Bayesian learning for cooperation in multi-agent systems

Mair Allen-Williams and Nicholas R Jennings

Abstract 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 many agents and millions of states. We improve on these approaches by extending a principled Bayesian model into more challenging domains, using heuristics and exploiting domain knowledge in order to make approximate solutions tractable. We show the effectiveness of our approach applied to an ambulance coordination problem inspired by the Robocup Rescue system.

1 Introduction

As computing power and ubiquity increase, the use of multi-agent technology in large distributed systems is becoming more widespread. For example, sensors are often now included in new buildings or vehicles. When these sensors are able to sense intelligently and communicate with one another, they form a multi-agent system Beyond sensing, mobile sensors may be able to make inferences about a scenario such as a terrorist attack or a flood, and provide human teams with uncertainty estimates and suggest actions. In situations where a human would be at some risk, intelligent communicating machines may be deployed—for example, thousands of UAVs (unmanned aerial vehicles) can collaborate to search over a wide area [32], or robots may also act as intelligent agents, aiding or replacing humans to perform a co-

Mair Allen-Williams, Nicholas R Jennings
School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK.
{ mhaw05r, nrj } @ecs.soton.ac.uk

ordinated search of a burning building [33]. Consequently, as such systems develop, the scalability of complex interacting systems becomes increasingly important.

In more detail, an autonomous agent is an entity which makes local perceptions within an environment and processes these perceptions in order to decide how to act on that environment, based on some internal goals. When many such agents are acting within the same environment then the actions of one can affect the perceptions of others. This is then the essence of a multi-agent system and the reason why cooperation is essential to its effective operation.

In order to provide a focus, and to motivate this work, we will consider the disaster response domain as an example multi-agent system. Disaster scenarios form rich grounds for multi-agent distributed problem solving, allowing us to explore several features of complex multi-agent problems. While there are many characteristics which may be present in disaster scenarios, we will find that there are two common themes: *uncertainty*, and *coordination*.

The first of these, uncertainty, may concern the environment ("What's going on?") and the agent's position in the environment ("Where am I?"); it may be about any other agents which might exist in the environment ("Who else is around? Where are they?") and their behaviour ("What are they going to do?"). In these uncertain situations, each agent must do some form of discovery to determine the essential characteristics of the scenario, including the agent's collaborators, before and alongside directly working to achieve its goals. This discovery phase in a multi-agent system is tightly linked with the presence of other agents in the system. As well as determining which other agents are present, agents may be able cooperate to search over different regions, sharing information with each other as appropriate.

In addition to explicitly sharing information, observing the behaviour of the other agents allows an autonomous agent to make inferences about the system. For example, in a scenario involving a burning building, a rational agent will not enter the building (although a specially designed robot or one which believes itself to be expendable may). Beyond discovery, there will continue to be interaction between the agents in a multi-agent system, whether explicit via communications and negotiations, or implicit through activity. Moreover, achieving some subgoals may involve a collaboration between several agents, as in a rescue operation where two ambulance members are required to carry a stretcher, or a driving team with a navigator and a driver.

Now, this general problem of taking others into account, *coordination*, is the second key issue we have identified for multi-agent systems. In uncertain or open systems, fixed protocols for coordination must function against a background where agents are not fully aware of the situation; that is, their environment, the resources available to them, or the behaviour of the other agents. The negotiation of coordinated behaviour in such systems is intertwined with the discovery phase, as agents interact with one another, perhaps cooperating to determine properties of the situation.

In order to achieve such a comprehensive model for planning and acting, we have built on existing techniques for cooperative decision making under uncertainty. Now, the inherent dynamism in many of these problems calls for timely online responses, rather than offline computation of strategies. For partially observable multi-agent problems, recent work has advanced the state of the art for finding offline solutions in systems of stationary agents, with solutions in systems containing at least fifteen agents (where previously the maximum was five or six) [23]. Building on this, we describe a related *online* approach which is suitable for complex dynamic systems with mobile agents, and also scales to tens of agents.

