experimental setup in section 5.3.1 and then explaining our results in section 5.3.2.

5.3.1 Experimental setup

For these problems, we compare the `smart` policy (noting that it is still less optimal as it was not designed to take penalised movements into account), our algorithm not using any information from the rewards (`basic best response policy`) and our algorithm inferring as much as it can from the reward observations (the `full best response policy`). The timing information for this problem was very similar to that for the previous variant of the problem: updates are a little slower but not significantly so and the limiting factors are identical. We therefore do not include any further timing data in the evaluation below. Similarly, we do not investigate the simplified best response algorithms (`sampling best response` and `maximum likelihood best response`) with this problem as their usefulness has already been proven limited.

5.3.2 Experimental evaluation

This section follows a similar format to the previous section's experiments. We begin by looking at the effects of varying the sample size in section 5.3.2.1. We go on to look at the consequences of randomising moves (section 5.3.2.2) and finally investigate problem scaling factors: the number of agents and the board size.

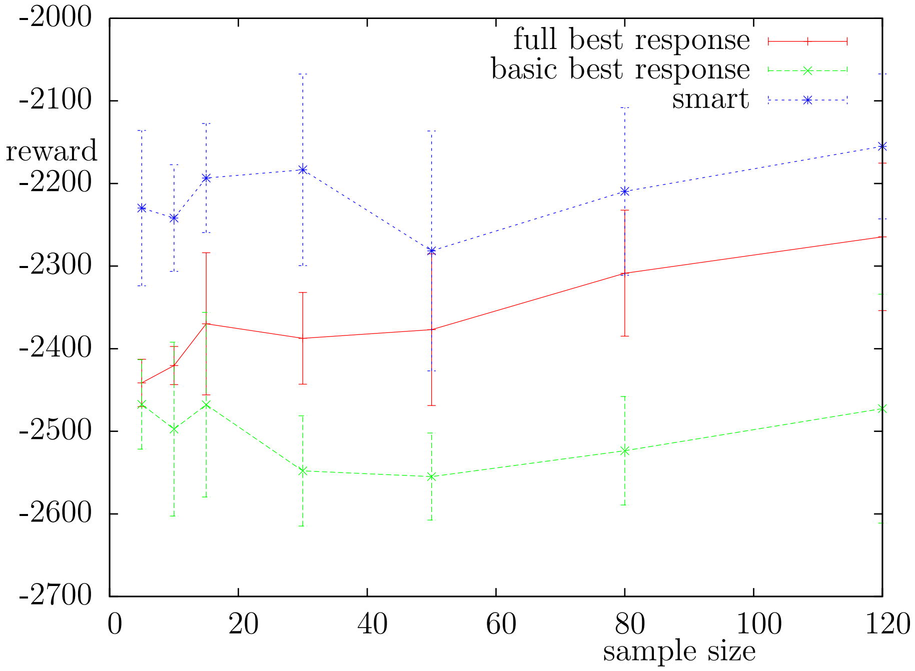


FIGURE 5.14: Effect of varying sample size with inference from rewards

5.3.2.1 Varying the sample size

We begin, once again, by comparing the policies when changing the sample size. As before, we expect that increasing the sample size should result in small improvements in the reward. Since we are sampling from the millions of states rather than the few actions, we do not anticipate that changing the sample size will have dramatic effects on the total reward.

Specifically, figure 5.14 shows the results of our experiments on the 7x7 grid with three agents. We see that the sample size contributes noticeably to the `full best response` policy. The line for the `basic best response` policy is less clear: although there is an upward trend, it follows a dip, and the error bars are broad enough to permit a straight line through. Given the similarities with the previous problem, it seems that taking few samples can be as effective or even more effective than taking 30-50 samples. Again, this is because as the samples increase, the agent has enough information to form erroneous conclusions. As the number of samples increases again, the conclusions become more accurate and the agent's behaviour improves again.

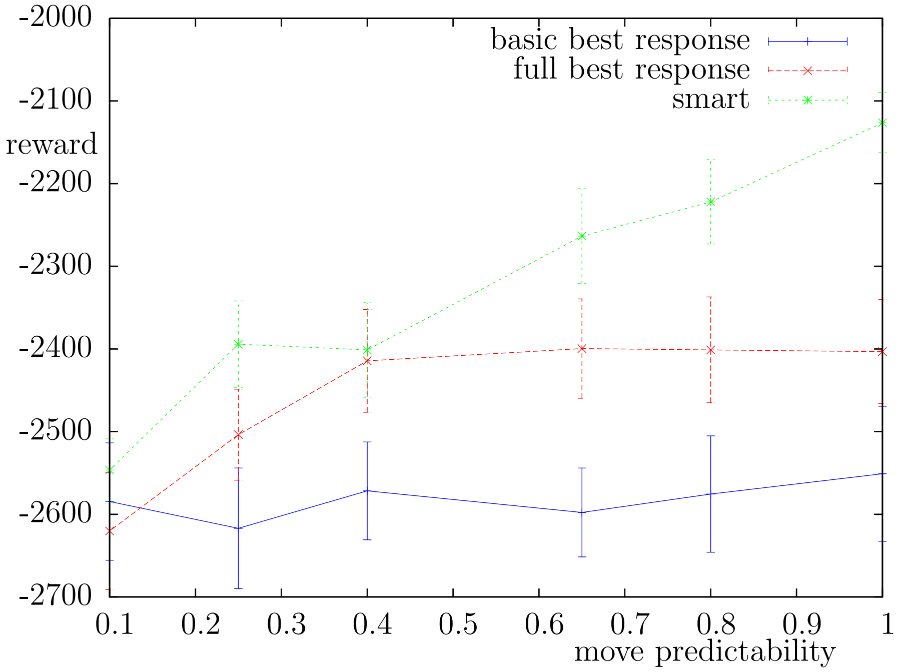


Figure 5.15: Effect of varying move predictability with inference from rewards

5.3.2.2 Varying the move predictability

Figure 5.15 shows the effect of adjusting the move predictability: as expected, as the moves become more predictable, the policies improve. From the figure, we can see that both the `smart` policy and the two best response policies do less well on this changed problem than they did on the problem without penalties: firstly, the rewards are considerably lower, and secondly, neither of the best response policies is making efficient use of the increased predictability. We also do not see the "topping out" effect of the `smart` policy that we observed on the previous problem, because the problem is harder.

5.3.2.3 Varying the number of agents

Figure 5.16 shows the effect of increasing the number of agents. As before, increasing the number of agents scales very badly timewise; here, particularly with the additional inference step which has to consider all agents. Consequently, we do not investigate more than seven agents (on the 7x7 grid).

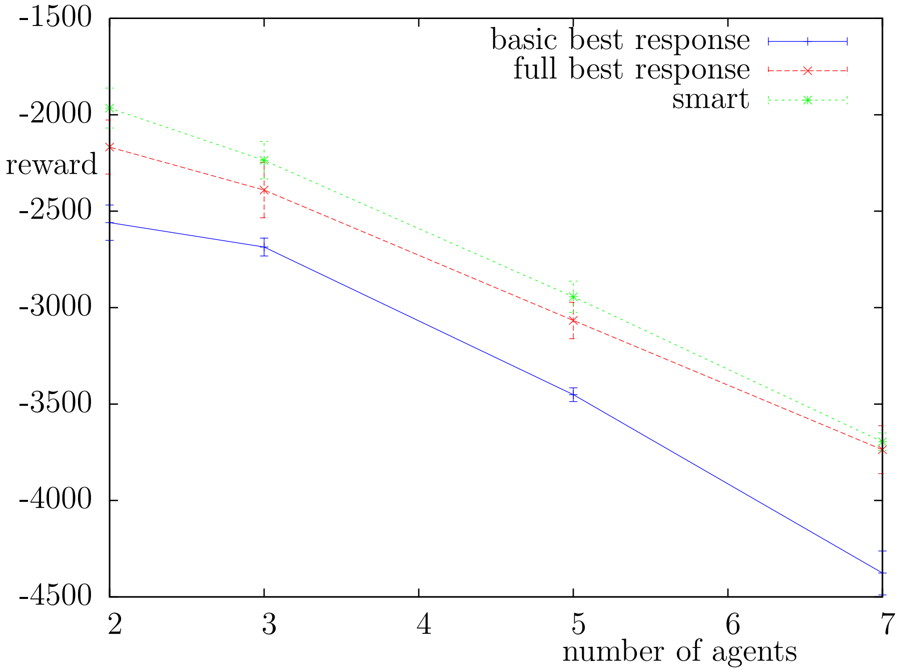


FIGURE 5.16: Effect of increasing numbers of agents with inference from rewards

We expect that the behaviour will be similar to the behaviour on the previous problem, once again with the `smart` policy outperforming the `full best response` policy on the new problem. In fact, we see again the increased difficulty of this version of the problem in the fact that the reward actually drops as the number of agents is increased, rather than climbing. The relative behaviour of the policies is as expected, with the `smart` policy outperforming the `full best response` policy. We do see that the `smart` policy, which doesn't take the action penalties into account at all, drops linearly, while the more flexible `full best response` policy is able to hold up slightly better. We go on to investigate our final scaling factor, board size.

5.3.2.4 Varying the board size

Figure 5.17 shows the results of the different algorithms as the board size is increased. From this figure we see that in this problem, the `full best response` algorithm does less well than the `smart` algorithm at smaller board sizes. The `full best response` continues to struggle with the harder problem, but the `smart` is

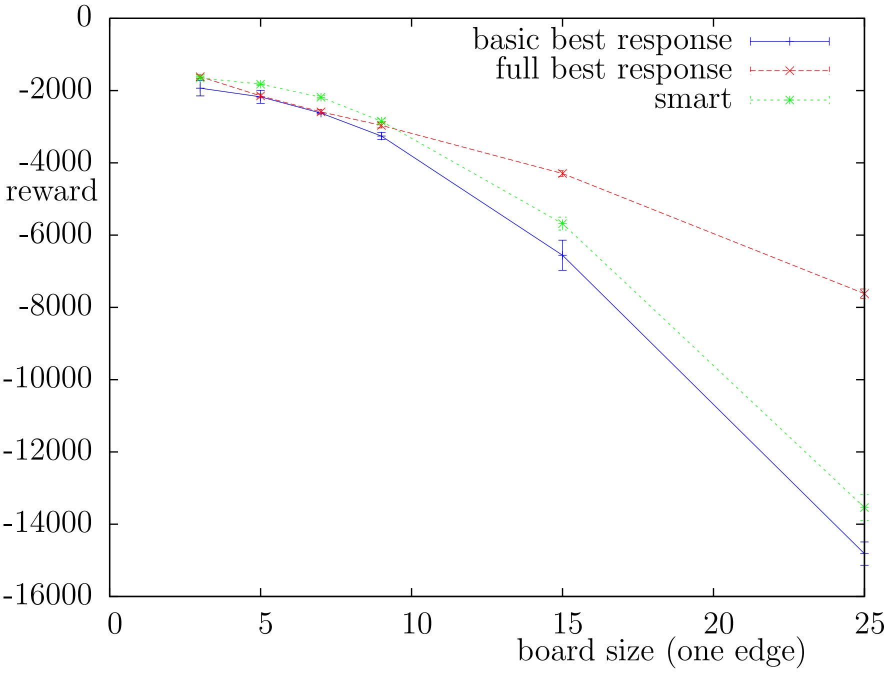


FIGURE 5.17: Varying board size with inference from rewards

able to coordinate over small boards. However, as with the simpler variant of the problem, the `smart` policy's deficiencies become more noticeable as the board size is increased. We now see the `full best response` algorithm outperforming the `smart` policy once the board size increases past a 9x9 grid. This is encouraging as it demonstrates that our algorithm scales well as the number of states increases, and is able to find satisfactory policies where a handwritten policy is suboptimal.

5.4 Summary

In this chapter we have explained how to specialise the model of chapter 3 to a specific case of partially observable actions. This is the first such attempt to describe partially observable actions using a formal Bayesian model. We have then evaluated this model using two variants of the ambulance rescue problem, comparing it with a handwritten policy designed for this problem. For the simpler variant of the simulation, we have found that although the handwritten policy can optimise well on small problems, our model is better able to find satisficing policies for much larger problems.

For the more difficult variant of the simulation, we have seen that the behaviour trends for our policy are, as expected, the same; however in most of the smaller problems the handwritten policy is able to outperform the best response algorithm. Encouragingly, when the number of states is scaled up, the best response algorithm again begins to outperform the handwritten policy, demonstrating its potential for larger problems. Thus in both cases, we conclude that the best response algorithm, with all its approximations, is not optimal and does not replace an optimal algorithm for small problems. However, it is able to scale well to problems with many more states where a satisfactory rather than optimal problem is required. It does not perform so well in problems with many agents which have exponentially many joint actions to be considered as possible future actions. In section 8.3.2 we outline some possible ways of approximating the joint actions in order to tackle this problem.

In the next chapter, we continue the evaluation of the model described in chapter 3, considering a second, more complex, specialisation of the model in chapter 3 and again demonstrate our algorithm on the ambulance problem. In this chapter we have applied our model to learning agent models for coordination in scenarios in which agent actions are partially observable. We go on to investigate learning agent models for coordination in scenarios in which states are partially observable.

Chapter 6

Coordination in the presence of partially observable states

In this chapter, we evaluate the model of chapter 3 for the case in which the actions of the other agents and the transition function are known, but the state is only partially observable and the behaviour policies of the other agents (which depend on their observations) are known. We use finite state machines to approximate the observation-based policies of other agents, enabling us to efficiently evaluate our online multi-agent POMDP. Thus, we substantiate the claim in 1.4 that this is the most effective online multi-agent POMDP with explicit modelling of other agents. We also demonstrate that our algorithm is more efficient than the current state of the art and more effective than a handwritten strategy for the problem.

To achieve this, we begin by explaining how the model in chapter 3 specialises to this case, detailing the update and best response computations (section 6.1). We then evaluate the model in section 6.2, beginning by investigating the clustering model which we detailed in section 3.4.2 and apply here (section 6.2.2.1) before going on to test our model over a number of problem parameters and size factors.

6.1 Evaluating the general model with partially observable states

In this model, the underlying global state is not visible to the agent, only some local observations o, as in the Tamptono example 4 where the agent can see the state of the nearby bridges, but does not know what is going on elsewhere. We assume that the model $P(o|s)$ is known to the agent. Furthermore, each state gives rise to a deterministic set of observations ($Obs(s) = \{o_1, o_2, ...\}$), and the agent will see some subset of these. Therefore, for each set of observations o, there will be a known, fixed set of states which may have given rise to the observations: s such that $o \subseteq Obs(s)$. Each set of observations o is local to a particular agent, so the agent's strategy will depend on its own local observations. The belief state now contains, instead of the current state, the current set of observations and a distribution over state probabilities.

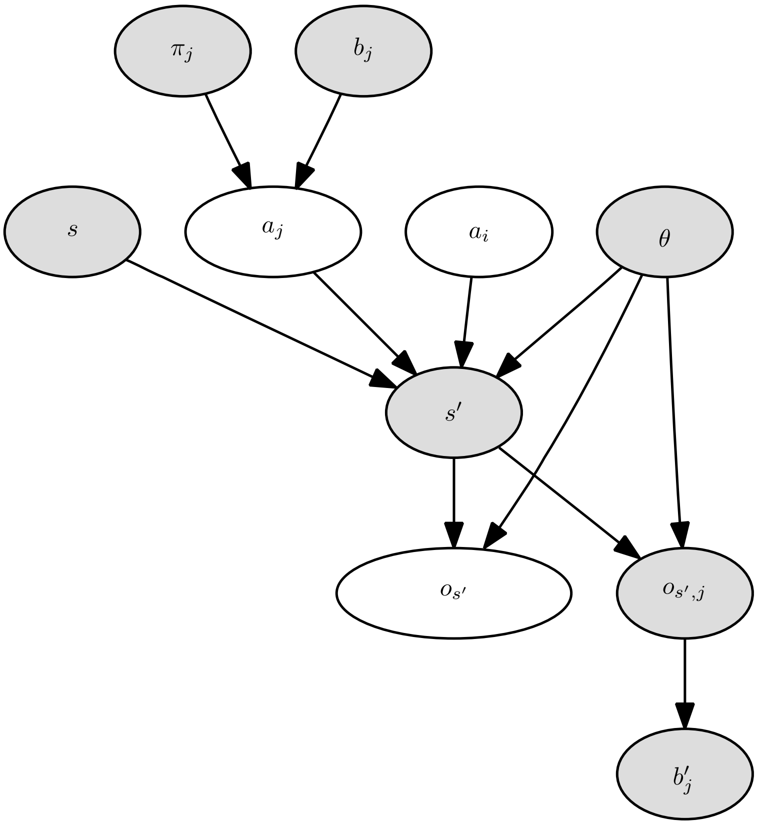


FIGURE 6.1: Partially observable states

Now, although the underlying process is still assumed to be an MDP (i.e., the current state is dependent only on the previous state and the action choices), the sequence of observations is no longer Markov. In particular, where the environmental models are completely known, we can define a probability distribution over states at each step, and the sequence of such distributions is itself Markov. If the probability distributions were themselves being estimated, the sequence of belief states would also be non-Markov (alternatively, each belief state should contain (and use in the updates), the complete history trail for the problem). However, we leave such problems to future work (Chapter 8).

Finally, in the multi-agent case, the learning agent must retain in its own belief state beliefs about the belief state of the other agents—not other agents' beliefs about the models or strategies, but their beliefs about the current underlying state. These beliefs will depend on the sequence of observations made by those agents, which will be only partially known to the learning agent. We use the "history" variable to represent this, noting that each agent will have an independent history variable unknown to the other agents.

To this end, figure 6.1 gives the Bayesian network diagram for a multi-agent system with partially observable states. In this diagram we have separated the observations and the actions of the agent under consideration (subscript i) and the other agents (subscript j). Tracking the network flow downwards, the root node is the original state, s, giving rise to a visible set of observations for the agent, and unknown sets of observations for the other players. We assume that the other players update their belief states given their observations and act accordingly. As previously, the actions, previous state and the system dynamics determine the next state, which emits a reward and which all agents make new observations from.

Referring to this diagram, we obtain the following updates (note that *obs* refers to all the observations made by our agent, including rewards and actions of other agents, as distinct from o_i the observations made from the state):

$$
\begin{aligned}
P(M|obs) &\propto \sum_{s,s',\pi_j,o_j,b_j,h} P(s,s',\pi_j,o_j,b_j,h,o'_i,o_i,r,\mathbf{a},M) \\
&= \sum_{s,s',\pi_j,o_j,b_j,h} P(r,o'_i|s')P(s'|M,\mathbf{a},s)P(M)P(a_j|\pi_j,b_j)P(b_j|o_j,h)P(o_j|s)P(s)P(\pi_j) \\
&= P(M)*\ldots \\
&\ldots\sum_{s',s} P(r,o'_i|s')P(s'|M,\mathbf{a},s)P(s)\sum_{o_j}P(o_j|s)\sum_{b_j,h}P(b_j|o_j,h)\int_{\pi_j}P(a_j|\pi_j,b_j)P(\pi_j)
\end{aligned}
$$

$$
\begin{aligned}
P(\pi_j|obs) &\propto \sum_{s,s',M,o_j,b_j,h} P(s,s',\pi_j,o_j,b_j,h,o'_i,o_i,r,\mathbf{a},M) \\
&= P(\pi_j)*\ldots \\
&\ldots\sum_{s,s'} P(o'_i,r|s')P(s)\sum_{b_j,h}P(\mathbf{a}|\pi_j,b_j)P(b_j|o_j,h)\sum_{o_j}P(o_j|s)\int_{M}P(s'|M,\mathbf{a},s)P(M)
\end{aligned}
$$

As before, the definition of $Q(a_i,b_i)$ remains unchanged. However, we now define

$$
\begin{aligned}
P(a_j|b) &= \sum_{\pi_j,b_j} P(a_j|b_j,\pi_j)P(b_j,\pi_j|b) \\
&= \sum_{\pi_j,b_j} P(a_j|b_j,\pi_j)P(b_j|b)P(\pi_j|b) \\
P(r,s'|\mathbf{a},b) &= \int_M P(r,s'|o,\mathbf{a},M)
\end{aligned}
$$

where

$$
P(b_j|b) = \sum_{h,o_j} P(b_j|h,o_j)P(h|b)\sum_s P(o_j|s)P(s|b)
$$

and

$$
\begin{aligned}
P(r, s'|o, \mathbf{a}, M) &= \sum_s P(r, s'|s, \mathbf{a}, M) P(s|\mathbf{a}, M) \\
&\propto \sum_s P(r, s'|s, \mathbf{a}, M) \int_b P(\mathbf{a}|b) P(b|s) P(s)
\end{aligned}
$$

(Note that the diagram in figure 6.1 shows the reward dependent on the initial state and action choice. However, similar to the discussion in section 5.1.3, this network (and equations) can be easily modified to make the reward dependent on the resulting state).

In principle, the equations above can be solved using a system of Dirichlets, with sampling for the integrals, similar to that already described for partially observable actions. However, these new equations are more computationally complex than those defined for the case of partially observable actions, with multiple summations which make them impractical to evaluate on any realistic problem because the complexity of the computation becomes exponential in each of the summed parameters.

In particular, notice that $P(M|obs)$ is no longer independent of the strategy model, despite the action observability. This is because of the need to maintain beliefs about the beliefs of the other agents; these beliefs are continuous, necessitating sampling, and they depend on another unobserved variable, the observations made by the agent. (An example might be an agent, Alice, observing smoke coming from a building and believing it to be on fire, so running from the building. From another angle, it may be clear to our agent, Bob, that the smoke is merely rising from a burnt-out fire and there are victims critically needing help within the building—but Bob must work with the behaviour of the others).

This Bayesian approach to multi-agent strategies is similar to the approach described by Emery-Montemerlo et al. (discussed in section 2.4), who only consider games with known dynamics. Despite this, several approximations are required to make computation feasible in their model, and they consider systems with few states and few agents. Thus for us also, several approximations are necessary: in particular, as well as the simple approximation strategies in section 3.4.4, we will apply the novel approach of using finite state machines to model other agents in this online setting, as detailed in section 3.4.3. We will also investigate the state clustering model described in section 3.4.2 in this chapter—the next section begins by investigating the effectiveness of the clustering model before going on to evaluate our models across different parameters, investigating in particular the scaling properties of the algorithm.