Our algorithm explicitly models other agents, demonstrating a principled approach to cooperative behaviour in uncertain and partially-observable multi-agent systems. Through empirical evaluation, we show that our approach, using learned finite state machines to approximate the behaviour of others, is more efficient than other principled approaches [14] and more effective than a state-of-the-art hand-written strategy for the same scenario. In practical terms, this algorithm could be directly deployed in systems such as collaborative mobile sensor networks or for UAVs sharing a distributed search. In more human-led systems, such as a disaster response, we anticipate that communicating sensors in both fixed networks (for example, in buildings) and hand-held applications could suggest actions or action sequences to human participants. While in the future we expect that robot teams will be able to become increasingly autonomous with valuable contributions to such scenarios, in this Chapter we think it best to consider our ‘agents’ to be computer-advised humans.

Over the next sections, we provide a detailed background to our work (Section 2) and motivate a cooperative approach using finite state machines to model agent behaviour. In Sections 3 and 4 we describe this approach in detail. In order to validate our model we test it on a problem taken from the disaster response domain. Section 5 describes this problem and Section 6 compares the performance of our algorithm with previous approaches to partially observable uncertain systems. In Section 7 we conclude and describe directions for future work.

2 Background

In this Section we introduce the ideas which we will use in our algorithm, 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. First, however, we introduce the disaster response domain as a motivation for this work and identify its key characteristics.

2.1 The disaster response domain

We ground our work in the disaster response domain. After a disaster such as an earthquake or a flood, the immediate situation and its environmental properties are typically unknown to the rescue teams, and the complete situation cannot be atomically observed by a single agent. Rescue teams may come from different regions but all must collaborate to search the area and rescue any disaster victims in a timely fashion. However, communication lines may be unavailable or restricted so that this collaboration is necessarily implicit or based on short one-way communications.

In more detail, we find that taking disaster response as our focus domain drives a particular interest in collaborative multi-agent domains which include the following properties:

Decentralisation: In these large and dynamic systems, providing a central controller is likely to be infeasible. Firstly, there are unlikely to be sufficient resources to allow communications between one central controller and every other node. Secondly, one central controller is almost certainly not going to be able to obtain a complete view of the system, and the potentially rapid changes as agents enter and leave the system would be difficult to track.

Dynamism: Realistic systems are rarely static. For example, in disaster recovery agents must adapt to changing weather conditions, any aftershocks, and unexpected events such as building collapse or fires. When taken together, this can lead to a very dynamic environment.

Partial observability: Along with decentralisation, it is likely that no one agent is able to see the complete system all the time. Although communication between agents may extend a particular agent's view of the system, the agent must continually make judgements based on an incomplete view.

Bandwidth-limited: Limited communication is a characteristic common to disaster scenarios—for example, mobile phone networks often become jammed [24], or time constraints can limit opportunities for communication. Thus, agents may be able to exchange some information, but both time and bandwidth restrictions will limit these exchanges.

Openness: The rescue agents are likely to be entering or leaving the disaster scene throughout the rescue operation. Agents may be harmed at the scene and thenceforth be out of action, while new agents may arrive late. A collaborative model in a disaster response scenario must therefore be able to adapt to the continual arrival and loss of agents.

Example 1 describes an earthquake scenario having many of these features. Keeping these driving forces in mind, we begin with a description of the most straightforward of this class of dynamic problems, the single agent observable Markov Decision Process (MDP) (Section 2.2.1). Building on the single agent MDP, we will generalise to partially observable (Section 2.2.4) and multi-agent environments (Section 2.3), discussing means of coordination among agents. We will not discuss open systems in detail until the final Section (Section 7).

Example 1 Earthquake scenario

An earthquake has damaged the small town of Tameugny (Figure 1), including its hospital and ambulance fleet. Buildings are still collapsing and there may be aftershocks (*dynamism*). Ambulance teams from nearby towns converge on Tameugny, 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 Tameugny, 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*.

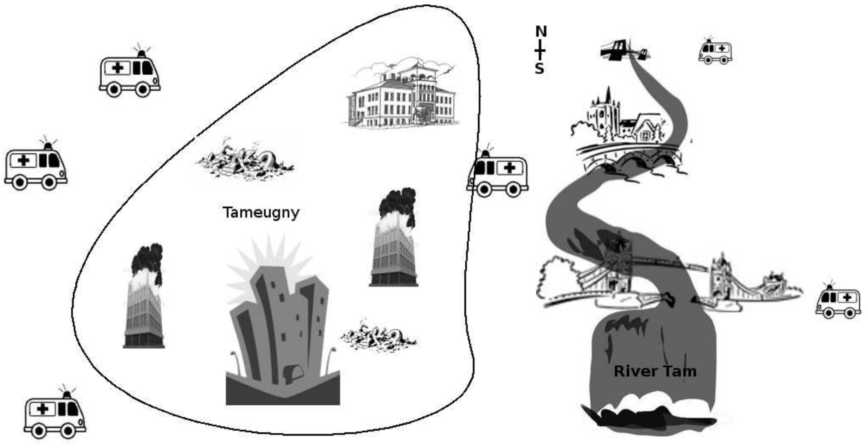


Fig. 1 Tameugny, after an earthquake

2.2 Single agent decision making

We begin with an introduction to decision making processes for a single agent acting in a dynamic environment, going on to explain how the agent can learn about uncertain environments via reinforcement learning, and introducing Bayesian reinforcement learning. We then extend these decision making techniques into partially observable environments, again using Bayesian techniques.

2.2.1 Markov Decision Processes

In this most basic model, the agent perceives the state of the world through its sensory inputs, and decides on its immediate action 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 of Markov decision theory [36]. The fundamental feature of this theoretical model is the assumption that the immediate next state is dependent only on the previous state and choice of action—this is the *Markov property*. Although the Markov property may not fully hold, it is often a sufficiently good approximation, and techniques which use this theory can get good results. This is demonstrated by many practical examples [18] [35] [2]. With the Markov assumption, if the models describing the transition and reward probabilities are completely known to the agent then the system can be solved, using a pair of recursive equations [36] which determine the optimal action from each world state. These are the *Bellman equations*. For large systems, there are efficient ways of approximating these solutions—we do not go into these here, as we will not be dealing with known MDPs, but refer the interested reader to [36], Chapter 9.

2.2.2 Reinforcement learning

Example 2 Rescue worker in the old people's home

A lone rescue worker searches Tameugny'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 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.

When there is uncertainty about the aforementioned models, as in Example 2, the agent can learn the optimal actions through experimentation. To this end, *reinforcement learning* techniques, such as Q-learning, TD(λ) and SARSA [36], provide techniques for the agent to do this. There are two types of learning: model-based or model-free. In the former, the agent aims to learn the system model, in this case the underlying MDP, and then solve that model (using the Bellman equations, as above) to decide an action. 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. However, in 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. Models or parts of models can also be re-used in different problems. Given this, we will 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.

2.2.3 Bayesian reinforcement learning

In particular, we focus on Bayesian model-based learning. By comparison to most model-based learning methods, which maintain a point estimate of the models, a Bayesian learning method will maintain a probability distribution over all possible models, in the form of a *belief state*. This provides a principled solution to the *exploration-exploitation* problem (Example 3): the decision an agent has to make between taking the action it currently believes to be optimal, and taking an exploratory action. In general, the more certain the agent is about its current model, the more likely it should be to take the currently optimal action. The Bayesian model pins this intuition down precisely.

Example 3 Exploration-exploitation in the Tameugny earthquake

The river Tam runs to the East of Tameugny. 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 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.

2.2.4 Partially observable Markov Decision Processes

Now, reinforcement learning, combined with the Bellman equations, will allow a single agent to solve any observable MDP which comes its way. However, 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 4), 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, and there are a host of POMDP-solution techniques.

Example 4 Partial observability in the Tameugny earthquake

Two Tameugny 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.