6.2 Ambulance rescue with partially observable states

In chapter 2, we have motivated the importance of investigating scenarios with partially observable states in our target domain of disaster response and above we have outlined how the algorithm in section 3.2, extended with statistical clustering (section 3.4.2) and finite state policy approximations (section 3.4.3) can be specialised to such cases. In this section we evaluate this finite-state machine algorithm in detail. We begin by outlining our experimental setup (section 6.2.1).

6.2.1 Experimental setup

In order to test our strategy, we compare it against two other online algorithms: the state of the art for online partially observable stochastic games is the Bayesian game approximation using the finite-horizon approximation technique (Emery-Montemerlo et al., 2004), described in section 2.3.1 (“`POSG`”). However, for large dynamic problems, this algorithm, which is exponential in the number of agents, proves to be very inefficient and we find that for all but the smallest variants of the rescue problem, `POSG` is too slow to be useful. Therefore, as before, we compare against the handwritten strategy written for Robocup Rescue, `smart`.

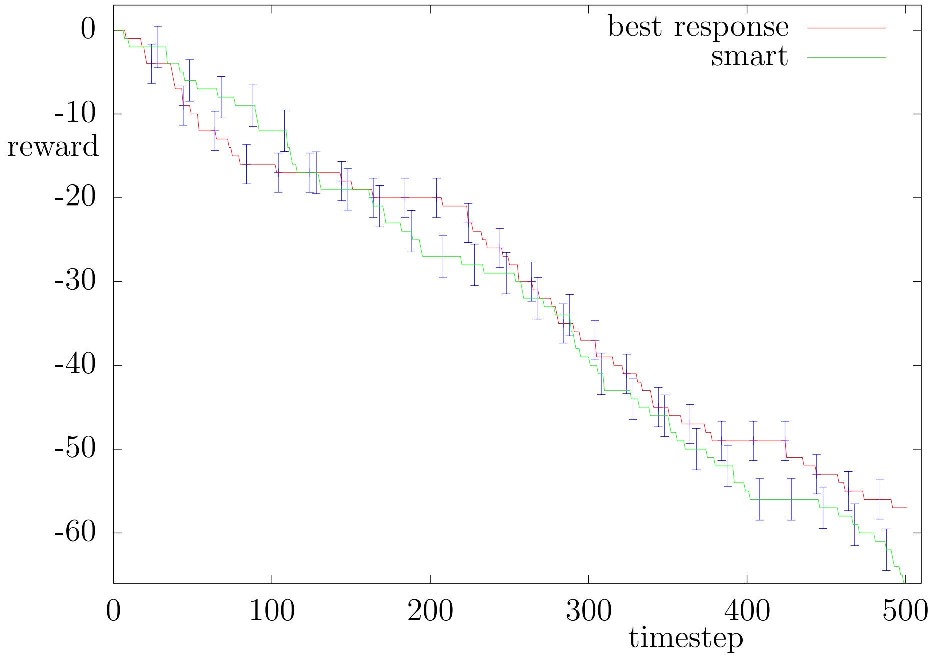
We investigate our algorithm, "`best response`", over different parameter settings on the rescue problem, and then focus on our particular interest, the scaling properties of the algorithm. As a baseline, we include the null policy in which agents move randomly, but never dig and so never effect any rescues ("`null`"). In the next section, we identify the fixed parameters, before going on to our results.

We use the same problem settings as in chapter 5, varying m, n and k as specified. We also experimented with increasing p_a towards problems where "dig nearby" becomes a viable strategy, and varying v, the state visibility parameter. Finally, in the belief-state based algorithms, we must take samples from the belief state. We define the *sampling rate* as the number of samples taken for each variable, initialising it at a rate of 35 (for comparison, previous work on a single agent problem found that 20 samples was sufficient for good solutions (Dearden et al., 1999)).

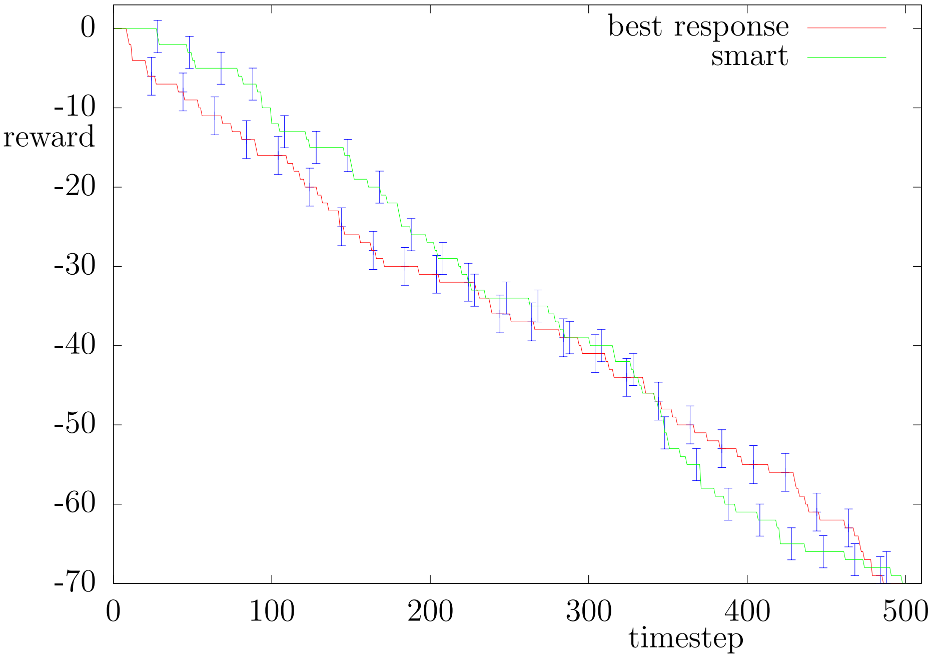
In every experiment, we carried out several runs of the problem, varying the initial placement of civilians and randomising their arrival and visibility. The same random seed was used to initialise each of the test algorithms in each run. The error bars included in the results show the 95% confidence intervals around each point. The rest of this section discusses our key results.

6.2.2 Experimental evaluation

We begin by investigating the properties of our FSMs, in order to settle on suitable parameters to continue with. This first section (section 6.2.2.1) will investigate the effects of the observation clustering outlined in section 3.4.2 on performance, and then sampling rate and observation history. We go on to examine the learning properties of the algorithm in section 6.2.2.2. The next parts of this section then examine the behaviour of our algorithm when varying parameters such as visibility (section 6.2.2.3) and victim arrival rate (section 6.2.2.4). Finally in section 6.2.2.5 we look at the effects of problem size on the performance of our algorithm.



(a) With max 500 clusters



(b) With max 1500 clusters

FIGURE 6.2: Effect of cluster capping over a run

6.2.2.1 Finite state machine properties

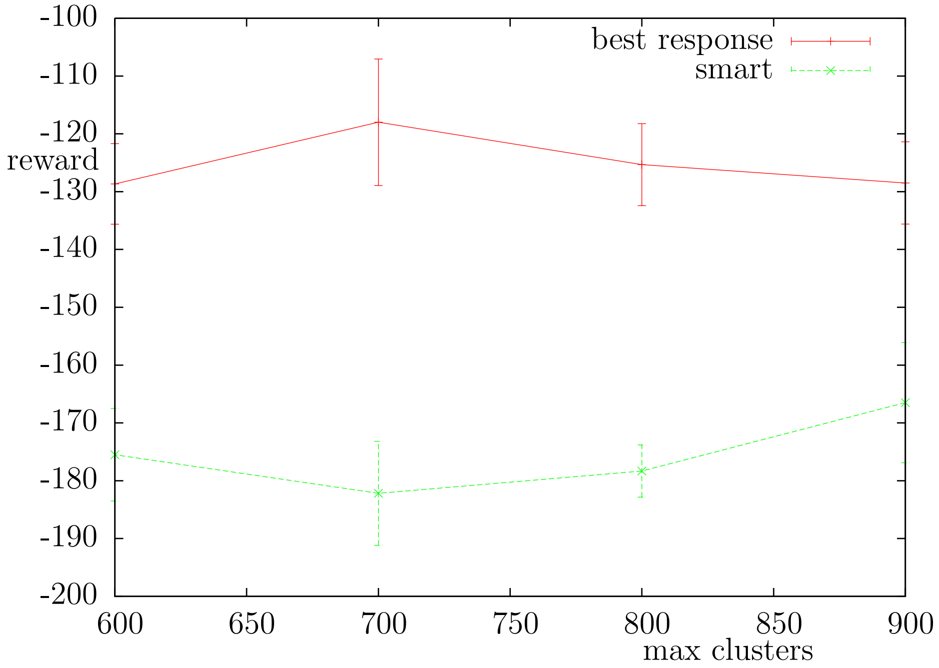
We first experimented with the clustering model, varying the maximum number of clusters to see the effect on the results. We expect to get poor results with low numbers of clusters, but to find that increasing the number of clusters does not improve the results much after a certain point. It is possible that as the number of clusters increases towards the total number of states, the agent may not be able to make use of the clusters to make inferences about future states, resulting in a loss of performance.

To this end, figure 6.2 compares two experiments on a 7x7 board with 3 agents (each averaged over 15 runs), one capping the number of clusters at 500 and one at 1500. Two effects can be seen: first, the learning in our algorithm is shown in the curving of the `best response` policy line against the straight `smart` policy line; second, with up to 1500 clusters, this learning effect kicks in a little later on.

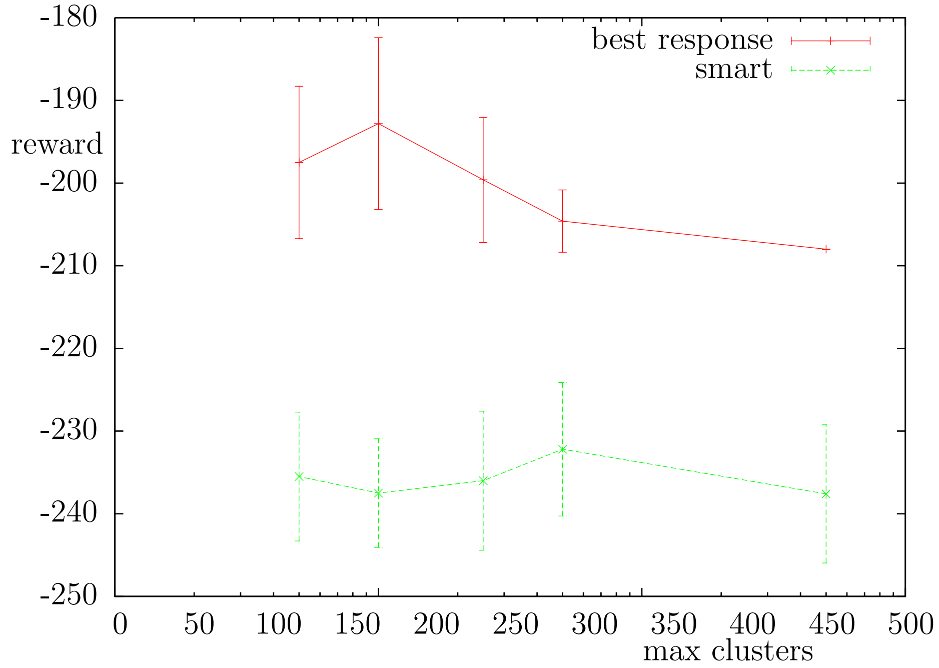
Figure 6.3 demonstrates the effects of clustering for a small and a medium variant of the problem: we see the small hump around 700 clusters for the three-agent problem and at around 200 clusters for the two-agent problem. Although these humps are dwarfed by their error bars, this effect was consistently observable across many experiments. This corresponds with our stated expectation that the results would improve up to a certain point, and then either stop improving or worsen. Thus, in general we see that there is a slight 'hump' in the results identifying an ideal number of clusters to use for a particular problem; however, the variation in results is sufficiently low that this choice is not critical. Furthermore, since we are clustering primarily for efficiency, we must also consider what the tradeoff is in terms of efficiency.

We now consider the notion of time. As the number of clusters increases, the generated finite state machines become larger, and searching in the structures thus slower. To this end, the timings in figure 6.4 refer to the experiments running on a dual-core machine in the university's Beowulf cluster, showing that the time needed increases exponentially with the number of clusters. Therefore, if time is a premium, reducing the maximum number of clusters could give noticeable efficiency gains without significantly affecting the results.

We now move on to experiments in which the sampling rate is varied. In order to examine how the `best response` algorithm will perform on challenging problems



(a) 7x7 board, 3 agents

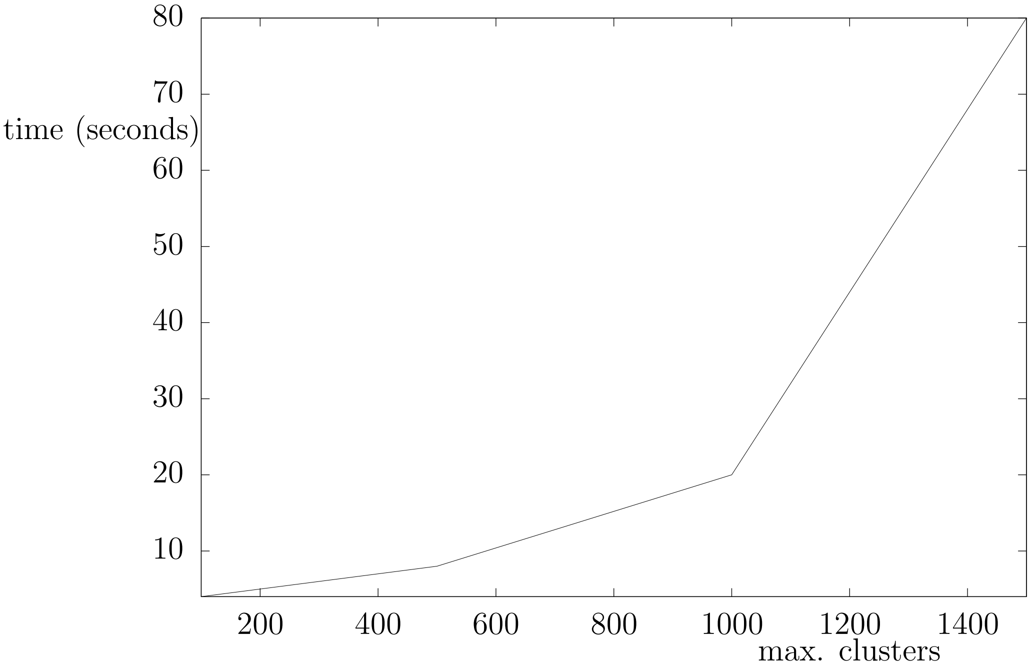


(b) 7x7 board, 2 agents

Figure 6.3: Effect of clustering on two variants of the 7x7 problem

7x7 board	
Max clusters	Time
100	4 seconds
500	8 seconds
1000	20 seconds
1500	80 seconds

(a)

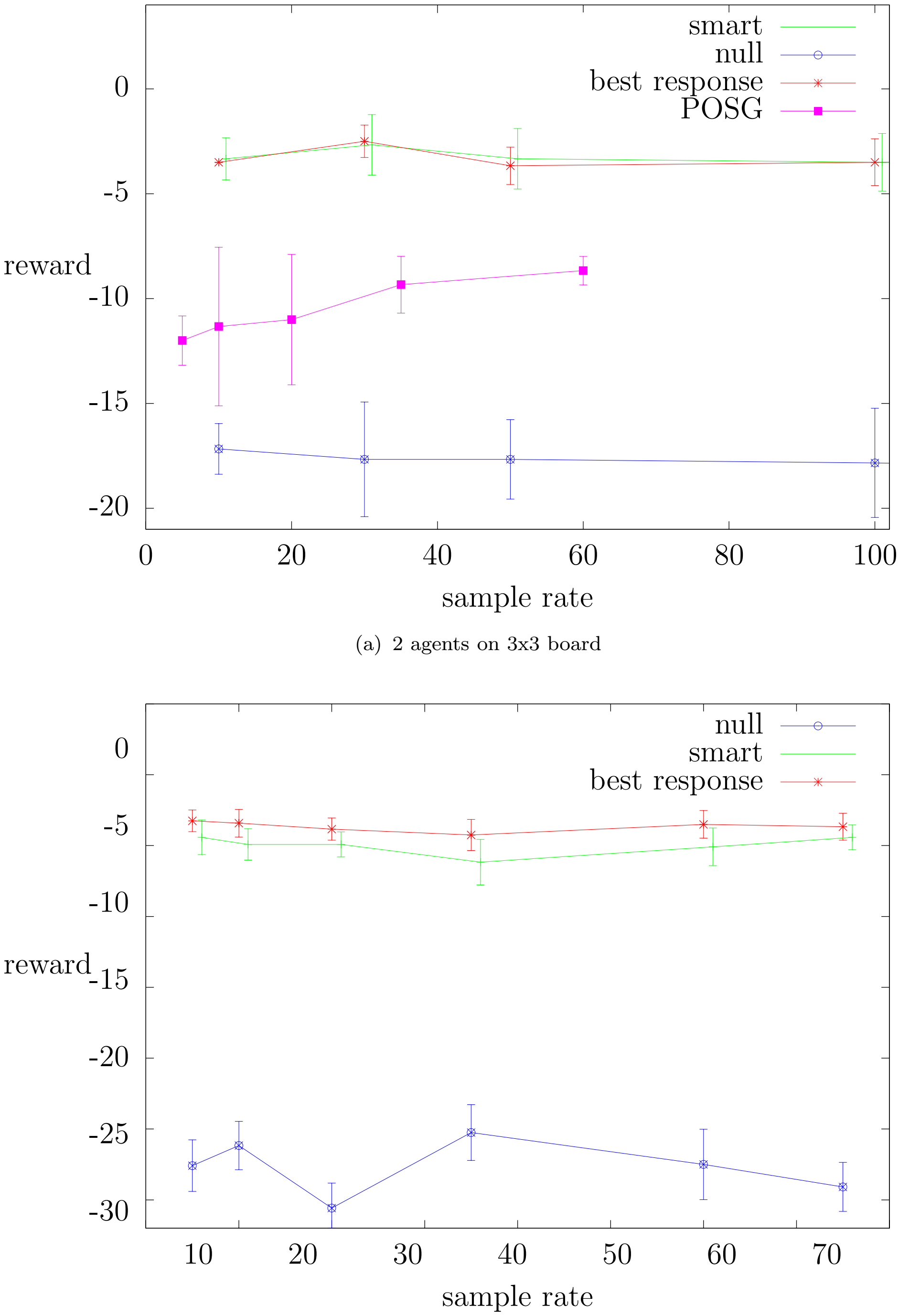


(b)

Figure 6.4: Time taken to complete one run of 1500 steps

such as those we identified in our domain requirements, we will consider the effects of scale both on solution quality and on the computational requirements. Linked to the way the solution scales is the number of samples taken in estimating beliefs. The sensitivity of the solution to the number of samples is therefore relevant in considering the effectiveness of the algorithm.

In more detail, for the `POSG` algorithm, on a 3x3 board with two agents, figure 6.6(a) shows the times for 100 steps, for various sample rates. Since our cut-off was one step per minute, we did not run any tests on the `POSG` algorithm beyond a sample rate of 75, the `null` policy and the `smart` policy do not do any sampling. For our own policy, which does not need to iterate over all *joint* policies, the scaling factor was much better: figure 6.6(b) shows the equivalent rates. The `POSG` algorithm is exponential in the number of agents, since it iterates over all



(a) 2 agents on 3x3 board

(b) 3 agents on 5x5 board

Figure 6.5: Effects of changing the sampling rate with two and three agents

3x3 board, 2 agents	
Sample rate	Time
10	720 seconds
20	1140 seconds
35	2280 seconds
60	4020 seconds
75	6780 seconds

(a) `POSG` algorithm

3x3 board, 2 agents	
Sample rate	Time
10	7 seconds
30	18 seconds
50	27 seconds
100	50 seconds
500	240 seconds

(b) `best response` algorithm

7x7 board, 3 agents	
Sample rate	Time
10	18 seconds
35	48 seconds
60	300 seconds

(c) `best response` algorithm

Figure 6.6: Time taken to complete one run of 150 steps

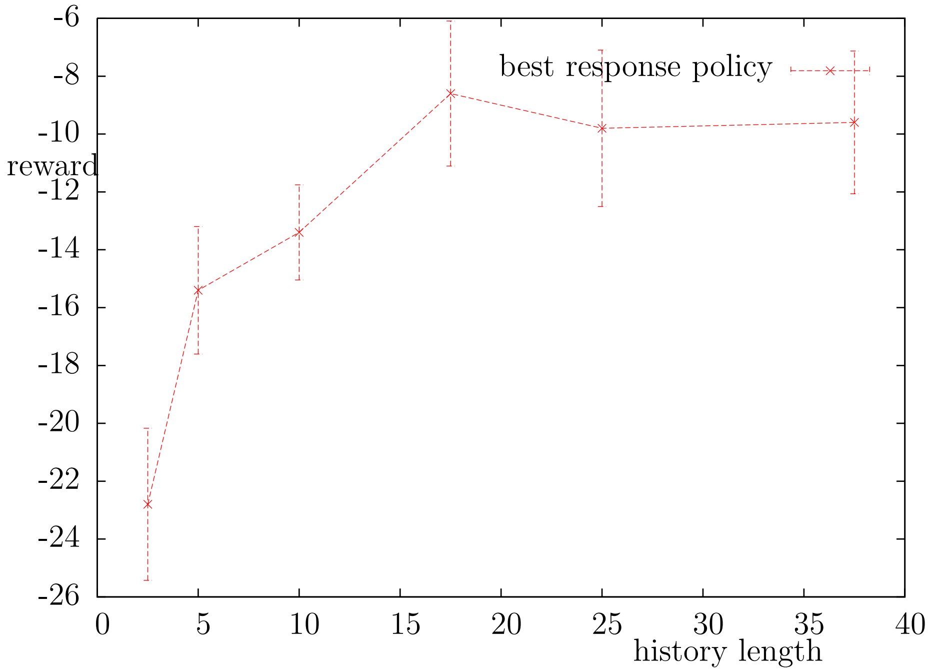
joint actions. It therefore scales badly as the number of agents is increased. By contrast, figure 6.6(c) shows the times for the `best response` algorithm running on the larger problem of a 7x7 board with three agents. Even on this larger board the times are well within the range of acceptable. We next investigate whether there is truly a need for higher sampling rates, since our earlier investigations (in section 6.2.2.1) indicated that the `best response` algorithm is able to perform quite well even at low sample rates.

To this end, figure 6.5 shows the effect of changing the sample rate. As expected, neither the `null` policy nor the `smart` policy are susceptible to changing sample rates. However, the performance of the `best response` policy also does not vary much with the changing sample rates. It is also worth remarking that the error does not reduce noticeably as the number of samples is increased, suggesting that the same actions are selected with as few as ten samples. This occurs because the effective options to the agent are “Move” or “Dig”—in a problem of this nature it is very difficult to differentiate between move directions except when the target is next door, since several different move sequences correspond to the same target. Therefore, if a majority of the samples indicate that the agent should dig, then it will dig. Otherwise, with the relatively small numbers of samples we consider, the agent is unlikely to settle clearly on a move direction. By contrast, the `POSG` algorithm, which initially performs more poorly than the `best response` policy, improves noticeably as the number of samples is increased, and the error around the points reduces. The `POSG` algorithm relies on more information than the `best response` algorithm and so the low numbers of samples are not sufficient for it to make a good decision even to the extent that the `best response` algorithm does.

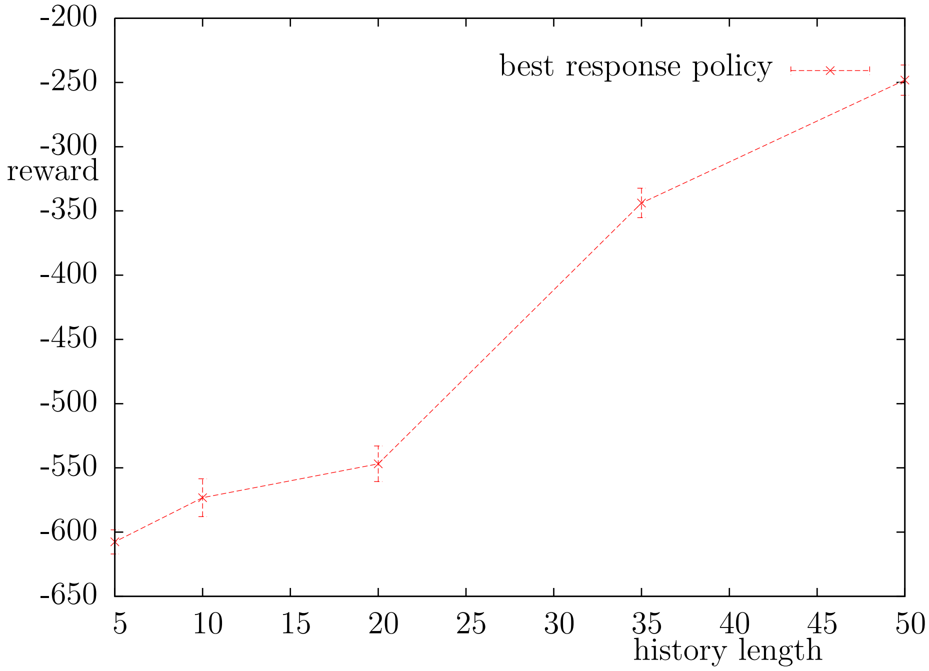
These results indicate that similar actions are selected even with a small number of samples, because the best response can be estimated well, and the `best response` algorithm performs well with small sampling rates, making it possible for the algorithm to be very efficient. This compares favourably with the `POSG` algorithm which approaches optimality at high sampling rates but performs very badly at low sampling rates, at least for this type of problem. Due to the time constraints we imposed, we do not investigate the `POSG` algorithm in the larger version of the problem (figure 6.5(b)) but we see that as for the larger problems above, the `best response` algorithm slightly outperforms the `smart` policy, due to its better handling of imperfect visibility. Section 6.2.2.3 will investigate the effects of visibility in more detail.

Finally, we consider the issue of observation history. In particular, in section 3.4.3.2 we noted that the other parameter affecting the finite state machines is the length of observation strings which they use. We expect that longer observation strings would result in more accurate FSMs and better strategies, since the agent can consider longer term strategies. Figure 6.7 shows the effect of lengthening the observation history for our `best response` policy, demonstrating that on both a small problem (figure 6.7(a)) and a large problem (figure 6.7(b)), longer observation histories do indeed result in improved strategies. Furthermore, it can be seen that on the smaller board, the longer history ceases to be useful after around a history length of 15, while on the larger board, with many more states and state trajectories, the results continue to improve as the history length is increased as far as 50.

The next step in our investigation of the behaviour of our finite state machines, and before continuing to investigate the effect of general problem parameters such



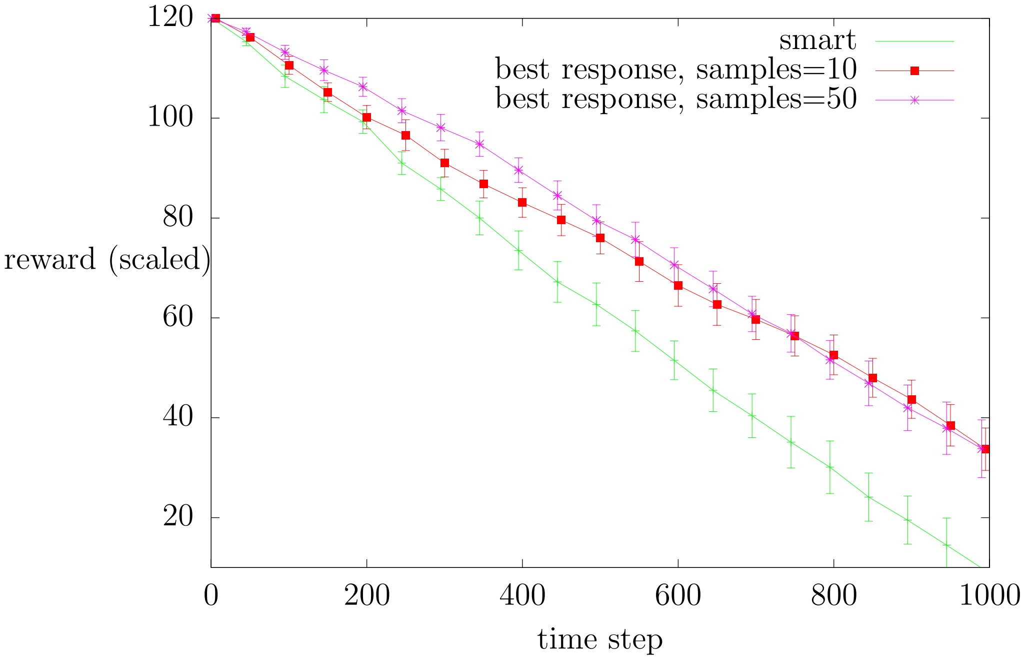
(a) 5x5 board, 2 agents



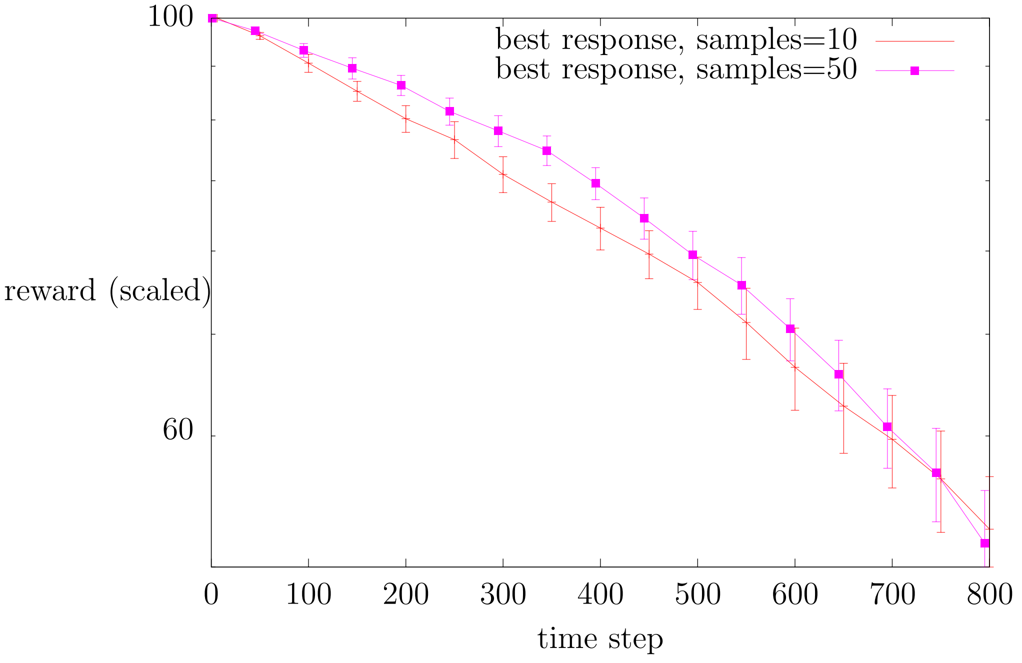
(b) 9x9 board, 5 agents

FIGURE 6.7: Effect of lengthening observation history

as problem size, is to take a look at the learning rate of our algorithm. This is important because it shows how the algorithm is making use of the models or FSMs which are being learned, and gives us some idea of how many steps are required before the algorithm becomes most useful.



(a) Algorithm performing over 1000 steps at two different rates



(b) Closeup of the first 800 steps

FIGURE 6.8: Comparison of two algorithms over time on a 7x7 board with 3 agents. Note that we use a log scale to show more clearly the differences between the algorithms, and the rewards are scaled up to > 0 for the log scale.

6.2.2.2 Examining the learning rate

To begin with, we compared the algorithms over the course of 1000 steps on a 7x7 board, with three agents. We found that the POSG algorithm, which is exponential in the number of agents, did not complete in any reasonable time (we consider one minute per step to be "reasonable") on this size of problem, taking ten minutes for one agent to complete a single step. Thus, figure 6.8 just compares the smart policy with our algorithm over 1000 steps. Our aim was to examine the performance of the best response algorithm on a challenging problem, focusing on any changes in its behaviour over time. To this end, we have used two different sampling rates for the best response policy, comparing how the agent learns when sampling very little information ($samplerate = 10$) or more information ($samplerate = 50$). We expect that the agent will both perform better, and learn faster at the higher sampling rate.

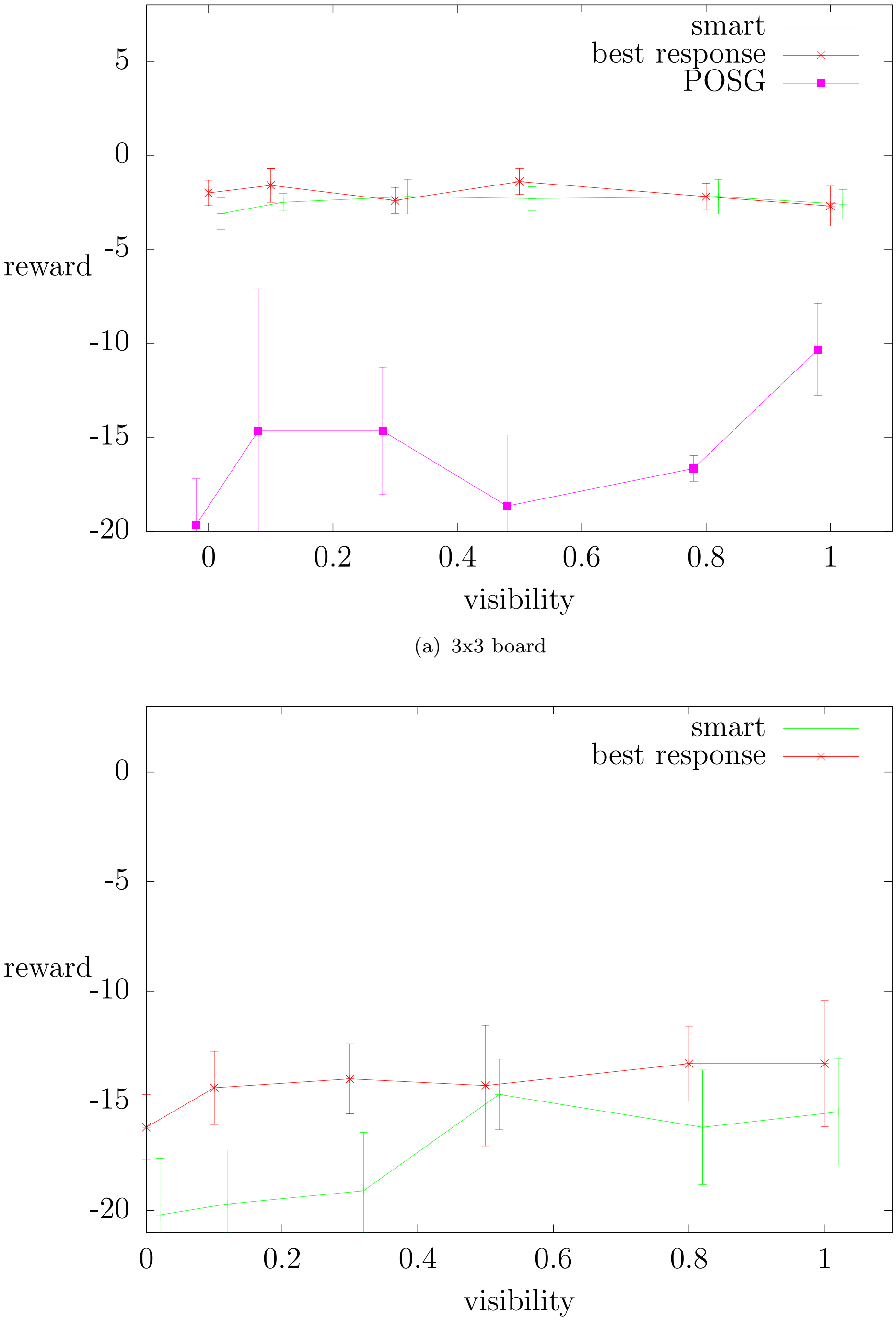
It is immediately clear from figure 6.8 that the best response algorithm is outperforming the smart policy for these parameters. Now, if our algorithm (best response) is benefitting from learning, we expect to see that the advantage the best response algorithm has over the smart (handwritten) policy is increasing over time. From figure 6.8(a) it is not clear that there is a large improvement in this advantage—that is, the lines are fairly straight. However, figure 6.8(b) shows a closeup comparison of the two different sampling rates, showing the way in which the lower sampling rate is able to match the performance of the higher sampling rate after around 800 steps. We therefore see that with better information, the best response algorithm is able to perform well on this problem even without accurate models of the other agents, but when the sampling rate is very low, the best response algorithm is able to compensate for this by learning.

Consequently, it seems that the best response algorithm is performing well primarily on the basis of the sampled best response, rather than accurate estimates of the behaviour of the others being critical. In order to investigate further, we compare the algorithms on some smaller problems which the POSG algorithm is able to run on, first looking at the effects of changing sample rates in more detail, and then varying two parameters relating to the character of the problem (visibility and victim distributions). This allows us to gain insights into the performance of our algorithm as the problem nature is changed. We also investigate parameters relating to the scale of the problem (number of agents, and size of board). For each of these experiments we compare the total reward after 150 steps—from figure 6.8 we can see that this is sufficient to show the differences between the algorithms or settings.

6.2.2.3 Varying the visibility

As the visibility increases and all agents have a better view of the scenario, we expect that the performance of all algorithms will improve. However, we expect the probabilistic algorithms (`POSG` and `best response`) to be at less of a disadvantage than the handwritten policy for the lower visibilities—this is because the handwritten policy always behaves as though the visibility is 100%, and so it does not do any exploration actions.

In more detail, figure 6.9 demonstrates the effects of varying visibility on a 3x3 board and on a 7x7 board, each with three agents. In figure 6.9(a) we see the performance of the `POSG` algorithm is much worse than either the `smart` policy or the `best response` policy and fluctuating at lower visibilities, but noticeably improving as the visibility is increased. However, both the `smart` policy and the `best response` policy do reasonably well even at the lower visibilities, but there is no discernible difference between them. This is because three agents on a three-by-three board can do fairly well using the very simple strategy of digging where they see victims and can probably directly observe most of the board between them. By contrast, figure 6.9(b) shows the performance on the larger board. We do not show the slow `POSG` algorithm on this problem since it does not meet our imposed time constraint (1 minute per step); the baseline of the `null` policy is at around -90. Here, we see that as expected the `best response` policy does outperform the `smart` policy at lower visibility levels, with the `smart` policy approaching the



(a) 3x3 board

(b) 7x7 board

FIGURE 6.9: Effects of varying visibility

performance of the `best response` policy as the visibility increases, although the `best response` policy continues to outperform the `smart` policy.

6.2.2.4 Varying the victim arrival rate

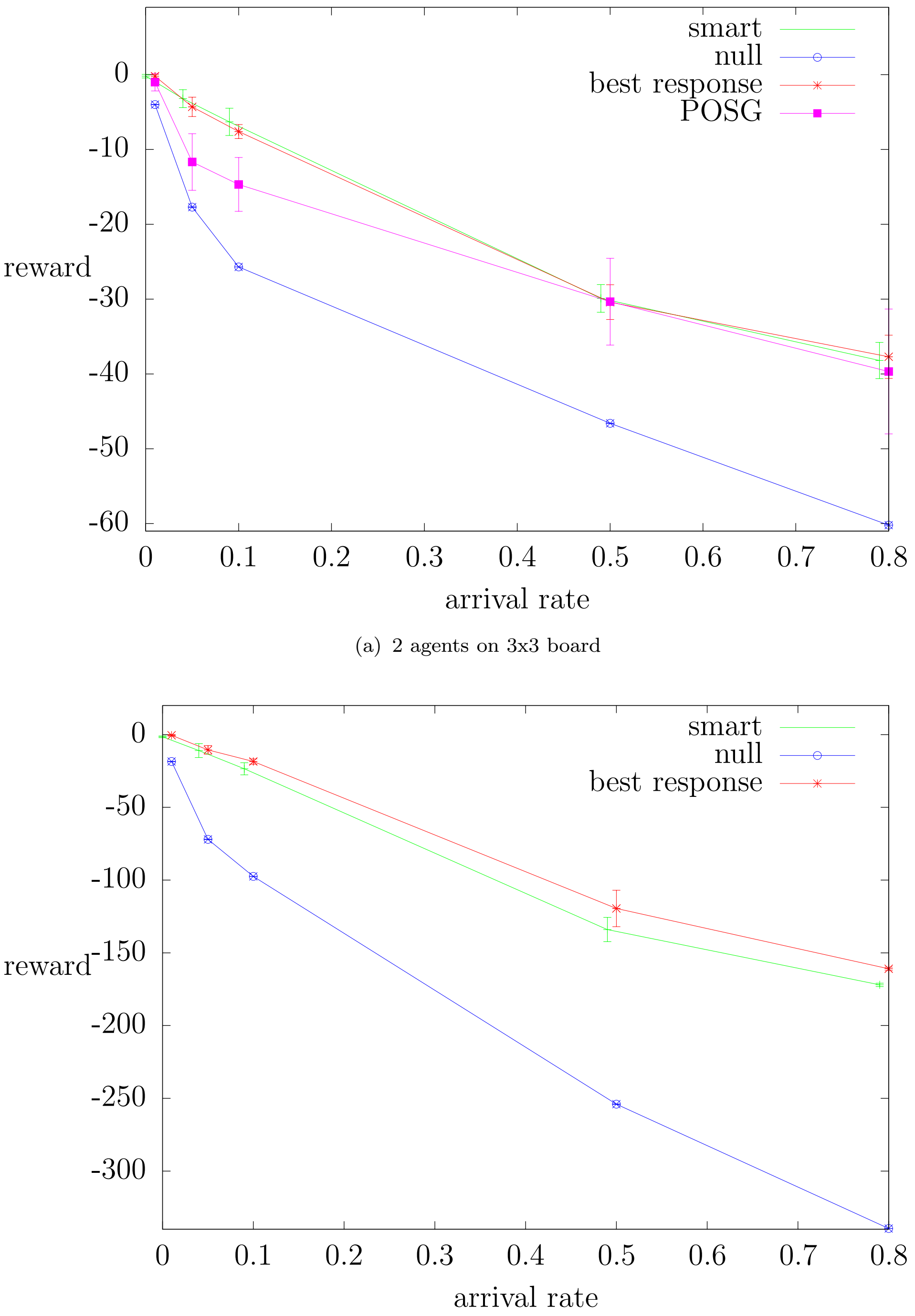
As well as varying the visibility, we can vary the problem by adjusting the victim arrival rate and thus victim density. We expect that increasing the rate at which victims arrive, p_a, and thus the overall density of victims, will make the problem easier, as agents can do well with the simple strategy of digging out the victims around them. As can be seen in figure 6.10, the reward for the `null` policy drops sharply—this is because there are more victims dying.

As the arrival rate increases, causing the victim density to increase, the optimal strategy approaches the very simple strategy of digging if there are any nearby victims. On the smaller problem, we see little improvement in the three policies beyond an arrival rate of around 0.5, indicating that the optimal strategy has been reached by all by this point. On the larger problem (figure 6.10(b)), the `smart` policy and `best response` policy continue to improve across the graph, indicating that there is some sophistication needed in the strategies even at the high victim densities. As expected, on the larger problem, the `best response` policy slightly outperforms the handwritten strategy due to its better handling of the imperfect visibility.

For the next sections, we fix the visibility at 0.5 and the arrival rate at 0.05, as discussed in section 6.2.1. We go on to investigate the scaling properties of the algorithms.