For example, when the underlying environmental model is known, the POMDP can be converted to a continuous Markov decision process by defining a *belief state* as a probability distribution over states. The resulting continuous MDP, from belief state to belief state, can be solved using exact algorithms [7] or using approximations to make computation easier [3], [20]. If the underlying model is not known, learning techniques must be used to refine a solution as the agent explores the system. Model-free approaches, such as [1], have had some success in using learning techniques to solve POMDPs. However, as discussed, we believe that model-based approaches may again have benefits—for example, [34] demonstrates a model-based algorithm which uses variable length suffix trees to address the fact that even if state transitions are Markov, the observable process may not be. However, existing approaches rely on a number of approximations and assumptions about the state space, hence are not entirely satisfactory. A principled approach may be to extend the Bayesian model described previously into partially observable domains [30].

2.3 Coordinated decision making

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 (Example 5). This interaction can be modelled by defining a (hyper)sphere of influence for each agent within the environment. Overlapping spheres of influence indicate interactions between agents [41]. 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. Making decisions in the context of these other agents is the fundamental principle of coordination [12]. Clearly, this is a central part of a reasoning agent in a multi-agent system. Thus, in the following sections we expand on how agents can reason about the behaviour of others and incorporate that reasoning into their own behaviour.

Example 5 Ambulance traffic at the the Tameugny earthquake

Consider again the ambulance driver arriving at the River Tam after the Tameugny 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 one bridge, a traffic jam would form there, perhaps wasting precious time.

Perhaps the simplest example: agents functioning in uncertain worlds among other agents may include others' behaviour in the Markov state transition model they develop. However, by doing this they may form inaccurate assumptions about the world, as agents adapt their behaviour to one another. Consequently, maintaining models of the world and of other agents separately provides greater flexibility and may enable the agent to reuse a world model as agents come and go, or reuse models built for known agents in fresh scenarios. Below, we outline three common ways in which agents may develop and use models of the world to coordinate.

2.3.1 Coordination mechanisms

Three, potentially overlapping, coordination mechanisms are identified by [4]: conventions, communication, and learning (Example 6). Firstly, **conventions** are typically the simplest form of coordination. In a convention-based coordination 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 [16]. However, it is inflexible, and relies on all participants knowing the conventions and complying with them. More complex models using conventions include role-based structures [37] and self-organising structures [39]. 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 [13]. 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. Learning techniques enable agents to evolve coordinated polices within uncertain state spaces, either with a group of learners exploring the space and converging towards an equilibrium (as in [10] and [22]), or by one agent explicitly learning about the behaviour of others in order to adapt its own appropriately [8].

Example 6 Coordination mechanism examples

Consider the ambulance drivers approaching the Tameugny earthquake. They can use all three of the above coordination mechanisms:

- By **convention**, they will drive on the left hand side of the road; they will use sirens to indicate their approach; they will rescue the elderly, the mothers and the children first.
- They will **communicate** with the other ambulance drivers, calling things like "road's gone up there", "I think someone needs to check out the North-East of the town", "We need three more people to help lift here". By convention, sirens also communicate with others.
- As they work with other ambulance teams, they will **learn** which teams need the most help, which areas of the city have been searched, and how a particular team tends to operate (for example, whether they use *one-two-**lift***, or *one-two-three-**lift***)

2.3.2 Game theory

Distinct from the three approaches to coordination identified above, another research domain which investigates coordination from a theoretical angle is **game theory** [21]. In game-theoretic formulations, agents model the scenario as a game and try and derive, either through exact evaluation or through learning, a *best response* to the strategies of the other players in the game (Example 7). If all the players iteratively keep playing best responses, and if strategies are *mixed* (stochastic) the play will converge to a (mixed) equilibrium, in which every player's strategy is a best response to every other player. One of the challenges of game theory is to direct the play so that convergence is not just to any equilibrium but to an optimal one [10].

Within the domain of game theory, the form of multi-agent learning in which the agents maintain explicit models of the other agents is described as learning in stochastic games. One effective approach to extending single-agent reinforcement learning into this 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 [5]. However, WoLF techniques can be improved upon by using a Bayesian model in which agents maintain beliefs about the behaviour of the other agents, as well as a probability distribution over world models [8]. The need for heuristically determined learning rates is then eliminated, while prior information about agents can be incorporated.