6.2.2.5 Varying problem size factors

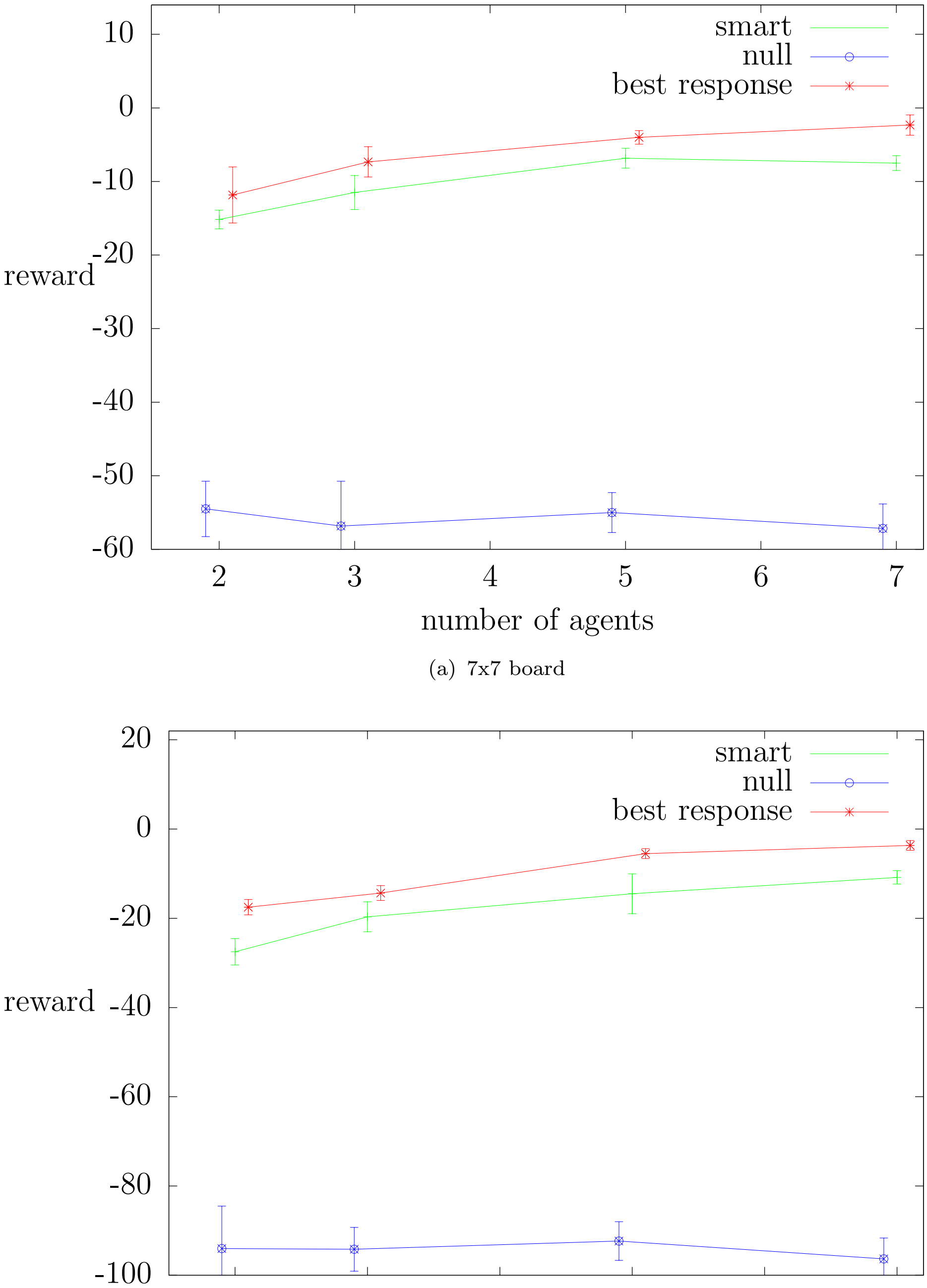
The difficulty of the rescue problem scales exponentially with the size of the board and the number of agents, which are related to the number of states and the number of joint actions respectively. Furthermore, in our implementation, all the agent threads were running on the same machine as one another and the



(a) 2 agents on 3x3 board

(b) 3 agents on 7x7 board

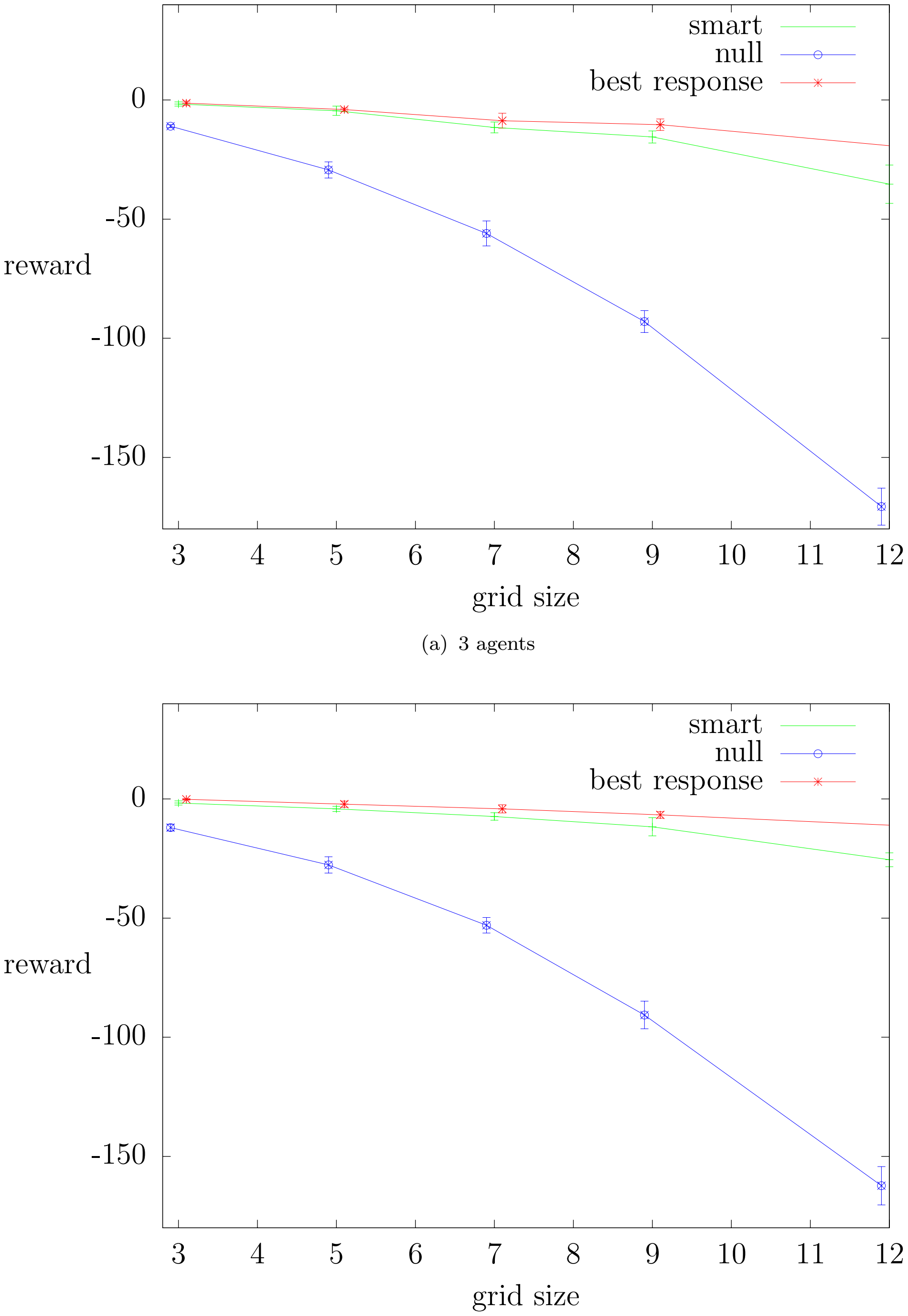
FIGURE 6.10: Effects of varying victim arrival rate



(a) 7x7 board

(b) 9x9 board

FIGURE 6.11: Effects of increasing the number of agents on the results for two large boards



(a) 3 agents

(b) 5 agents

Figure 6.12: Effects of changing the board size on the results for 3 and for 5 agents

environment thread. Since every agent maintains its own belief state, the memory requirements of our implementation scaled linearly with the number of agents. Nonetheless, we were able to test our algorithm on boards of up to 12x12 (2^{173} states), and with up to 7 agents (80,000 joint actions).

Now, although 7 agents is not a huge number for an algorithm which we would like to scale into dozens of agents, the primary limiting factor was the memory requirement for our implementation. In more detail, figure 6.11 shows the effect of increasing the number of agents on two larger boards, a 7x7 board and a 9x9 board. We observe that on the 7x7 board as the number of agents is increased, the `smart` policy appears to saturate while the `best response` policy continues to improve. The results are similar for both the 7x7 and the 9x9 board, although the `smart` policy does not saturate so much on the 9x9 board—the larger problem space provides more room for improvement.

The `best response` algorithm also performs well on the large boards with many millions of states: with five agents nearly all the victims are rescued (the reward does not fall far below 0) even on the largest (12x12) board. The `smart` policy falls away by comparison. This reflects the results we have seen earlier where the `best response` improves over the `smart` policy more as the board size increases, a consequence of the way in which the `best response` policy incorporates uncertainty and the need for search on larger boards. The results are very similar for both three agents (figure 6.12(a)) and five agents (figure 6.12(b)) although, as expected, five agents are able to make more rescues than three (the lines are slightly flatter), since they can parallelise more.

Thus, we have observed that the `best response` algorithm performs well by comparison with a handwritten strategy designed for the same problem, and requiring much less sampling than the `POSG` algorithm to achieve this performance. Furthermore, the `best response` algorithm scales well, solving problems with many states and increasing numbers of agents and improving on the handwritten strategy for these large problems.

6.3 Summary

In summary, we have considered the problem of agent coordination in partially observable systems, and demonstrated an approach to this problem using a Bayesian learning mechanism, extending previous work on learning models of other agents. This approach is effective for a cooperative scenario from the disaster response domain. To emphasise, the novelties in this work lie in an extension of online model-based learning techniques into partially observable domains, using the finite automaton-based algorithm given in section 3.4.3. Thus, we have substantiated the claim in section 1.4 that by using finite state machines to model the policies of other agents we can extend online learning techniques into millions of states and tens of agents.

To this end, we have examined the performance of this algorithm on a rescue problem with respect to differing problem parameters, finding that its performance consistently outperforms a handwritten strategy for this problem, more noticeably so as the number of agents and the number of states involved in the problem increase. We also observe that reducing the sampling rate of our algorithm has only small effects on its performance, indicating that the best response calculation is the most important feature—this is encouraging, as it enables us to use the best response algorithm with few samples, resulting in greater efficiency. However, we have commented that the limiting factor in running our algorithm, particularly as the number of agents increases, is the memory usage of our implementation, rather than the per-step time required.

To mitigate this limiting factor, we have investigated state clustering as described in section 3.4.2 to reduce the number of abstract states considered by the agent, showing that the use of clustering can improve the efficiency of the algorithm without compromising its effectiveness, and furthermore, a judicious choice of cluster sizes can in fact give better results for a particular problem size.

Although the work described above is encouraging, there remain a number of areas in which improvement can be made. As well as scaling the model into higher numbers of agents and larger state spaces, using a more efficient implementation for the environment and agents, and running the agents on distributed machines, there are improvements which can be made to the model. In particular, we propose to move beyond the scope of the current work, considering cases in which the environmental dynamics are unknown or are changing, and in which agents are able to enter and leave the environment as the problem progresses. As discussed in section 3.2, the algorithm we have presented can in principle be used to learn fixed parameters such as parts of the environmental dynamics, by treating these parameters as a part of a “grand state” from which observations are made. This is the subject of the next chapter.

Chapter 7

Coordination in the presence of dynamism and openness

In this chapter we substantiate the claim in section 1.4 that our model-based solution is able to handle changes in the world effectively, something which existing related models do not consider. To this end, we take the model of chapter 3 for the specific case in which the actions of the other agents and the transition function are known, while the state is only partially observable, and the behaviours of other agents are known, and explain how it can be applied to dynamic and open domains.

In more detail, in this chapter we experiment with making changes in the environmental dynamics during the scenario (section 7.2). This change may be either a substantial instantaneous change (modelling, for example, an aftershock in an earthquake) or a gradual change (i.e., a very small change at each timestep, for many steps: this models situations where, for example, a spreading fire is gradually worsening an earthquake scenario or natural evaporation is gradually improving a flood scenario).

We then experiment with open domains (section 7.3); adding new agents as the scenario progresses, or removing some of the agents while the rest continue. In every case, we assume that all the agents are aware of the new situation immediately. This simplifying assumption is necessary for conformity with our model, although generally unrealistic in the real world, and in section 8.3.5 we sketch an extension to our model which would allow agents to infer when the environment has changed.

7.1 Modelling dynamic and open domains

While the problems of open domains and dynamic domains are in general quite different problems, the formulation we have described allows for both to be treated in the same way. Therefore, we discuss them together here. Specifically, we will use the model described in section 3.2, with known dynamics, partially observable states and partially observable actions.

To include dynamic domains within this model, at specified points during a simulation (not known to the agent beforehand), the environmental model will be changed. We assume that the agent knows about the change and the new dynamics immediately. Similarly, we include open domains by introducing an agent to the environment at one or two points during the simulation, or removing agents from the environment. Again, we assume that our agent knows about the change immediately. As discussed in chapter 1, in both cases assuming perfect knowledge may be unrealistic, but investigating these cases provides stepping stones to future work with uncertain information.

Within our framework, both cases are straightforward to model. The beliefs that have been generated at each step are considered to be prior beliefs for the next step, so there is no difficulty with simply changing the model at a particular timestep. Over the next several timesteps the agent will gradually readjust to the new model. Similarly, it is no problem to remove an agent from our agent's model, or add a new one. In adding a new agent we have a choice of either using a uniform prior over its behaviour and belief state, or making some assumptions about the agent based on the other agents in the system: for simplicity we will use the uniform priors.

In the rest of this chapter we describe how we evaluate these two models using dynamic (section 7.2) and open (section 7.3) versions of the ambulance rescue problem. For both cases, as in chapter 6, we compare our strategy against the Bayesian game approximation using the finite-horizon approximation technique (Emery-Montemerlo et al., 2004) ("POSG") on small problems, and against our handwritten strategy ("smart"). However, while previously our interest was in the relative success of the strategies, in this section our interest is in how effectively they adapt to the changing circumstances. Now, when environmental changes occur, both the smart policy and the POSG policy should adapt instantly as they are doing no learning. Both policies should therefore demonstrate optimal adjustments.

7.2 Ambulance rescue in dynamic domains

We carry out two classes of experiments: single changes to the environment, both instantaneous and gradual, and multiple changes to the environment, changing the settings in one direction and then returning them to their original values. In each set of experiments we look at changes which make the problem more difficult, and changes which make the problem easier. The next section outlines our experimental setup in more detail and then section 7.2.2 provides our results.

7.2.1 Experimental setup

We keep $l_d = l_r = 4$ and v=0.5, as before. We experiment on a smaller problem with $m = n = 5$ and k (the number of agents) = 2, in order to compare against the slow POSG policy, and on the medium problem with $m = n = 7$ and $k = 3$ in order to see how our strategy behaves in larger problems.

To work with our approximations, we also have to fix the maximum number of clusters, the length of observation strings, and the sampling rate. Based on the results from chapter 6, we set the number of clusters to 200, the length of observation strings at 25 observations, and the sampling rate at 25. For each of these parameters, the solution time increases at least linearly and we can see from section 6.2.2.1 that these choices are sufficient to find good solutions without being overly demanding of time. In particular, we discovered that surprisingly few observation clusters are necessary in order to achieve good results. Furthermore, having few observation clusters, thus restricting the possible FSMs, results in agents quickly converging on a particular choice of FSMs and thus stabilising their own strategies.

Finally, in each experiment we adjust the (death rate, unbury rate, arrival rate) parameters (p_d, p_r, p_a). Increasing the death rate makes the problem harder for the agent (civilians die faster if they are not rescued), as does lowering the unbury rate (it takes longer to effect a single rescue). Increasing the arrival rate was investigated in chapter 6, where we showed that an increased arrival rate makes it easier for the agent to find a strategy, because there are more civilians nearby and it becomes effective just to dig on the spot. However, as the arrival rate increases, the death rate and so rewards are lower even though the "good" strategy is easier to find.

This leads to the following parameter choices:

Single, one off change: Here, we begin by taking a combination of parameters which we have previously seen is challenging for all strategies and requires cooperation between agents to be really successful (0.35, 0.25, 0.2), and make a single change which makes the problem much easier. However, although the new version of the problem requires less cooperation during individual rescues, it is now more important for the agents to spread out effectively (when the arrival rate is 0.2 the scenario comes to have a high density of victims, meaning that a strategy of digging locally does well). We also invert the problem, starting with the easier strategy and moving to the harder one. These parameters were chosen in order to show a clear distinction between the two parts of the problem, while having the easier problem not so easy that all the victims can be rescued all the time:

	Getting easier:	**Getting harder:**
p_d	$0.35 \rightarrow 0.25$	$0.25 \rightarrow 0.35$
p_r	$0.25 \rightarrow 0.35$	$0.35 \rightarrow 0.25$
p_a	$0.2 \rightarrow 0.12$	$0.15 \rightarrow 0.23$

Single, gradual change: These gradual changes begin with the same parameters as the oneoff changes. The changes that are made at each step are one-hundredth of the oneoff change. We show the change continuing over 300 steps (150 steps for the small problem) in order to observe the effects of the change over a long time period. This means that by the end of the run, the problem has become much harder (or easier) than the oneoff variant, enabling us to observe the effects of the changes more clearly.

	Getting easier:	**Getting harder:**
p_d	$0.35 \rightarrow_{-0.001/step} -0.05$ (300 steps)	$0.25 \rightarrow_{0.001/step} 0.55$ (300 steps)
p_r	$0.25 \rightarrow_{0.001/step} 0.55$ (300 steps)	$0.35 \rightarrow_{-0.001/step} 0.05$ (300 steps)
p_a	$0.2 \rightarrow_{-0.0005/step} 0.05$ (300 steps)	$0.15 \rightarrow_{0.0005/step} 0.3$ (300 steps)

Double, one off change: These runs are similar to the oneoff changes described above and we use the same choices of parameters. However, after making the problem easier, we then make it harder, to see how the agents are able to cope with making the adjustment when they have previously adjusted the other way. We allow the same time (300 steps for the medium problem) between the two changes.

	Getting easier, then harder:	**Getting harder, then easier:**
p_d	$0.35 \rightarrow 0.25 \rightarrow 0.35$	$0.25 \rightarrow 0.35 \rightarrow 0.25$
p_r	$0.25 \rightarrow 0.35 \rightarrow 0.25$	$0.35 \rightarrow 0.25 \rightarrow 0.35$
p_a	$0.2 \rightarrow 0.12 \rightarrow 0.2$	$0.15 \rightarrow 0.23 \rightarrow 0.15$

Double, gradual change: Finally, we repeat the gradual changes with the adjustment taken back the other way after 300 steps (150 for the smaller problem). For these results we show another 300 steps of the problem after the parameters have returned to their original settings, taking the parameter beyond its initial setting. We do this because we found that we can see more of what is happening with the longer runs.

	Getting easier, then harder:	**Getting harder, then easier:**
p_d	$0.35 \rightarrow_{-0.001/step} -0.05$	$0.25 \rightarrow_{0.001/step} 0.55$
	$\rightarrow_{0.001/step} 0.65$ (300 steps, 600 steps)	$\rightarrow_{-0.001/step} -0.05$ (300 steps, 600 steps)
p_r	$0.25 \rightarrow_{0.001/step} 0.55$	$0.35 \rightarrow_{-0.001/step} 0.05$
	$\rightarrow_{-0.001/step} -0.05$ (300 steps, 600 steps)	$\rightarrow_{0.001/step} 0.05$ (300 steps, 600 steps)
p_a	$0.2 \rightarrow_{-0.0005/step} 0.05$	$0.15 \rightarrow_{0.0005/step} 0.3$
	$\rightarrow_{0.0005/step} 0.35$ (300 steps, 600 steps)	$\rightarrow_{-0.0005/step} 0$ (300 steps, 600 steps)

For all of these experiments, we performed runs on the medium problem, a 7x7 board with three agents, thereby investigating the effect of a problem which has millions of states and some scope for coordination. For the medium problem, we tested our `best response` strategy against the handwritten `smart` strategy. In each case, we began the environmental changes after 300 steps, giving the agents plenty of time to settle into a strategy, particularly given the parameter choices discussed above.

We also felt it would be useful to observe the `POSG` strategy in these domains. As shown in chapter 6, the `POSG` strategy is impractical to run on the medium problem. Therefore, we did a set of tests on a smaller problem, with a 5x5 board and two agents. For this problem, because there are far fewer states and consequently far fewer possible observations, less "settling down" time is needed and so we ran the experiments for half the time, bringing in the environmental changes after 150 steps. We note that, as shown previously, setting the sampling rate to 25 rather than, say, 50 or 100 has a more noticeable effect on the `POSG` strategy than on either of the other strategies, and so throughout the `POSG` strategy, which we will see performing less well than the others, could be improved relative to the other two by increasing the number of samples (although this improvement would come with a time penalty exponential to the number of samples and thus the rate of improvement).

As before, we carried out several runs of the problem, varying the initial placement of civilians and randomising their arrival and visibility. The same random seed was used to initialise each of the test algorithms in each run. The next section explains our key results. The error bars included in the results show the 95% confidence intervals around each point; in these tests we have shown the reward during the course of an averaged run of several hundred steps. For ease of reading, we have only shown the error bars every 20 steps, and we have staggered the choice of steps for the different algorithms.

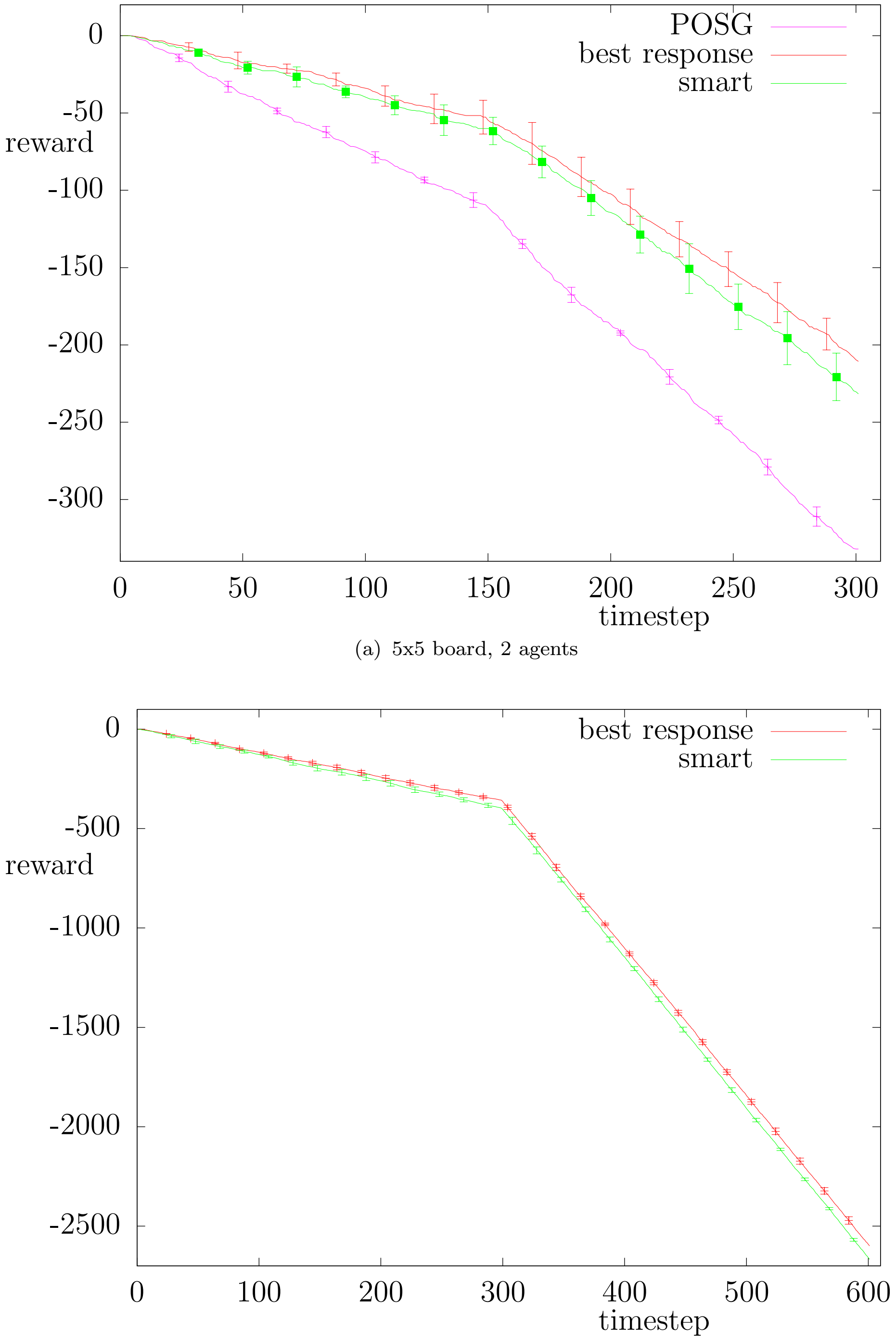
7.2.2 Experimental evaluation

In this section we show the results of the experiments described above, in the classes given: a single oneoff change (section 7.2.2.1); two oneoff changes (section 7.2.2.2) and finally two gradual changes (section 7.2.2.3), showing the similarities and difference between these different variants. We do not show the single gradual change because of its similarities to the double gradual change. The relationship between single and multiple changes is shown over the first two sections for the oneoff changes, and we feel it would be needlessly repetitive to rehash it over the gradual changes.

7.2.2.1 Single, one off changes

The simplest experiments we carried out make a single, one-off change to the environmental dynamics, which is immediately known to all agents. To this end, figure 7.1 shows averaged runs when the change makes the problem harder and figure 7.2 shows the averaged runs when the change makes the problem easier. With each of these changes, we expect to see the `smart` policy and the `POSG` policy immediately adapting to the new dynamics, turning a corner quickly, since neither of these policies does any form of learning.

By contrast, we expect to see the `best response` policy doing some learning as it starts before settling into a straight line, and then more learning when



(a) 5x5 board, 2 agents

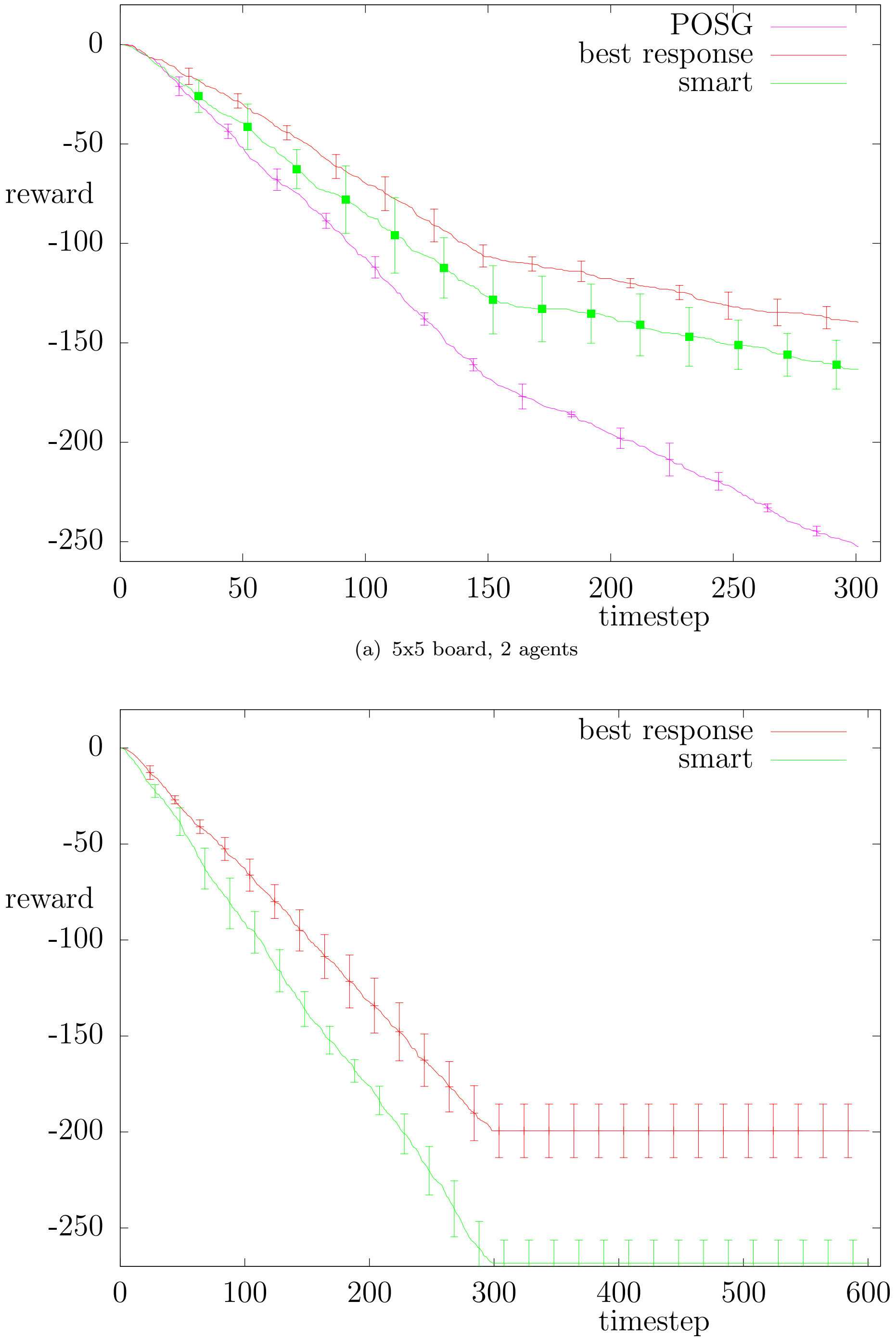
(b) 7x7 board, 3 agents

Figure 7.1: Single, one off change in the environmental dynamics, making the problem harder

the environmental dynamics change. In fact, we see that for the problem and parameters we have chosen, with the small clustering value and relatively long observation strings resulting in the `best response` policy turning the corner as quickly as the `smart` policy. In fact, it is the `POSG` policy which shows a very slight curvature. This is because the `POSG` relies on sampling from the belief state, which takes a few steps to "catch up" with the new dynamics. Although the other policies sample from the belief state, it transpires that because they rely less on the other agents to behave according to shared knowledge, with more unknown information they simply perform a best response to "random", which turns out to be quite an effective strategy on this problem. The effects are even more marked on the medium sized problem.

Another effect which is visible in both cases, but particularly on the medium problem, is that when the problem is made easier after 150 steps, the agents do better than they did on the same variant of the problem after timestep 0. This is a consequence of: (1) the agents having spread out across the board, getting themselves into advantageous positions; (2) the agents having built up belief states over the previous steps, so that at the change point they have a good idea of the situation; (3) in the case of the `best response` policy, this belief state including some knowledge about the behaviour of the other agents. This last point is a small contributor because all the agents must re-adapt their behaviour according to the new dynamics.

In the next section we consider what happens when having changed the scenario in one direction, it is then changed back in the other direction.



(a) 5x5 board, 2 agents

(b) 7x7 board, 3 agents

FIGURE 7.2: Single, one off change in the environmental dynamics, making the problem easier

7.2.2.2 Multiple, oneoff changes

Figures 7.3 and 7.4 show the effects of carrying out a second change in the environment, restoring it to its initial value. We expect that the behaviour will be the same for particular parameter settings, whether the parameters are brought in as the first change to a scenario or the second. The only difference we expect to see is that the advantages of a developed belief state and good position will have been reaped on the first change and will not be improved upon when the environment is changed again.

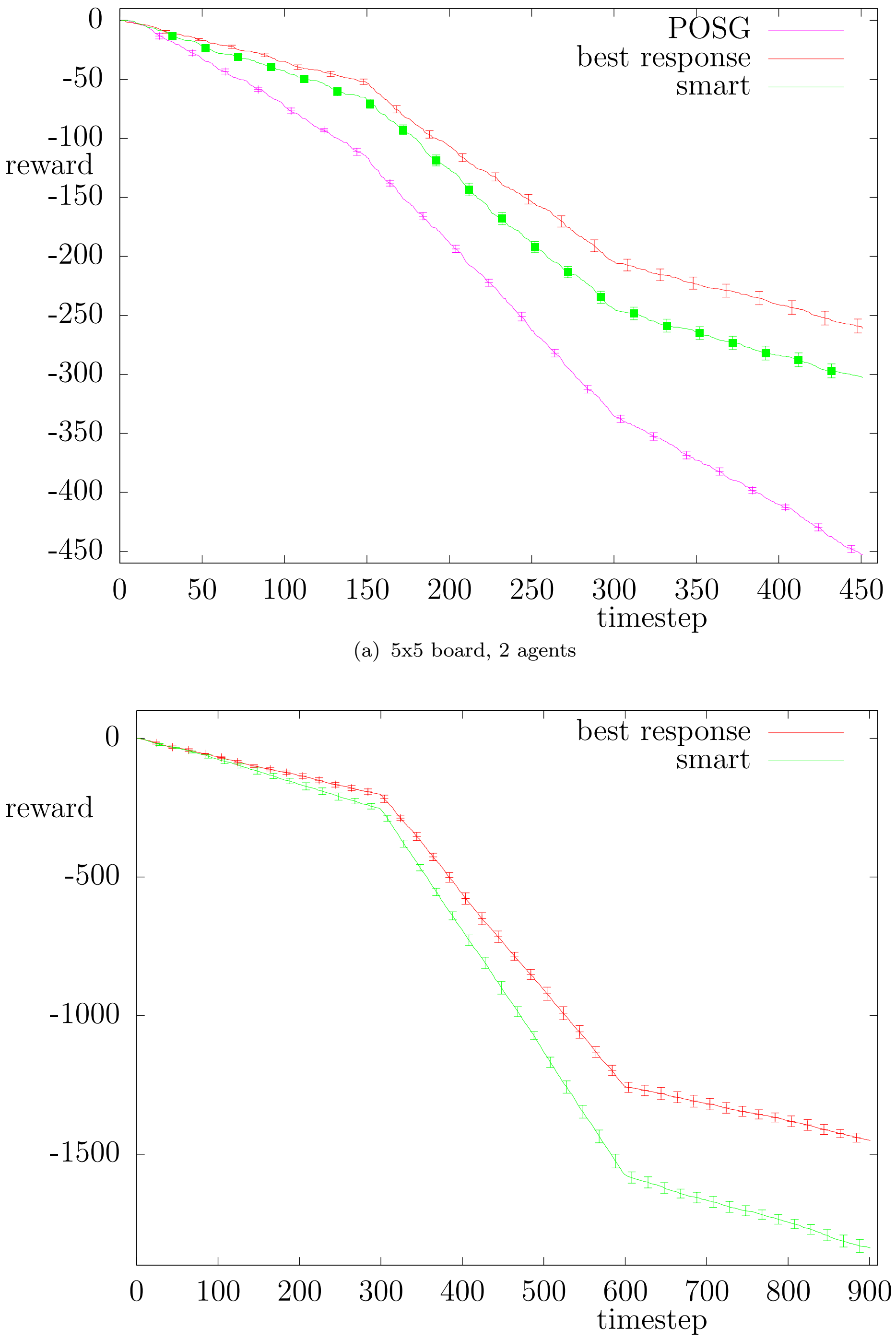
In fact, in figure 7.3 we see in the small problem that the `best response` policy has learned more about the situation between the two changes. Consequently, its performance is noticeably better on the second iteration of the "easy" problem, particularly for the small variant of the problem, for which there is less to learn.

Another effect which we see in the small variant of the problem is that, as with the single change, the `POSG` policy does not adapt as well to the more difficult environment (figure 7.4)—this is just because the `POSG` policy is doing less well anyway and the effect is more marked when the problem becomes more difficult.

Now, given that we have shown in the previous sections the advantages of a good belief state, we consider in the next section what happens when the dynamics are changing continually, so that agents cannot maintain such a good belief state.

7.2.2.3 Multiple, gradual changes

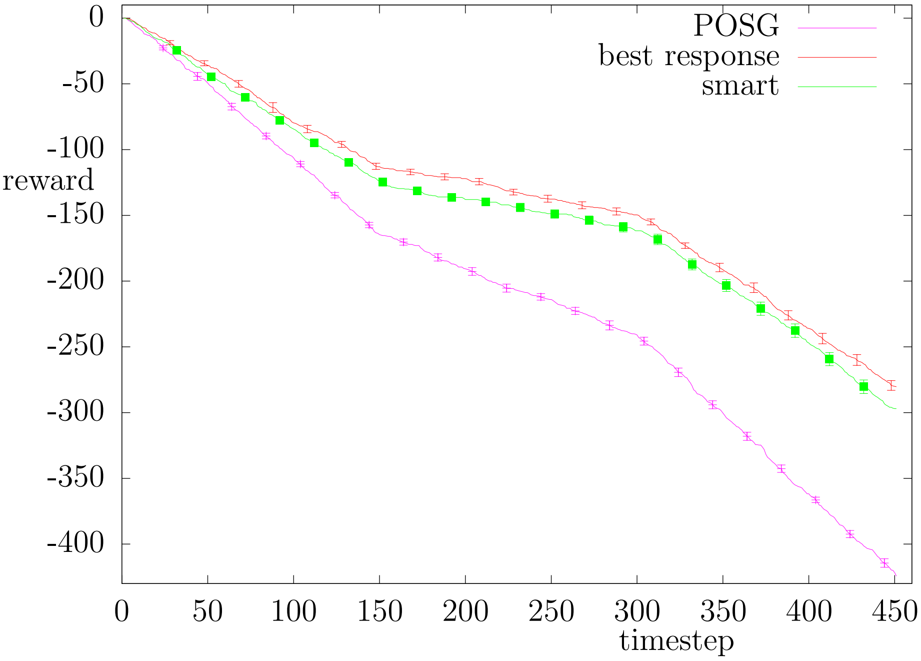
Figures 7.5 and 7.6 show the effects of changing the agents' environment gradually. Specifically, in figure 7.5, we show the agents' environment getting gradually more difficult over three hundred steps (150 for the small problem), and then improving again for six hundred steps (300 for the small problem), so that after three hundred steps (150 for the small problem) the environment is back at its starting point and after another three hundred (150 for the small problem) it is much easier than it was to start with.



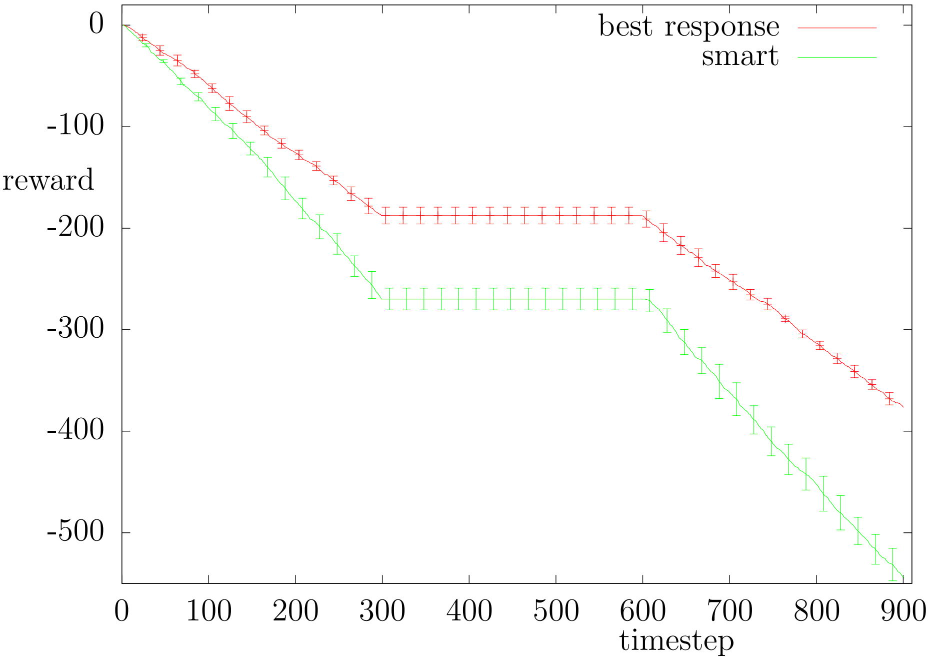
(a) 5x5 board, 2 agents

(b) 7x7 board, 3 agents

FIGURE 7.3: One off changes in the environmental dynamics, making the problem harder and then easier

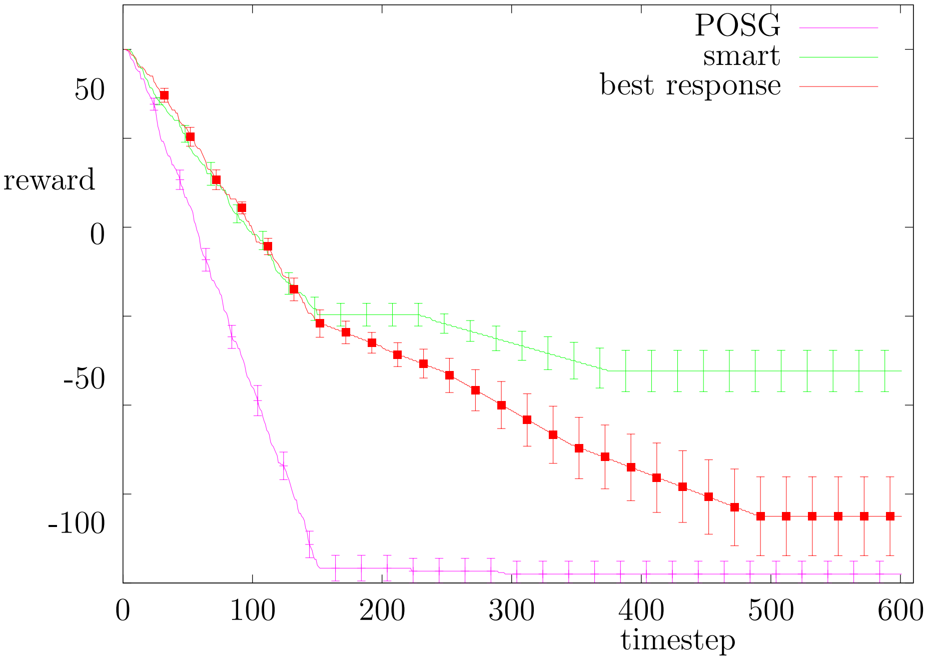


(a) 5x5 board, 2 agents

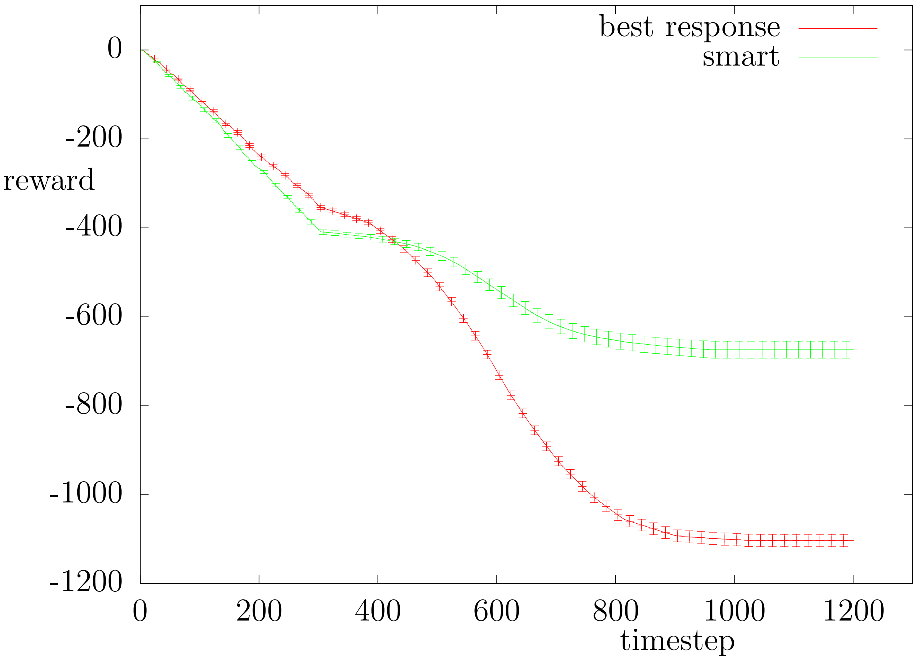


(b) 7x7 board, 3 agents

Figure 7.4: One off changes in the environmental dynamics, making the problem easier and then harder