Game theory is an obvious model for scenarios with heterogeneous and competitive agents, but searching for the optimal Nash equilibrium is also a useful formulation for cooperative problems. WoLF and related approaches are often applied in such problems, with each agent gradually adjusting to the others so that the whole system is incrementally improved. Although there is no guarantee that the optimal equilibrium is found, the technique is effective, and in large problems it is often sufficient to come up with a "good" solution rather than the optimal solution.

2.3.3 Integrated learning and coordination

Considering these coordination techniques 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 smoothly make decisions about coordinated actions. Moreover, we believe that such an integrated approach should be based on sound theoretical principles, allowing us to reason about the behaviour of agents. In uncertain and dynamic domains, this motivates the use of multi-agent learning models, since these provide a basis for such coordinated action selection and are designed for uncertain domains. Furthermore, although in large domains it may be impractical to learn a complete solution in real time, we have explained that learning methods can be used on top of other coordination mechanisms to pro-

Example 7 Game theory in Tameugny

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 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 $Lossofstatus$ $S = -20$

vide adaptability on top of known conventions or communication languages, to select between coordination mechanisms, or to use learning for some subproblem. We therefore explore the application of multi-agent learning models to dynamic, partially-observable domains. Finally, we observe that within model based learning it is sensible for agents to maintain models of the other agents separately from the environment, as these models need not be treated as Markovian. Therefore, the game theoretic paradigm, computing "best responses" to agents within their environment, is appropriate and more flexible than treating other agents implicitly.

2.3.4 Learning models of other agents' behaviour

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 7, 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* [17]. 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. In a rescue scenario, some rescue tasks require several agents, and so 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. In Example 7, 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.

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—for example, projecting just a small number of steps into the future, and using a domain-specific heuristic to estimate the values of those future states [14]. Even this heuristic approach relies on each agent being able to predict the computations of the other agents—each must be initialised with the same random seed. A different approach to approximation is to restrict the possible opponent strategies to those which can be described by regular automata, often called *finite state machines* or *finite automata*.

2.3.5 Finite state machines

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 [38] [6] demonstrate that it can be very effective. Furthermore, approximate best responses to finite state machines can be computed efficiently [23].

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: 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, there may be no problem in generating redundant information. Other approaches use 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 discretized state space [31]. 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.

In the next Section we expand some of these ideas and describe in detail an algorithm for online cooperative action in partially observable multi-agent systems in which agent communication is limited to information-sharing. 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 Bayesian learning models

As outlined in the previous Section, we will use finite state machines to model individual agent policies in a multi-agent setting. In this Section we flesh out the theoretical background behind this model, first outlining an exact theory and then showing how this is approximated by a finite state machine model. Then in Section 4 we describe the finite state machine model in more detail.

3.1 Bayesian Learning

We begin by specifying our 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:

- $S : \{s_0, \dots s_{n_s}\}$, a set of states. A state will generally be described by a set of state variables.
- $I : \{I_1, \dots I_k\}$, a set of k agents
- $L \subset S = \{L_1, \dots L_k\}$, a location variable for each agent. These determine the viewpoint from which agents make local observations.
- $A = \{a_1, \dots a_{n_a}\}$, 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 emphasize that it is a vector. We may 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.
- $O : \{o_o, \dots o_{n_o}\}$, a set of observations
- $T_f : T_f(s_{t+1}, s, \mathbf{a}_t) = P(s_{t+1}|s_t, \mathbf{a}_t)$, the transition function from state to state, where $s_{t+1}, s_t \in S$ and $\mathbf{a} \in \mathbf{A}$.
- 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 $s_t \in S$.
- $R : \{r_1, \dots .r_{n_r}\}$, $n_r \leq n_s$, a set of possible rewards which an agent may receive

- $R_f : SxAxS \rightarrow R$, a reward function, specific to the agent. 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 may 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 $\pi : (p, h, o_t) \rightarrow a$ where h defines all relevant historical information (observation sequences including communications from other agents), 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, we go on in the rest of this Section to build up a formal model of learning in multi-agent systems. The following Section (Section 4) explains an approximation which can be used to make implementing this model practical in a particular special case of interest to us. First, however, we introduce *Markov Decision Processes*.