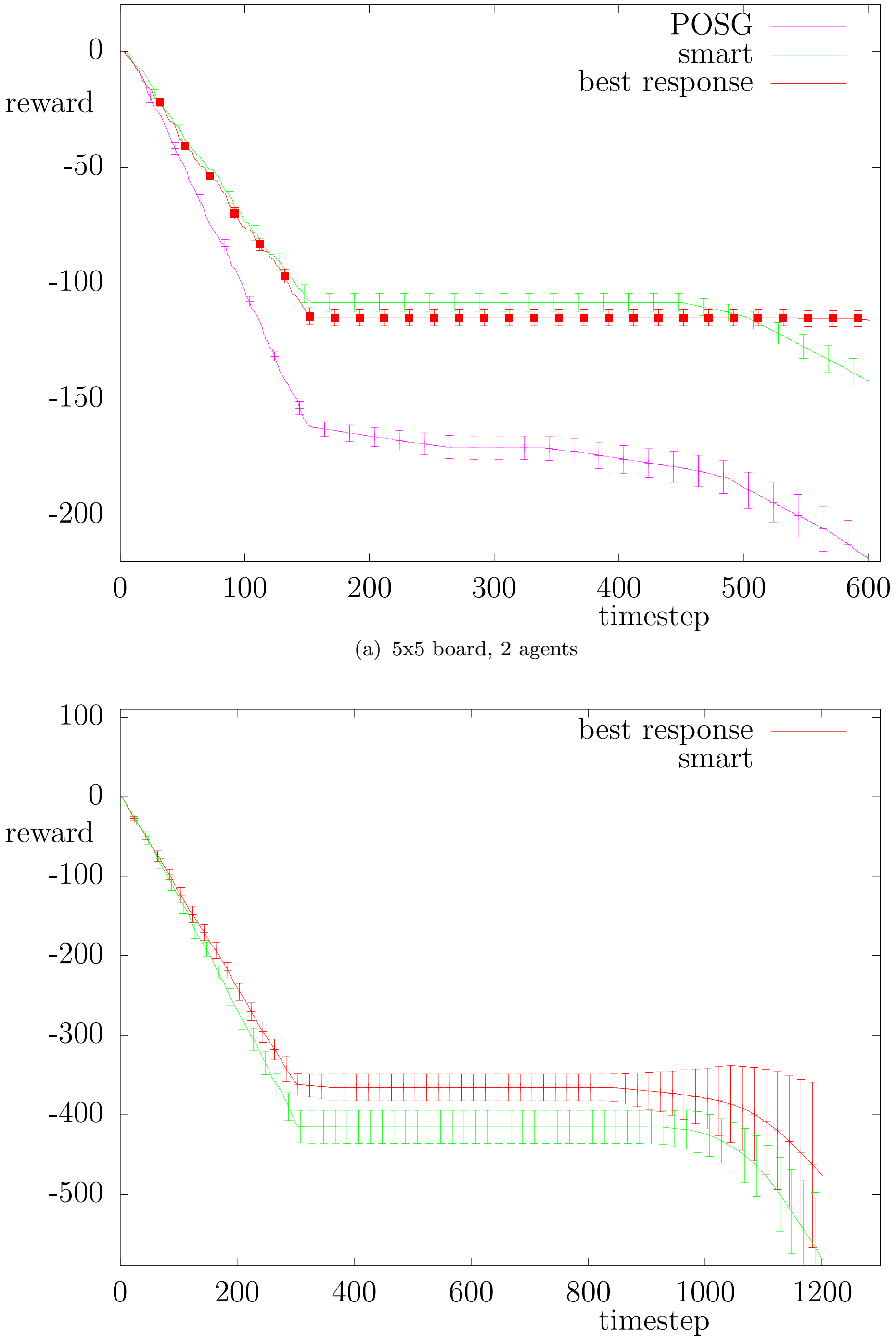
(a) 5x5 board, 2 agents



(b) 7x7 board, 3 agents

Figure 7.5: Gradual changes in the environmental dynamics, making the problem harder and then easier

We expect to see, for all policies, that as the environment gets more difficult the rewards drop increasingly quickly. Furthermore, since the agents will find it hard to track an accurate belief state with the changing environment (even though



(a) 5x5 board, 2 agents

(b) 7x7 board, 3 agents

FIGURE 7.6: Gradual changes in the environmental dynamics, making the problem easier and then harder

they know about all the changes), we expect that the performance should become worse than for the same strategies after a oneoff change. Counteracting this, as the problem gets more difficult the agents will tend to spread out more and dig locally more rather than teaming up to rescue urgent victims in more far-off locations: this is because it becomes more challenging for them to reach far-off locations in time and because there will be more local victims as the arrival rate increases. In turn, this means that although the belief state becomes less accurate for distant victims, it matters less because the agents are not giving much consideration to distant victims in their strategy—particularly in the `best response` policy.

Looking first at figure 7.5, we note the following points:

- For all strategies, as soon as the environment begins to change, the strategy begins to perform better—even though the change makes the environment more difficult. This is an artefact of the probability distributions resetting at that point

- However, as the problem continues to get more difficult, the strategies continue to perform well. In the small problem, there is very little worsening as the problem becomes increasingly difficult, while in the large problem both policies can be seen getting gradually worse, but particularly the `best response` policy which, contrary to expectations, does not handle the more difficult environment well. This indicates that as the problem gets more difficult, the effects of a good belief state (which is lost during the gradual changes) are more important than the effects of a localised digging strategy, particularly on the larger problem.

- When, at 300 steps on the small problem and 600 steps on the large one, the problem difficulty reverts, the `best response` strategy improves again at about the same rate it started to fail. By the time the values return to their original values, both the `smart` and `best response` have managed to manoeuvre to positions where they are rescuing all the victims.

When the change is made in the other direction, making the problem easier, we see (figure 7.6):

- The easing and then worsening of the problem can be seen most clearly for the `POSG` policy on the small problem. This begins to improve immediately the change is made at 150 steps and then improves more gradually to be saving all victims by 300 steps, when the environment is worsened again.
- As the environment returns to its original value and then continues to become more difficult, all the policies do much better than they did on the same environment over the first 150 steps, but in this case the `best response` policy manages better than the `smart` policy; the inverse of the previous section.

Overall, we have seen that generally, after settling into the problem, the agents do better than they did initially. This is because they have developed more accurate belief states and, particularly, better positioning. The effects of better positioning are noticeable especially when the problem is made more difficult, forcing the agents to spread out, and then eased off again. In this case, the agents do much better on the easier problem than they were doing initially. We note from this that the agents initially are not finding the best strategy for the scenario, but do not seem to be improving their strategies much until forced to by environmental changes. Consequently, future work could try and find ways of improving on our policies to allow them to do so.

In the next section, we investigate a related problem where the environmental dynamics remain constant, but agents may enter and leave the scenario during the run of the problem.

7.3 Ambulance rescue problem in open domains

As with the dynamic domains, we experimented both with single changes (a group of agents entering or leaving the scenario) and multiple changes (when agents may come or go at several points during the scenario). To this end, in section 7.3.1 we describe our experimental setup in more detail before going on to give and explain our results in section 7.3.2.

7.3.1 Experimental setup

In section 6.2.2.5 we noticed that on a 7x7 board the `smart` policy does not make good use of the larger numbers of agents, with performance being just the same for five agents as seven. Even with a 9x9 board, the improvements with larger numbers of agents are small. We therefore took the 9x9 board as our "medium" problem, rather than the 7x7 board, and also investigated a "small" problem of 5x5 and a larger problem of 15x15.

Now, although we previously compared with the `POSG` policy, in these experiments doing so was not possible, for two reasons. Firstly, the `POSG` policy has time requirements exponential in the number of agents and so it is not feasible to run experiments with more than three agents. Secondly, the `POSG` policy is not designed for open domains. All the agents are assumed to begin with the same shared seed and their belief states are evolved from this shared seed. Consequently, it would be necessary to modify the `POSG` policy for it to function in open domains. We therefore just looked at the `smart` and `best response` policies.

In more detail, keeping all the parameters but the number of agents and the board size set as before, we carried out the following experiments:

Single change: As the problem size increased, we wanted to increase the size of the single change, to show the difference to best effect for that problem. Thus we have a difference of two agents for the 5x5 board, of three agents for the 9x9 board and of four agents for the 15x15 board. Now, in no case do we believe it useful to consider more agents than can fit on one side of the board, since this would be disproportionate for the kinds of scenario we are trying to model, and does not promote search. Therefore, for the 5x5 board we use a maximum of five agents. For the larger boards we go up to seven agents, giving us the following settings:

	Easier	Harder
5x5	3 → 5 agents	5 → 3 agents
9x9	4→7 agents	7→4 agents
15x15	3→7 agents	7→3 agents

Multiple changes: Unlike the dynamic problem where we made the problem better or worse, and then reset it, here we begin with a moderate number of agents, and take the total number to both extremes, so as to see the effects of both small and large changes in the number of agents. For the 5x5 problem, this means beginning with three agents, and both dropping to one agent (no coordination possible at all!) and increasing to five agents. For the biggest board (15x15), we begin with five agents and go to extremes of one agent and seven agents, so as to see the maximum changes. For the medium board (9x9) changes we have not looked at such dramatic changes, so that at least one problem has continual coordination, which is our primary interest in these investigations. In more detail, we use the following settings:

	Easier then harder	Harder then easier
5x5	3→ 5 → 1 agents	3→ 1 → 5 agents
9x9	5→ 7 → 5 → 3 agents	5→ 3 → 5 → 7 agents
15x15	5→ 7 → 3 → 1 agents	5→ 1 → 3 → 7 agents

We make the changes after 150 steps for the small problems and 400 steps for the larger problems (to give the agents more time to settle into larger scenarios). Where there are multiple changes, these occur every 150 steps or every 400 steps. Taking these settings, the next section shows the results of our experiments.

7.3.2 Experimental evaluation

This section gives results for the experiments as outlined above: first single changes (section 7.3.2.1) and then multiple changes (section 7.3.2.2), showing all three problem sizes for each experiment type.

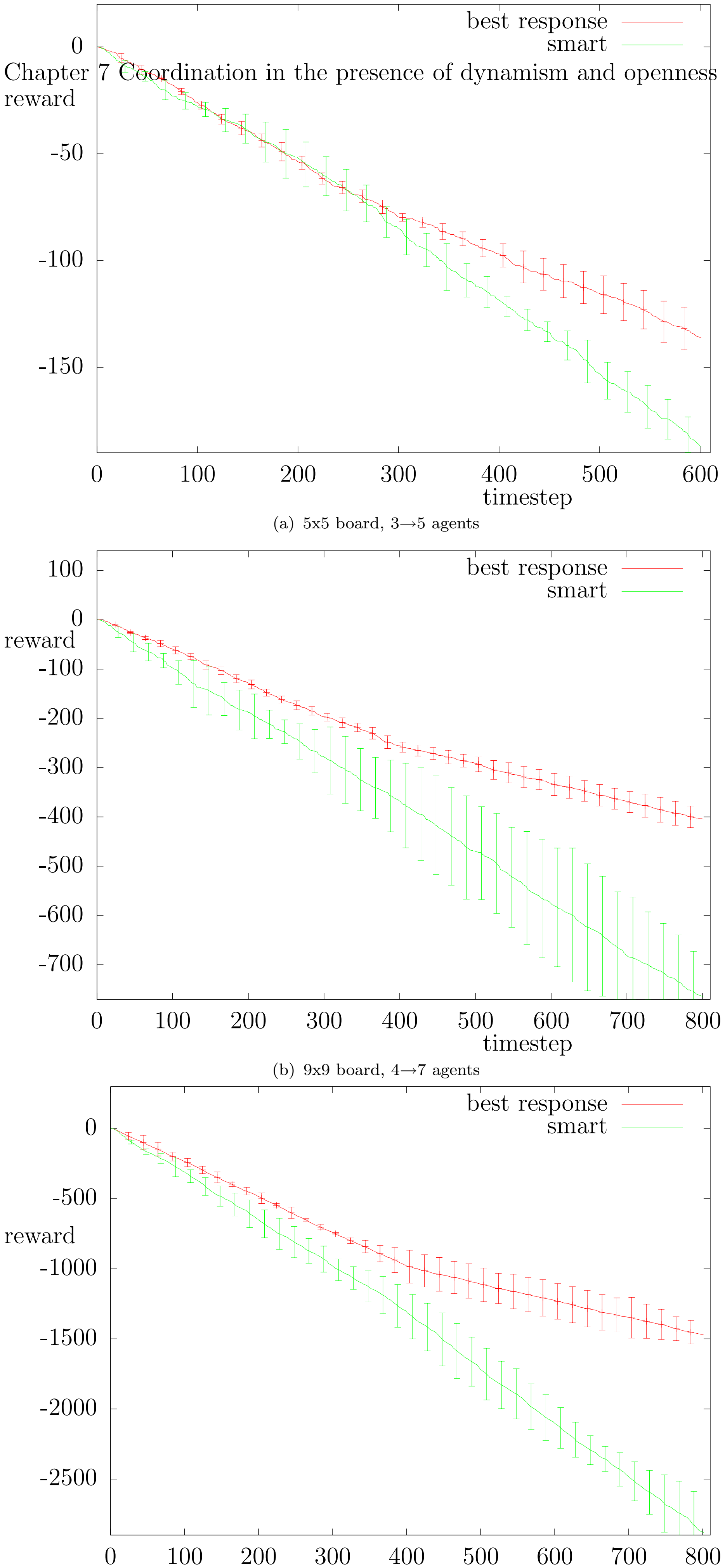
7.3.2.1 Single changes

Figure 7.7 shows the three problem sizes with the number of agents increasing halfway through the run. In each case, we expect to see that as new agents are added, the strategies improve, and in the case of the `best response` policy continue to improve for some time, as the agents adapt to one another.

In fact, for each case we see the same effect on the `best response` policy which responds immediately to the increase in the number of agents. In particular, we notice that the `best response` policy does not need a learning phase to adapt to new agents effectively. Indeed, as we have seen with previous results, there is little or no improvement in the policy over time, partly a consequence of the small number of clusters we are using.

By contrast, the `smart` policy does not improve at all with the addition of new agents: in each case the only evidence of the additions is in the size of the error bars which increase when there are more agents around, indicating that the policies vary more. In fact, the error bars generally increase over time (as the agents all begin in the same place on a blank board, and over time different decisions result in scenarios diverging) so this effect is not particularly relevant.

Figure 7.8 shows the equivalent experiment, beginning with many agents and reducing their number. Since the experiment is similar to the above, but with the increases and decreases interchanged, we expect to see similar graphs, but with the flatter and steeper parts interchanged. As before, there is little change in the results from the `smart` policy when the number of agents is reduced. By contrast, the `best response` policy does less well with fewer agents (as we discovered in the previous chapter). The effect is more noticeable as the board size increases, since on larger boards the `smart` policy manages to make use of more agents (as we showed in section 6.2.2.5).

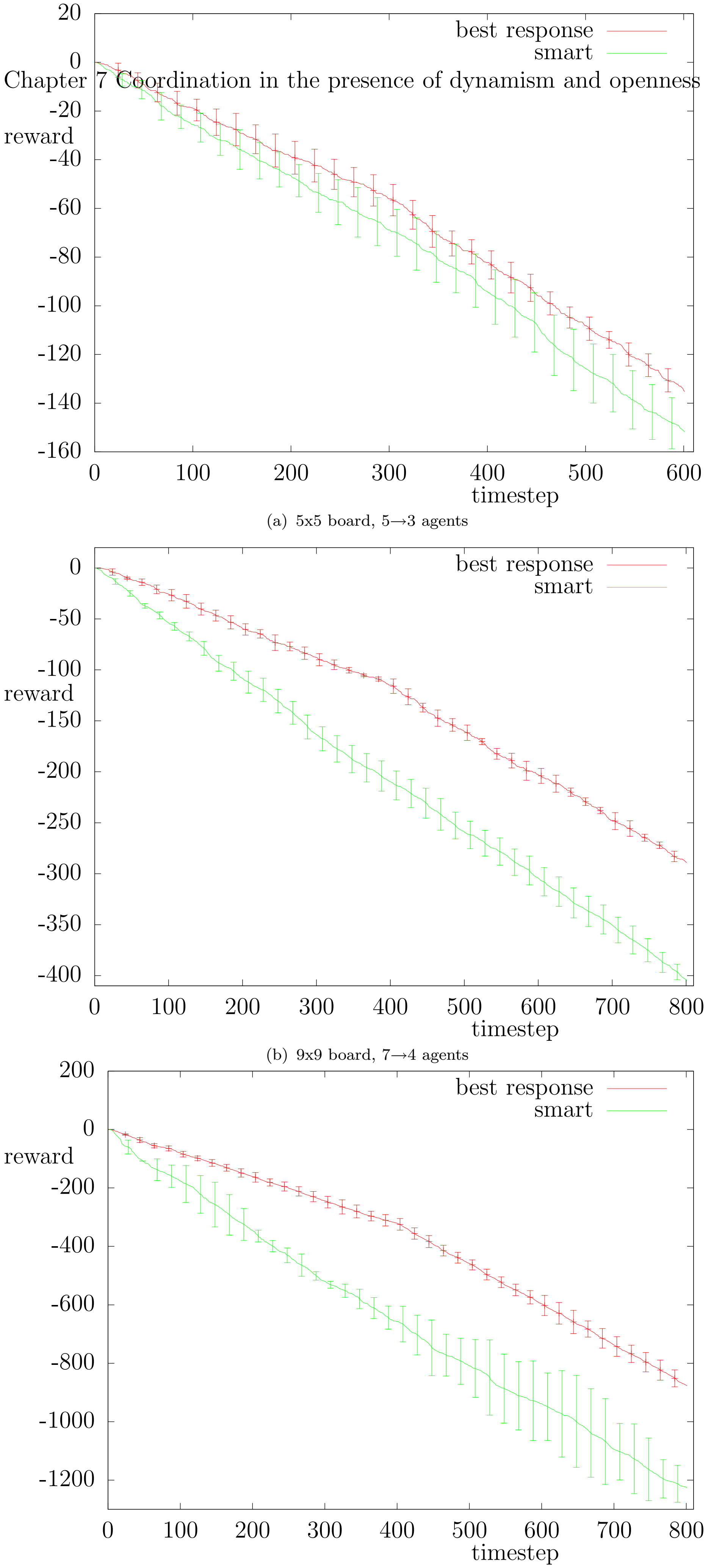


(a) 5x5 board, 3→5 agents

(b) 9x9 board, 4→7 agents

(c) 15x15 board, 3→7 agents

FIGURE 7.7: One off change: increasing the number of agents



(a) 5x5 board, 5→3 agents

(b) 9x9 board, 7→4 agents

(c) 15x15 board, 7→3 agents

FIGURE 7.8: One off change: decreasing the number of agents

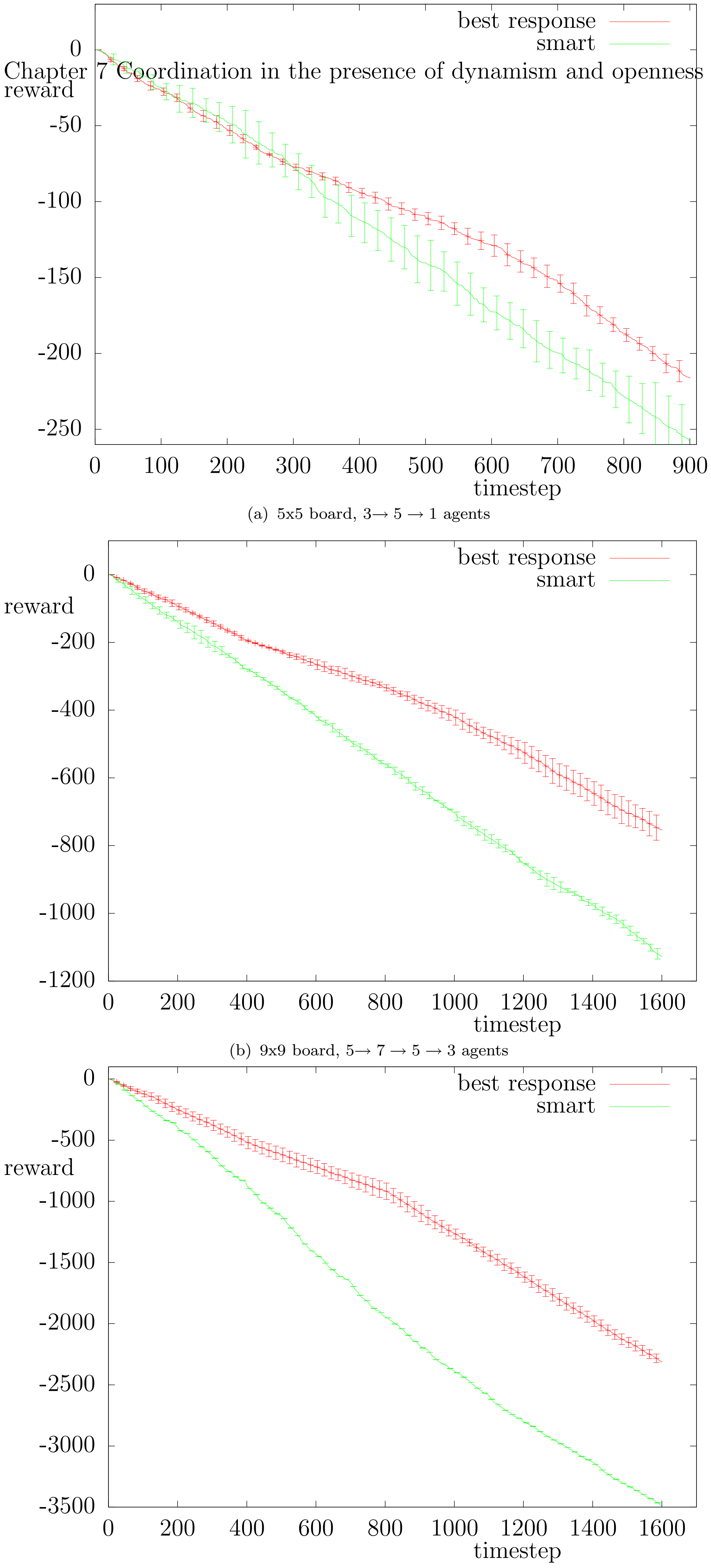
7.3.2.2 Multiple changes

Figure 7.9 shows the effects of changing the number of agents more than once, beginning from a medium number of agents (3 on the 5x5 board and 5 on the two larger boards) and first increasing the number, then decreasing it: on the larger boards the number of agents is decreased twice, more on the 15x15 board to better show the effect of differing numbers of agents on this big board. We expect to see similar results to the above experiments with just one change. Since we did not see much learning or adaptation in the single change experiments we do not expect to see it here either.

In figure 7.9 we see that the `best response` policy behaves as expected. Again, increasing the number of agents results in the team doing better and decreasing the number of agents results in the team doing less well, and again there is little evidence of adaptation. If we look closely at the results we can see that the line is slightly steeper immediately after the changes (at 300 and 600 on the smaller board, and at 400, 800, and 1200 on the larger board) but this effect has disappeared within 20 steps, indicating that the agents spread out immediately.

In figure 7.10, the scenarios begin from the same starting points but remove agents before adding them. As before, the lines for the `best response` policy are roughly proportional to the number of agents present at the time and the change points can be seen as bends in the line. However, it is noticeable that where the number of agents is increased later in the run, there is some curvature of that section of the line. This corresponds to the new agents arriving in board square 0 and gradually spreading out to more useful positions.

As before, the `smart` policy does not respond so well to changes in the number of agents. In figure 7.9 the only noticeable difference in the policy is on the 15x15 board. This agrees with our previous observation that the `smart` policy only succeeds in exploiting larger numbers of agents on much larger boards than the `best response` policy. We note that the rate of negative reward for the `smart` policy is in each case very similar to the rate for the last section of the `best`



(a) 5x5 board, 3→ 5 → 1 agents

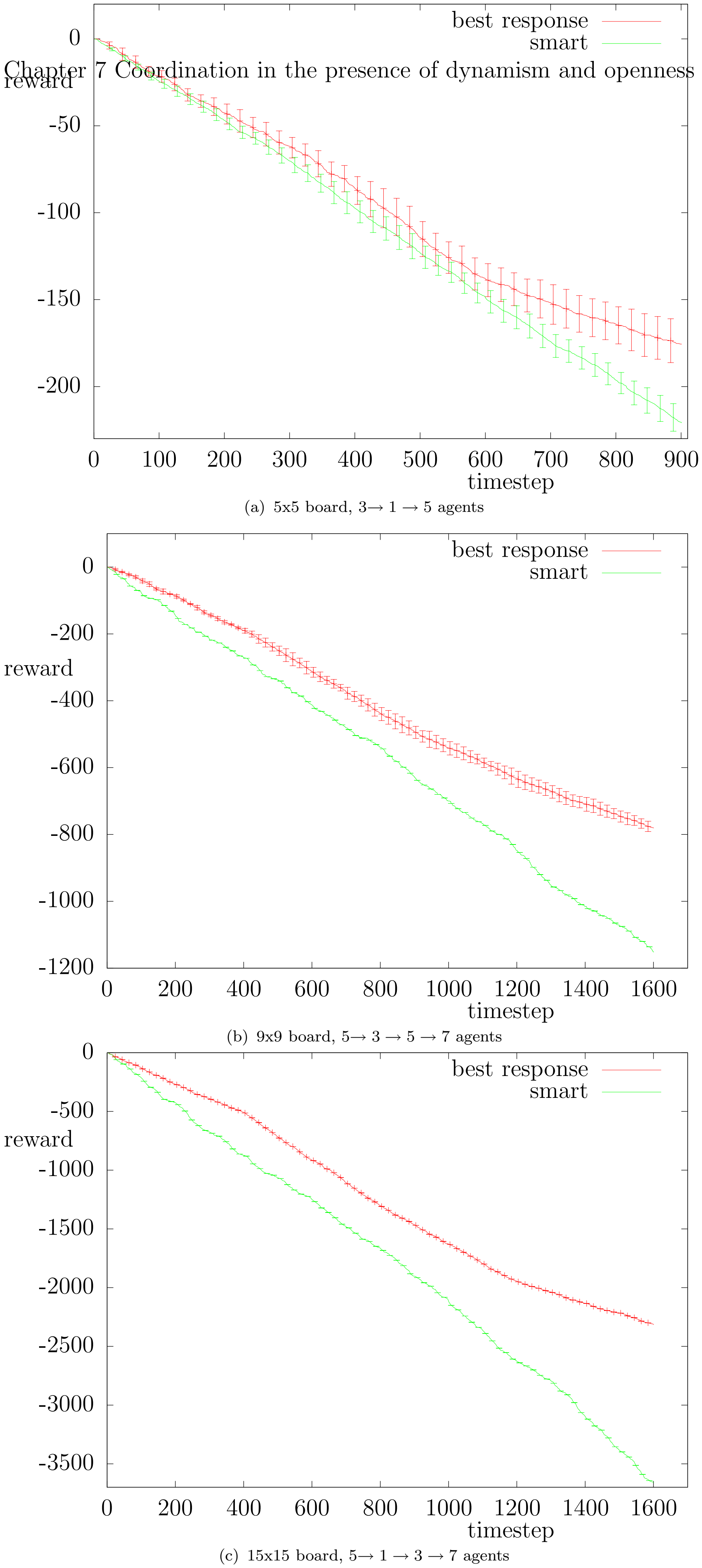
(b) 9x9 board, 5→ 7 → 5 → 3 agents

(c) 15x15 board, 5→ 7 → 3 → 1 agents

FIGURE 7.9: Increasing the number of agents and then decreasing the number

`response` policy, indicating that the `smart` policy just isn't making good use of the additional agents. Similarly, in figure 7.10, although slight bends in the `smart` line can be seen, it is only on the 15x15 board that there is a noticeable change.

Thus, we have observed that firstly the `best response` policy adapts quickly to changes in the environment competing favourably with a handwritten strategy for the same problem. We have also seen that although over time both the `best response` policy and the `smart` policy are able to improve their strategies by improving agent positioning and filling in the agent belief states, there is little learning or adaptation going on in the `best response` policy with the parameters we have chosen, and proposed that future work could investigate this property further. Secondly, we have shown that our `best response` policy, unlike the `POSG` policy, is able to function in open domains, responding as quickly to the additional or removal of agents as a handwritten strategy for the same problem (which in any case performs less well than the `best response` policy for high numbers of agents).



(a) 5x5 board, 3→ 1 → 5 agents

(b) 9x9 board, 5→ 3 → 5 → 7 agents

(c) 15x15 board, 5→ 1 → 3 → 7 agents

FIGURE 7.10: Decreasing the number of agents and then increasing the number

7.4 Summary

In summary, we have considered the problem of agent coordination in dynamic and open domains, and demonstrated that our Bayesian learning algorithm is able to handle both kinds of domain effectively without major adjustments to the algorithm. This is in contrast to the related POSG algorithm described in section 2.3.1 which can handle dynamic domains but not open ones. We have then shown our Bayesian learning algorithm acting in a variety of dynamic and open scenarios. Thus, we have substantiated the claim in section 1.4 that our model-based solution is effective in scenarios with changes in the world: both changes in the environment (dynamic domains) and agents entering and leaving the scene (open domains). This is the first online learning solution to explicitly consider these cases.

In so doing, we have found that our algorithm, like the handwritten solution, responds quickly to changes in the scenarios, with the timing of the change having only small effects on its consequences. We have also noted, firstly, that the effects of good positioning and an up-to-date belief state, catalysed by a change in the environment can result in a better performance than if the change never occurred, and secondly, that there is little adaptation to new agents or learning going on when the scenario is changed. While our algorithm is still outperforming the handwritten strategy, these observations indicate that there may be room for further investigation and we discuss some avenues for such investigation in section 8.3.2.

Having demonstrated that our general model can be practically implemented on a medium scale problem, and is then able to handle partially observable actions (chapter 5), partially observable states (chapter 6) and open and dynamic domains (this chapter), we summarise all these results and conclude the thesis in the next chapter.

Chapter 8

Conclusions

In this chapter we take stock of our work, outlining both what we have achieved, and what avenues have been left open for future investigation. In doing so, this chapter comprises three main sections: in 8.1 we give an overview of what has been discussed so far, summarising the main points from the previous chapters; section 8.2 then gives a more detailed view of the contributions we have made to the state of the art; and finally, section 8.3 discusses the main avenues by which our methods could be extended in the future.

8.1 Thesis Summary

This thesis has considered the problem of scalable agent coordination in uncertain multi-agent systems. Typically, in a large multi-agent system, such as a disaster scenario or a military operation, no agent will be able to see the entire scene or be certain about how the other agents are viewing the scene or planning to behave. This is particularly true if communication is limited, such as in disaster scenarios in which time constraints or network failures may limit communication, or military operations where secrecy is important. In order to coordinate with others in such scenarios, agents must make inferences about one another from their own observations and respond accordingly.

To date, existing work has dealt with such problems either using negotiation techniques within a system structure (Sims, Corkill, & Lesser, 2004) (Giampapa & Sycara, 2001) (Excelente-Toledo & Jennings, 2005) or using techniques based on formal POSG models (Emery-Montemerlo et al., 2004) (Bernstein et al., 2005) (Oliehoek & Vlassis, 2007). Although there has been considerable interest in making the latter tractable, the inherent complexity within such models has meant that techniques which move beyond small problems have been obliged to use domain-specific heuristics in evaluation (Ross, Pineau, et al., 2008).

Despite this, we took a particular interest in work which extends Markov decision process models, as these models provide formal building blocks on which a principled approach can be built, meaning that there is a framework available for guarantees within the system. In particular, Bayesian MDP models such as (Dearden et al., 1999), (Chalkiadakis & Boutilier, 2003), (Emery-Montemerlo et al., 2004), (Ross, Chaib-draa, & Pineau, 2008) offer a principled solution to the exploration-exploitation problem within unknown systems. However, each of these approaches is limited to a specific case. Our first contribution was to develop an approach to this problem using a Bayesian learning mechanism, generalising the previous work on learning models of other agents. Chapter 3 described this model in section 3.2, using Bayesian networks to visualise specific cases of the model and thus as an aid in deriving the update equations for the system.

Furthermore, due to the difficulties associated with their inherent complexity, the previous approaches were not scaled up to the large systems of interest to us. One approach which has been shown to scale well for networked offline problems uses finite state machines to model other agents (Marecki et al., 2008). We used this insight to develop an approximate scalable algorithm applicable to our general model, described in section 3.4.3, in combination with adapting a number of existing approximation techniques including state clustering, described in section 3.4.2.

We have examined the performance of this algorithm on several cases of a rescue problem with respect to differing problem parameters. Specifically, we evaluated first the case where agents are aware of the complete situation, but are not certain about the behaviour of others. That is, our model with all elements observable, except the actions (chapter 5). Secondly, we examined the more complex case where agents can see the actions of others, but cannot see the full state and thus cannot be sure about the belief state of others (chapter 6). Finally, we looked at the performance of this partially-observable state model when the system was dynamic or open (chapter 7).

We found that our best response algorithm consistently outperforms a handwritten strategy for this problem, more noticeably so as the number of agents and the number of states involved in the problem increase. We also observed (section 6.2.2.1) that reducing the sampling rate of our algorithm has only small effects on its performance, indicating that the best response calculation is the most important feature—this is encouraging, as it enables us to use the best response algorithm with few samples, resulting in greater efficiency.

8.2 Research contributions

The main contributions of this thesis are thus twofold. First, we have outlined a model for coordination in multi-agent systems which generalises existing models. Secondly, we have demonstrated the use of this model in three specific cases. We elaborate on these contributions below.

8.2.1 A general model for partially observable multi-agent systems

In more detail, previous work (Chalkiadakis & Boutilier, 2003) (Emery-Montemerlo et al., 2004) (Ross, Chaib-draa, & Pineau, 2008) has described online Bayesian learning models for a number of variants on partially observable systems, evaluating these models on small problems. We explicitly generalised these models to a general Bayesian learning model which can be applied to arbitrary partially-observable multi-agent systems—this is the first such model. However, if nothing is observable, then it will, obviously, be impossible to learn anything. Bayesian network diagrams can be useful in visualising how to apply our model to any particular case of the system, identifying the dependencies between the hidden and observed variables in order to write down a factorised belief update.

Although Bayesian MDP models are theoretically elegant, for most cases they are intractable to evaluate. For each step, the agent must evaluate the Bayesian belief updates, which may be non-trivial, and then solve a best response equation over all belief states. Practically, to solve real-world multi-agent problems, it is necessary to approximate the system or parts of the system. We identified the finite state machine as a model which has been used effectively to approximate agent belief-based strategies in offline systems with partially observable states (Marecki et al., 2008) and incorporated a finite state approximation into our online model. We then extended our model with a statistical clustering technique to reduce the state, or observation, space into a more compact higher level cluster space.

Given this approximate model, we explicitly instantiated, and evaluated our algorithm in three specific partially observable cases. The first, with partially observable actions, has not previously been treated within an explicit Bayesian framework and we demonstrated the efficacy of the Bayesian approach within this model. The second, with partially observable states, has been examined within the model of (Emery-Montemerlo et al., 2004), which we extended. We showed that applying our approximation framework enabled us to efficiently find satisfactory solutions to much larger problems than have been approached with related techniques. The third, applying the model to dynamic and open domains, has not previously been considered within this context and we have shown that we can apply our model to such domains without difficulty.

8.2.2 Partially observable actions

A form of partial observability which has not received much attention in the POMDP literature occurs in scenarios where an agent cannot fully observe the actions of the other agents. The agent may be able to make inferences about other agents, for example through state changes or reward observations. Generally, learning techniques have approached such problems by considering the other agents to be a part of the environment and thus treating the problem as one of learning the environment. However, as argued by (Chalkiadakis & Boutilier, 2003), adaptive agents can have non-Markovian behaviour, so that it is not correct to treat them as part of a Markov environment. Instead, by modelling the other agents separately and explicitly marginalising over the expected behaviour in calculating the value of a state, (Chalkiadakis & Boutilier, 2003) showed that in fully-observable learning environments, it is possible to improve on such single-agent approaches. In this context, in chapter 5 we instantiated a model related to that of (Chalkiadakis & Boutilier, 2003). However, instead of the agent actions observed and the environment unknown, our model has a known environment but partially observable actions. In evaluating this model, which is the first Bayesian learning model to explicitly consider other agents when they are not fully observable, on a rescue problem, we demonstrated that it is effective against a handwritten strategy for the same problem. We also extended the simple form of this instantiation to include inference based on reward observations, again the first Bayesian learning model to do so, and thereby demonstrating the flexibility of model-based systems.

8.2.3 Partially observable states

By contrast, problems with partially observable states (POSGs) have been given considerable attention, although much of this attention is still focused on offline solutions—which are primarily appropriate to short-horizon problems for which an n-stage policy can be found offline. Conversely, our interest is in long-term strategies where it is really necessary to find a short term approximate solution at each step. Since POSGs are so common in the multi-agent world, a litmus test of our general algorithm is its ability to solve a challenging POSG. In chapter 6 we showed how to instantiate our model for the POSG case, and demonstrated its effectiveness and scalability in the rescue domain. Our algorithm is both considerably more scalable and more successful in this problem than the existing state of the art algorithm of (Emery-Montemerlo et al., 2004). Our algorithm also performs better than a handwritten strategy for the rescue problem, particularly on larger problems.

All such POSG models are exponentially complex in the number of agents and we found that, in particular, the memory requirements for our agents formed a limiting factor in scaling our model up beyond medium-sized systems. We investigated observation clustering in an attempt to reduce the memory requirements of each agent, with some success. We also observed that reducing the sampling rate of our algorithm had only small effects on its performance, indicating that the best response calculation is the most important feature—while this allows us to use the best response algorithm with fewer samples, it indicates that the agents are not exploiting the learned FSMs efficiently. Future work should explore the properties of the learned FSMs in more detail and seek ways to improve on this learning.

8.2.4 Open and dynamic domains

One of the advantages of a model-based technique with explicit models for each variable in the system is the ability to exchange or adapt any of these models independently. This allowed us to apply our POSG algorithm to both dynamic and open domains. Previous work on dynamic domains in the learning context has assumed that a learner will adapt to the domain over time, but any guarantees about convergence of the strategy cannot hold (Panait & Luke, 2005). Indeed, in any dynamic system, if the system is changing faster than the learner can adapt, convergence is neither possible nor desirable. Since the online POSG solutions we considered above are, unlike our model, not adaptive, they respond instantly to changes in the environment. However, since our model's performance is primarily a function of its best response calculation, rather than its learned properties (the other agent FSMs), we found that it also responded very quickly to changes in the environment.

Furthermore, open systems have typically been left alone by the multi-agent POMDP community, with algorithms such as (Emery-Montemerlo et al., 2004) (Varakantham et al., 2007) relying on the set of agents remaining consistent through the problem. By contrast, our model is able to handle fluidly agents entering or leaving the system, simply adding new models to its set of agent models or dropping them from the set as appropriate. The best response calculation at the next step operates over the new set of agents, while all agents gradually adapt to the change.

8.3 Future work

Although the work described above has made a number of advances to the state of the art, there remain a number of areas in which improvement can be made. In general, we have highlighted several of these in earlier chapters and here we expand on these. As well as the engineering challenges associated with scaling the model into higher numbers of agents and larger state spaces, using a more efficient implementation for the environment and agents, and running the agents on distributed machines, there are more fundamental improvements which can be made to the model. We discuss each of these in turn below.

8.3.1 Scaling up using sophisticated approximation techniques

In chapter 2 we identified some techniques which could be used to extend our model into larger systems. Each of these techniques adds a layer of approximation to the model, thus trading optimality for efficiency. Further investigation and experiments would be needed to determine the best way to implement or combine these techniques.

Neural networks: neural networks are commonly used to implement function approximation and thus to replace table lookups with parameterised functions (Ren & Williams, 2003) (Baxter & Bartlett, 2000) (Fogel, 2002). A small number of function parameters is learned, rather than many table values (Sutton & Barto, 1998). In the context of our general model (section 3.2), functions could be used to represent any of the unknown models: the environmental dynamics or the other agents' behaviours. In the specific context of the variant instantiated in chapter 5, neural networks could be used to learn about agent behaviours: for each other agent, we would learn a function from state variables to action choice. In the variant instantiated in chapters 6 and 7, finite state machines already provide an approximation over agent behaviours, thus function approximation has no part to play.

Principal components analysis: PCA provides a way to map high-dimensional spaces into lower dimensional spaces. Previous work (Roy & Gordon, 2002) has used PCA to map the belief space into a lower dimensional space. However, the analysis operates over a body of experience and thus PCA cannot be carried out immediately in an online problem. However, we believe that it should be possible to build up some experience during the initial steps and then reduce the number of dimensions used as the problem continues.

Hierarchical learning: Hierarchies provide a way to abstract higher-level information, without losing the detailed information (Hoey, 2001) (Fischer et al., 2004). There are a number of issues involved in exploiting hierarchical abstractions efficiently. In particular: formulating abstractions; deciding what level of the hierarchy is appropriate; constructing strategies across different levels of hierarchies. In our work we have touched on using statistical clustering for abstraction, but future work could extend our abstraction techniques to use abstraction hierarchies.

8.3.2 Finite state machine properties and improvements

As discussed in chapters 6 and 7, our agents only gain limited benefits by maintaining finite state machine models, primarily performing well on the basis of their best response computation. A starting point for future work is, therefore, to investigate the properties of the finite state machines in detail, and to compare their performance across a variety of multi-agent problems, with a view to obtaining a clearer insight on their performance and learning rates.

Related to improving the policy abstractions (finite state machines), future work could investigate action abstractions. Consider again the rescue problem described in chapter 4 which we have evaluated our models on. Considering the domain, a natural formulation for an agent's decision making process, given a state, is for the agent to decide a mapping from states to victims, or from states to target squares, where the target square may be a search target rather than a victim. Indeed, this is how the `smart` policy operates. A limitation of an action based policy is that many different sequences of actions may target the same square and, even given a substantial observation history, it is not clear that our FSM formulation is able to encapsulate this notion of higher level actions. Approaches to higher level actions include action hierarchies (Fischer et al., 2004), relational approaches (Otterlo & Kersting, 2004) and topological mappings (Smith, 2002). In our work, the necessity to find an abstraction which can be learned online, without domain knowledge, indicates that clustering techniques such as statistical clustering or topological maps may form an appropriate starting point for further work.

8.3.3 Theoretical properties

One of our motivations for using a principled approach was that it is possible to specify precisely the properties of algorithms which have been built on a well-understood theoretical framework. We have not discussed the theoretical properties of our best response algorithm in this work, but believe that an important direction for future study would consider best response in the context of the properties introduced in section 2.3 such as: convergence, rationality, regret and dynamism.

By investigating these properties, we can begin to better identify the scenarios in which our algorithm is most appropriate. For example, we have shown some empirical data concerning the way our algorithm responds to change in the environment; by investigating its convergence properties we might be able to better identify how much dynamism we expect to be able to handle effectively. Another example might be to quantify how well we expect our algorithms to be able to handle stupid team members (for example, other agents who always take the same action) or malicious agents, using regret properties.

As well as exploring the behaviour of our algorithm in particular scenarios, theoretical investigations would enable us to quantify the effects of our various approximations, subject to particular experimental conditions. This would allow an experimenter to make informed decisions about the tradeoffs between optimality and efficiency when deciding on experimental parameters such as observation history length or number of clusters.

8.3.4 Incorporating graphical model techniques

In section 3.3 we introduced graphical models as a useful visualisation for partially-observable problems. However, as well as a visualisation tool, framing our problem as inference in a Bayesian network allows us to harness general techniques for efficient belief update, based on message passing between the nodes of the tree (Mackay, 2003). By making use of exact techniques such as the junction tree[1], or (for the continuous nodes) approximate techniques such as loopy belief propagation (Murphy, Weiss, & Jordan, 1999), it may be possible to make the belief update step much more efficient while still approximately correct, allowing us to scale up to more agents with little cost.

[1] `http://www.cs.ubc.ca/~murphyk/Bayes/jtree.html`

Furthermore, the Bayesian network expression permits considerable freedom in the description of variable dependencies. For example, we can separate out all the state variables into individual nodes, using arrows to connect variables which are not independent. Observability can then be considered at this finer-grained level of individual variables. Consider again the full grand MDP model of section 3.2, in which none of the parameters are fully observable: we stated that such a scenario was intractable to work with. However, if many of the variables are mostly-observed or mostly-known, it would be possible to estimate the remaining values efficiently using graphical update techniques.

A further extension of this would be to investigate the work of (Toussaint, Harmeling, & Storkey, 2006) in our context. In this model, MDPs (or POMDPs) are formulated as a dynamic Bayesian network (DBN): that is, a Bayesian network with progression over time steps $t_1, t_2, ...t_3$, similar to a hidden Markov model (HMM) (Roweis, 2003), but with actions. Just as expectation-maximization techniques such as the Baum-Welch algorithm can be applied to solve HMMs, so they can be applied to find a solution to the DBN which maximises the corresponding value function. This solution technique applies to POMDPs and consequently, we believe, should be extensible to our multi-agent POSGs which have been formulated as POMDPs. Such a technique might provide an interesting anytime algorithm or an extension to an online algorithm, carrying out expectation-maximisation sweeps during otherwise idle CPU cycles.