3.2 MDPs and POMDPs

The transition function T_f has the *Markov property*: the probability of future states depends only on the current state and the action choices, and not on past state history. Consequently, $\{S, A, T, R\}$ defines a *Markov Decision Process* (Figure 2):

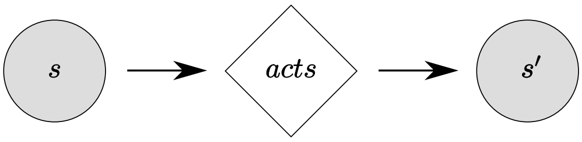


Fig. 2 Markov Decision Process progression

In choosing an action at time $t = t_T$, the agent's aim is to optimise the expected discounted future rewards, defined by:

$$R_T^\gamma = \sum_t \gamma^t r_t \quad (t \text{ ranges from } T \text{ to } \infty) \tag{1}$$

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 well). 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. 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.

Now, in a fully observable world, $O = S$ and $P(o_t|s_t) = (1 \text{ if } o_t = s_t, 0 \text{ otherwise})$, i.e. the agent knows the complete state s_t at every timestep t. Given the *Markov property*, its optimal policy therefore need depend only on the current state. We can therefore define a policy in a fully observable MDP by $\pi(s) = a$, a function from states to actions. Then, if the strategies of the other agents are known, the agent can compute its own optimal policy by solving the large simultaneous equation known as the Bellman Equations (2 and 3), via *dynamic programming*, and then taking the policy π^* described in equation 4.

In more detail, $Q_\pi(s,a)$ is the (discounted expected) value of taking action a from state s, and then working to policy π. $Q^*(s,a)$ is the (discounted expected) value of taking action a from state s, and then working to the optimal policy π^*. We will use "best response" to refer to the optimal single-agent action, a maximising $Q(s,a)$ throughout this paper as we replace s with more complex models.

$$\begin{aligned}
Q(s,a) &= \sum_{s'} P(s'|s,a)[r(s') + \gamma V(s')] && (2)\\
V(s) &= \max_a Q(s,a) && (3)\\
\pi^*(s) &= a \text{ such that } Q(s,a) = \max_a Q(s,a) && (4)\\
P(s'|s,a) &= T_f(s', s, a \circ \mathbf{a_{-i}}) && (5)
\end{aligned}$$

where $\mathbf{a_{-i}}$ is the joint actions of the other agents as defined by their strategies

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 [36]. We do not go into details of these solution techniques as realistically we are unlikely to know all the necessary parameters. In the next Section we explain how this model is extended into systems with unknowns.

3.2.1 Partially observable systems

It is often the case that the agent may not know (in the case of static parameters), or be able to observe (in the case of state-related values) all the details of the MDP. If the underlying state s cannot be observed, then the problem becomes a *POMDP*: a "partially observable" Markov decision process (Figure 3). At each step, the MDP proceeds behind the scenes, while the agent makes observations o derived from the underlying state s, where o is insufficient for the agent to reliably determine s. $O_f(s,i)$ describes the probability density function $P(o|s)$ for agent i.

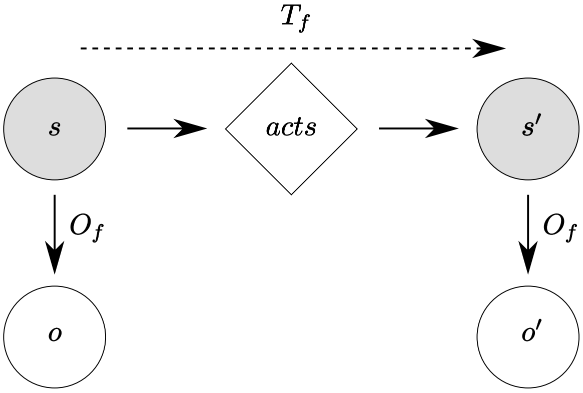


Fig. 3 Partially observable Markov Decision Process

To solve this POMDP, we can derive from it a secondary MDP—a *belief MDP*. 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 probability 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 6) to update the state probabilities (Figure 4):