8.3.5 Future trends

We also believe that our model could be used for the following kinds of problem:

Competitive environments Although we have focused on cooperative problems in this thesis, our model is based on self-interested agents and thus is equally applicable to problems which have competitive or malicious agents. Future work could investigate how well the best response strategy fares in scenarios such as a rescue scenario with journalists whose goals are contrary to the rescue agents, or a terrorist attack with terrorist agents obstructing the best response agents.

Error-prone environments Throughout this thesis, we have assumed that when an agent makes an observation that observation is correct. However, we now suppose that some of the communicated observations may be subject to error, either through system noise or, in the case of malicious agents, deliberate misdirection. This adds an extra layer of uncertainty to the agent's belief state (Even-Dar, Kakade, & Mansour, 2007) and, in the case of deliberate misdirection, requires us to consider notions of trust such as (Patel, Teacy, Jennings, & Luck, 2005).

Environments with unknown dynamism Given models for noisy or error-prone environments, it may be possible to apply similar techniques to environments which are changing over time without the agent's knowledge. By assuming that the models the agent has may be noisy, as in (Even-Dar et al., 2007), the agent can allow for changes in the environment. Subsequently detecting these environmental changes and incorporating them into its model could be done using Bayesian learning techniques in which the initial environment forms the prior environmental model.

Continuous environments In section 5.1.1, we mentioned that to apply our model in continuous environments it would be necessary to discretise the continuous states or actions. However, there are existing models for carrying out belief updates in continuous spaces and in particular continuous POMDPs (Thrun, 2000) (Doshi & Gmytrasiewicz, 2005). Thus, extending our model into such spaces would be an interesting future direction.

References

Aberdeen, D., & Baxter, J. (2002). Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the nineteenth international conference on machine learning (ICML 2002)* (Vol. 2, pp. 3–10). Sydney, Australia: Morgan Kaufmann.

Abul, O., Polat, F., & Alhajj, R. (2000). Multiagent reinforcement learning using function approximation. In *Institute of electrical and electronics engineers transactions on systems, man, and cybernetics, part C (IEEE 00)* (Vol. 30, p. 485-497). Washington, DC, USA: IEEE.

Allen-Williams, M., & Jennings, N. R. (Forthcoming in 2009a). Bayesian adaptation for complex dynamic systems. In M. Wang & Z. Sun (Eds.), *Handbook of research on complex dynamic process management: Techniques for adaptability in turbulent environments.* Hershey, Pennsylvania: IGI Global.

Allen-Williams, M., & Jennings, N. R. (Forthcoming in 2009b). Bayesian learning for cooperation in multi-agent systems. In C. L. Mumford & L. C. Jain. (Eds.), *Studies in computational intelligence: collaboration, fusion and emergence.* London, England: Springer-Verlag.

Amato, C., Bernstein, D. S., & Zilberstein, S. (2006). Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS 06)* (pp. 341–343). New York, USA: Association for Computing Machinery.

Amit, A., & Markovitch, S. (2006). Learning to bid in bridge. *Journal of Machine Learning*, *63*(3), 287–327.

Atkeson, C., & Stephens, B. (2008). Random sampling of states in dynamic programming. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems 20* (pp. 33–40). Cambridge, MA: MIT Press.

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDPs via direct gradient ascent. In *Proceedings of the seventeenth international conference on machine learning (ICML 2000)* (pp. 41–48). San Francisco, California, USA: Morgan Kaufmann.

Becker, R., Lesser, V. R., & Zilberstein, S. (2005). Analyzing myopic approaches for multi-agent communication. In *International conference on intelligent agent technology (IAT 2005)* (p. 550-557). Washington, DC, USA: IEEE Computer Society Press.

Bernstein, D. S., Hansen, E. A., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In L. P. Kaelbling & A. Saffiotti (Eds.), *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI 2005)* (p. 1287-1292). Denver, Colorado, USA: Professional Book Center.

Bigham, J., Cuthbert, L., Yang, X., Lu, N., & Ryan, D. (2004). Using intelligent agents for managing resources in military communications. *Computer Networks*, *46*(5), 709–721.

Bishop, C. M. (2004). *Neural networks for pattern recognition.* Oxford, England: Oxford University Press.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the sixth conference on theoretical aspects of rationality and knowledge* (pp. 195–210). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Bowling, M. (2005). Convergence and no-regret in multiagent learning. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in neural information processing systems 17* (p. 209-216). Cambridge, MA: MIT Press.

Bowling, M., & Veloso, M. (2001). Rational and convergent learning in stochastic games. In *Proceedings of the seventeenth International Joint Conference on Artificial Intelligence (IJCAI 01)* (p. 1021-1026). Seattle, Washington, USA: Morgan Kaufmann.

Burke, J. (2003). *Moonlight in Miami: A Field Study of Human-Robot Interaction in the Context of an Urban Search and Rescue Disaster Response Training Exercise.* Unpublished doctoral dissertation, University of South Florida.

Burkov, A., & Chaib-draa, B. (2007). Multiagent learning in adaptive dynamic systems. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems (AAMAS 07)* (pp. 1–6). New York, USA: Association for Computing Machinery.

Carmel, D., & Markovitch, S. (1996). Learning models of intelligent agents. In *Proceedings of the thirteenth national conference on artificial intelligence (AAAI 96)* (Vol. 2, pp. 62–67). Menlo Park, California, USA: AAAI Press.

Cassandra, A., Littman, M., & Zhang, N. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the 13th annual conference on uncertainty in artificial intelligence* (p. 54-61). San Francisco, CA: Morgan Kaufmann.

Castro, P. S., & Precup, D. (2007). Using linear programming for Bayesian exploration in Markov decision processes. In *Proceedings of the twentieth international joint conference on artificial intelligence (IJCAI 07)* (pp. 2437–2442). California, USA: IJCAI Incorporated.

Chalkiadakis, G., & Boutilier, C. (2003). Coordination in multiagent reinforcement learning: a Bayesian approach. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems* (pp. 709–716). New York, USA: Association for Computing Machinery.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the tenth national conference on artificial intelligence (AAAI 92)* (pp. 183–188). Menlo Park, California, USA: AAAI Press.

Clark, A., & Thollard, F. (2004). PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, *5*, 473–497.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the fifteenth national/tenth conference on artificial intelligence/innovative applications of artificial intelligence* (pp. 746–752). Menlo Park, California, USA: AAAI Press.

Dearden, R., Friedman, N., & Andre, D. (1999). Model-based Bayesian exploration. In *Proceedings of the 15th annual conference on uncertainty in artificial intelligence* (p. 150-15). San Francisco, CA: Morgan Kaufmann.

Dorais, G., Bonasso, R., Kortenkamp, D., Pell, P., & Schreckenghost, D. (1998). *Adjustable autonomy for human-centered autonomous systems on Mars.* (Presented at the Mars Society Conference.)

Doshi, P., & Gmytrasiewicz, P. J. (2005). Approximating state estimation in multiagent settings using particle filters. In *Proceedings of the fourth international joint conference on autonomous agents and multiagent systems (AAMAS 05)* (pp. 320–327). New York, USA: Association for Computing Machinery.

Durfee, E. H. (1999). Practically coordinating. *AI Magazine*, *20*(1), 99–116.

Dutech, A. (2000). Solving POMDPs using selected past events. In W. Horn (Ed.), *Proceedings of the fourteenth European conference on artificial intelligence (ECAI 2000)* (p. 281-285). Amsterdam, the Netherlands: IOS Press.

Dutta, P. S., Dasmahapatra, S., Gunn, S. R., Jennings, N., & Moreau, L. (2004). Cooperative information sharing to improve distributed learning. In *Proceedings of the workshop on learning and evolution in agent-based systems, at the third international conference on autonomous agents and multiagent systems (AAMAS 04)* (pp. 18–23). New York, USA: IEEE Computer Society Press.

Dutta, P. S., Goldman, C. V., & Jennings, N. R. (2007). Communicating effectively in resource-constrained multi-agent systems. In *20th international joint conference on artificial intelligence (IJCAI 07)* (p. 1269-1274). California, USA: IJCAI Incorporated.

Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2004). Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems* (pp. 136–143). Washington, DC, USA: IEEE Computer Society Press.

Even-Dar, E., Kakade, S. M., & Mansour, Y. (2007). The value of observation for monitoring dynamic systems. In M. M. Veloso (Ed.), *20th international joint conference on artificial intelligence (IJCAI 07)* (p. 2474-2479). California, USA: IJCAI Incorporated.

Excelente-Toledo, C. B., & Jennings, N. R. (2005, August). Using reinforcement learning to coordinate better. *Computational Intelligence*, *21*(3), 217–245.

Fischer, F., Rovatsos, M., & Weiss, G. (2004). Hierarchical reinforcement learning in communication-mediated multiagent coordination. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems* (pp. 1334–1335). Washington, DC, USA: IEEE Computer Society Press.

Fitoussi, D., & Tennenholtz, M. (2000). Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, *119*(1-2), 61-101.

Fogel, D. B. (2002). *Blondie24: playing at the edge of AI.* San Francisco, California, USA: Morgan Kaufmann Publishers Inc.

Friedman, N., & Singer, Y. (1999). Efficient Bayesian parameter estimation in large discrete domains. In *Advances in neural information processing systems 12 (NIPS 1999.* Cambridge, Massachusetts, USA: MIT Press.

Fudenberg, D., & Levine, D. K. (1998). *The theory of learning in games.* Cambridge, Massachusetts, USA: MIT Press.

Giampapa, J. A., & Sycara, K. (2001, July). Conversational case-based planning for agent team coordination. In D. W. Aha & I. W. (editors) (Eds.), *Case-based reasoning research and development: Proceedings of the fourth international conference on case-based reasoning (ICCBR 2001)* (Vol. 2080, pp. 189–203). Berlin Heidelberg: Springer-Verlag.

Gilpin, A., & Sandholm, T. (2007). Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems (AAMAS 07)* (pp. 1–8). New York, USA: Association for Computing Machinery.

Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, *24*, 49-79.

Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the nineteenth national conference on artificial intelligence (AAAI 04)* (p. 709-715). Menlo Park, California, USA: AAAI Press.

Hoar, J. (1996). *Reinforcement learning applied to a real robot task.* (MSc Dissertation, Department of Artificial Intelligence, University of Edinburgh)

Hoey, J. (2001). Hierarchical unsupervised learning of facial expression categories. *International Conference on Computer Vision, Workshop on Detection and Recognition of Events in Video.*

Ishii, S., Fujita, H., Mitsutake, M., Yamazaki, T., Matsuda, J., & Matsuno, Y. (2005). A reinforcement learning scheme for a partially-observable multi-agent game. *Journal of Machine Learning*, *59*(1-2), 31–54.

Izadi, M. T., & Precup, D. (2006). Exploration in POMDP belief space and its impact on value iteration approximation. In *European conference on artificial intelligence (ECAI 06), workshop on planning, learning and monitoring with uncertainty and dynamic worlds (PLMUDW).* Amsterdam, the Netherlands: IOS Press.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1-2), 99–134.

Kim, Y., Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2006). Exploiting locality of interaction in networked distributed POMDPs. In *Proceedings of the AAAI spring symposium on "Distributed plan and schedule management".* Menlo Park, California, USA: AAAI Press.

Lee, H., Battle, A., Raina, R., & Ng, A. Y. (2007). Efficient sparse coding algorithms. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems 19* (pp. 801–808). Cambridge, MA: MIT Press.

Leslie, D. (2004). *Reinforcement learning in games.* Unpublished doctoral dissertation, University of Bristol.

Lesser, V. (1999, January). Cooperative Multiagent Systems: A Personal View of the State of the Art. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering*, *11*(1).

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th international conference on machine learning* (pp. 157–163). New Brunswick, NJ: Morgan Kaufmann.

Littman, M. L., & Stone, P. (2001). Implicit negotiation in repeated games. In *Proceedings of the eighth international workshop on agent theories, architectures, and languages (ATAL 2001)* (p. 393-404). Berlin, Germany: Springer.

Mackay, D. J. C. (2003). *Information theory, inference, and learning algorithms.* Cambridge, England: Cambridge University Press.

Marecki, J., Gupta, T., Varakantham, P., & Tambe, M. (2008). Not all agents are equal: scaling up distributed POMDPs for agent networks. In *Proceedings of the seventh international joint conference on autonomous agents and multiagent systems (AAMAS 08).* New York, USA: Association for Computing Machinery.

Matsuno, Y., Yamazaki, T., Matsuda, J., & Ishii, S. (2002). A multiagent reinforcement learning method based on the model inference of the other agents. *Systems and Computers in Japan*, *33*(12), 67-76.

Mitchell, T. M. (1997). *Machine learning.* Maidenhead, England: McGraw-Hill.

Murphy, K., Weiss, Y., & Jordan, M. (1999). Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of the conference on uncertainty in artificial intelligence (UAI 99)* (p. 467-475). San Francisco, California, USA: Morgan Kaufmann.

Naeem, U., & Bigham, J. (2008). Activity recognition using a hierarchical framework. In *Proceedings of the second international conference on pervasive computing technologies for healthcare, 2008* (p. 24-27). Washington, DC, USA: IEEE Computer Society.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D. V., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In G. Gottlob & T. Walsh (Eds.), *"18th international joint conference on artificial intelligence"* (p. 705-711). San Francisco, California, USA: Morgan Kaufmann.

National Research Council. (2005). *Summary of a workshop on using information technology to enhance disaster management.* Washington, DC, USA: National Academies Press.

Nikovski, D., & Nourbakhsh, I. (1999). Learning discrete Bayesian models for autonomous agent navigation. In *Proceedings of the first IEEE symposium on computational intelligence in robotics and automation* (pp. 137–143). Washington, DC, USA: IEEE Computer Society.

Oliehoek, F. A., & Vlassis, N. (2007). Q-value functions for decentralized POMDPs. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems (AAMAS 07)* (pp. 1–8). New York, USA: Association for Computing Machinery.

Otterlo, M. van, & Kersting, K. (2004, December). Challenges for relational reinforcement learning. In P. Tadepalli, R. Givan, & K. Driessens (Eds.), *Proceedings of the workshop on relational reinforcement learning at the international conference on machine learning (ICML 2004)* (pp. 74–80). New York, USA: Association for Computing Machinery.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, *11*(3), 387–434.

Paquet, S., Tobin, L., & Chaib-draa, B. (2005). An online POMDP algorithm for complex multiagent environments. In *Proceedings of the fourth international joint conference on autonomous agents and multiagent systems* (pp. 970–977). New York, USA: Association for Computing Machinery.

Patel, J., Teacy, W. T. L., Jennings, N. R., & Luck, M. (2005). A probabilistic trust model for handling inaccurate reputation sources. In *Proceedings of the 3rd international conference on trust management* (Vol. 3477, p. 193-209). Rocquencourt, France: Springer-Verlag.

Powers, R., & Shoham, Y. (2005). New criteria and a new algorithm for learning in multi-agent systems. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in neural information processing systems 17* (p. 1089-1096). Cambridge, MA: MIT Press.

Powers, R., Shoham, Y., & Vu, T. (2007). A general criterion and an algorithmic framework for learning in multi-agent systems. *Journal of Machine Learning*, *67*(1–2), 45–76.

Ramamritham, K., Stankovic, J. A., & Zhao, W. (1989). Distributed scheduling of tasks with deadlines and resource requirements. *Institute of Electrical and Electronics Engineers Transactions on Computers*, *38*(8), 1110–1123.

Rasmussen, C. E., & Williams, C. K. (2006). *Gaussian processes for machine learning.* Cambridge, MA, USA: MIT Press.

Reisinger, J., Stone, P., & Miikkulainen, R. (2008, July). Online kernel selection for Bayesian reinforcement learning. In *Proceedings of the twenty-fifth international conference on machine learning (ICML 2008).* New York, USA: Association for Computing Machinery.

Ren, Z., & Williams, A. B. (2003). Lessons learned in single-agent and multiagent learning with robot foraging. In *Institute of electrical and electronics engineers transactions on systems, man, and cybernetics (IEEE 03)* (Vol. 3, pp. 2757–2762). Washington, DC, USA: IEEE Computer Society.

Rivest, F., Bengio, Y., & Kalaska, J. (2005). Brain inspired reinforcement learning. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in neural information processing systems 17* (p. 1129-1136). Cambridge, MA: MIT Press.

Rogers, A., David, E., Schiff, J., & Jennings, N. R. (2007). The effects of proxy bidding and minimum bid increments within eBay auctions. *ACM Transactions on the Web.*, *1*(2), 9.

Ross, S., Chaib-draa, B., & Pineau, J. (2008). Bayes-adaptive POMDPs. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in* *neural information processing systems 20 (NIPS 2007)* (pp. 1225–1232). Cambridge, Massachusetts, USA: MIT Press.

Ross, S., Pineau, J., Paquet, S., & Chaib-draa, B. (2008). Online planning algorithms for POMDPs. *Artificial Intelligence Research*, *32*, 663-704.

Roweis, S. (2003). *Hidden Markov models.* (SCIA Tutorial)

Roy, N., & Gordon, G. (2002, December). Exponential family PCA for belief compression in POMDPs. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing* (p. 1043-1049). Vancouver, British Columbia, Canada: MIT Press.

Sadikov, A., & Bratko, I. (2006). Learning long-term chess strategies from databases. *Journal of Machine Learning*, *63*(3), 329–340.

Sallans, B. A. (1999). Learning factored representations for partially observable Markov decision processes. In *Advances in neural information processing systems 12 (NIPS 1999* (p. 1050-1056). Cambridge, Massachusetts, USA: MIT Press.

Sallans, B. A. (2002). *Reinforcement learning for factored Markov decision processes.* Unpublished doctoral dissertation, University of Toronto. (Adviser-G. E. Hinton)

Scerri, P., Sycara, K., & Tambe, M. (2004). Adjustable autonomy in the context of coordination. In *American institute of aeronautics and astronautics 3rd "unmanned unlimited" technical conference, workshop and exhibit (AAAI 04).* Chicago, Illinois, USA: American Institute of Aeronautics and Astronautics. (Invited Paper)

Schurr, N., Marecki, J., Lewis, J. P., Tambe, M., & Scerri, P. (2005). The DEFACTO system: Coordinating human-agent teams for the future of disaster response. In R. Bordini, M. Dastani, J. Dix, & A. El Fallah Seghrouchni (Eds.), *Multi-agent programming:languages, platforms and applications. multiagent systems, artificial societies, and simulated organizations* (Vol. 15, p. 197-215). Heidelberg, Germany: Springer.

Shani, G., Brafman, R. I., & Shimony, S. E. (2005). Model-based online learning of POMDPs. In *Proceedings of the European conference on machine learning (ECML 2005)* (p. 353-364). Oporto, Portugal: Springer.

Shi, J., & Littman, M. L. (2002). Abstraction methods for game theoretic poker. In *Revised papers from the second international conference on computers and games (CG 00)* (pp. 333–345). London, England: Springer-Verlag.

Sims, M., Corkill, D., & Lesser, V. (2004, November). *Separating Application-Specific and Organizational Coordination Issues during Multi-Agent Organizational Design and Instantiation* (Computer Science Technical Report No. 04-98). Cambridge, Massachusetts, USA: University of Massachusetts.

Smith, A. J. (2002). *Dynamic generalisation of continuous action spaces in reinforcement learning: A neurally inspired approach.* (Ph.D. thesis, Division of Informatics, Edinburgh University, UK.)

Stenning, K., & Lambalgen, M. van. (2005). Semantic interpretation as computation in nonmonotonic logic: the real meaning of the suppression task. *Cognitive Science*, *29*(6).

Sun, R., & Naveh, I. (2004). Simulating organizational decision-making using a cognitively realistic agent model. *Journal of Artificial Societies and Social Simulation*, *7*(3).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, Massachusetts, USA: MIT Press.

Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the twenty-first annual conference on uncertainty in artificial intelligence (UAI 05)* (p. 576). Arlington, Virginia: AUAI Press.

Takeuchi, I., Kakumoto, S., & Goto, Y. (2003). Towards an integrated earthquake disaster simulation system. *First International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster.*

Tambe, M., Bowring, E., Pearce, J. P., Varakantham, P., Scerri, P., & Pynadath, D. V. (2006). Electric elves: What went wrong and why. In *Proceedings of the AAAI spring symposium on what went wrong and why: Lessons from AI* *research and applications.* Menlo Park, California, USA: AAAI Press.

Tanner, B., Bulitko, V., Koop, A., & Paduraru, C. (2007). Grounding abstractions in predictive state representations. In *Proceedings of the twentieth international joint conference on artificial intelligence (IJCAI 07)* (pp. 1077–1082). California, USA: IJCAI Incorporated.

Tesauro, G. (2004). Extending Q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems 16 (NIPS 2003).* Cambridge, Massachusetts, USA: MIT Press.

Thrun, S. (2000). Monte Carlo POMDPs. In S. Solla, T. Leen, & K.-R. Müller (Eds.), *Advances in neural information processing systems 12 (NIPS 00)* (pp. 1064–1070). Cambridge, Massachusetts, USA: MIT Press.

Toni, F., & Bentahar, J. (2008). Computational logic-based agents. *Autonomous Agents and Multi-Agent Systems*, *16*(3), 211–213.

Toussaint, M., Harmeling, S., & Storkey, A. (2006, Dec). *Probabilistic inference for solving (PO)MDPs* (Tech. Rep.). Edinburgh: University of Edinburgh.

Varakantham, P., Marecki, J., Yabu, Y., Tambe, M., & Yokoo, M. (2007). Letting loose a SPIDER on a network of POMDPs: generating quality guaranteed policies. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems (AAMAS 07)* (pp. 1–8). New York, USA: Association for Computing Machinery.

Virin, Y., Shani, G., Shimony, S. E., & Brafman, R. I. (2007). Scaling up: Solving POMDPs through value based clustering. In *Proceedings of the twenty-second AAAI conference on artificial intelligence* (p. 1290-1295). Vancouver, British Columbia, Canada: AAAI Press.

Vu, T., Powers, R., & Shoham, Y. (2006). Learning against multiple opponents. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS 06)* (pp. 752–759). New York, USA: Association for Computing Machinery.

Wooldridge, M. (2002). *An introduction to multi-agent systems.* Chichester: Wiley.

Yuichi, Y., Makoto, Y., & Atsushi, I. (2007). Multiagent planning with trembling-hand perfect equilibrium in multiagent POMDPs. In *Proceedings of the* *tenth Pacific Rim international workshop on multi-agents (PRIMA 2007)* (Vol. 11). Bangkok, Thailand: Springer-Verlag LNCS.