$$P(x|observations) \propto P(observations|x)P(x) \tag{6}$$

This belief MDP is, therefore, completely known, and although continuous and high-dimensional has an exact solution describing the optimal action in any belief state. This solution will inherently take into account the need for exploratory actions. In principle, any general techniques for continuous MDP solutions can be used to solve the belief MDP [36]. However, all belief-state MDPs fall into a particular class of continuous MDPs, since each belief state restricts the possible future belief states. More efficient solution techniques exploit the properties of these MDPs [28] [27].

Given this, we can extend the belief MDP idea further to consider cases where the environmental dynamics, θ, are not known or are partially known. In these cases, we can consider an underlying MDP which has the dynamics, θ, as one of its state variables. This MDP has a known transition function: $(s,\theta) \rightarrow (\theta(s),\theta)$. The obser-

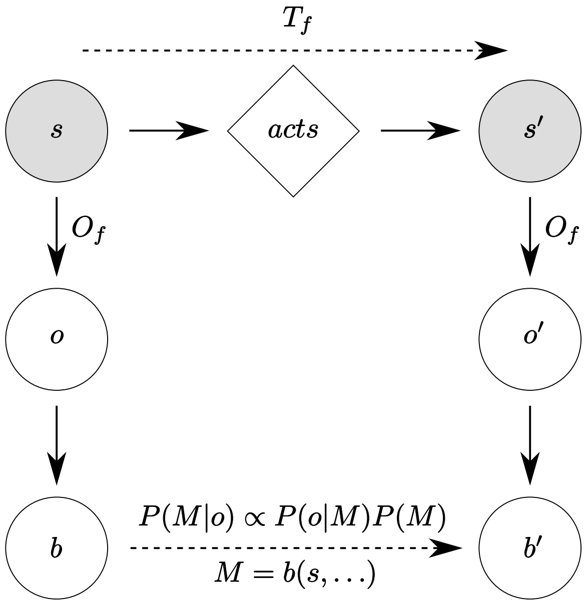


Fig. 4 POMDP inducing a Bayesian belief state MDP

vations for the POMDP associated with this MDP will include state transitions as well as the immediate observations. In principle, this POMDP can be solved exactly as described above. Finally, the same model extends into the multi-agent world by including the actions of other agents in the underlying state, and the behaviour functions of other agents in θ. In a partially observable system, the behaviour of another agent will depend on its beliefs about the state, and so we also add the beliefs over states of the other agents to our own MDP state:

$$s_{MDP} = \{s, \forall j.(\sigma_j), \forall j.(b_j(s)), \theta\}$$

To date, existing work has studied some sub-cases of this general model: the fully-observable case where the dynamics are unknown, for single agent problems [11] and multi-agent problems [8]; and the partially-observable case where the dynamics are unknown for multi-agent problems [30]. All of these find online solutions using appropriate approximation techniques. In particular, in solving the Bellman equations, typically these techniques will only refer to a small number of belief states, beginning at the current one. Recall,

$$Q(s,a) = \sum_{s'} P(s'|s,a)[r(s') + \gamma V(s')]$$

The belief-state version, writing b for the belief state and leaving s to refer to the underlying state, is

$$Q(b,a) = \sum_{s'} P(s'|b,a)[E[r(s')|b] + \gamma V(b')]$$

where b' is the belief state resulting from the transition to state s'.

Finally, the case of particular interest to us, the partially-observable multi-agent case with known dynamics (sometimes described as a partially observable stochastic game, or POSG) has also been investigated. For example, in one online approximate algorithm [14], each agent tries to compute the joint optimal action for that step, then executing its own part of this joint optimal action. Providing that all agents are initialised with the same information (in particular, they should share a random seed), every agent can compute the 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. 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 [23].

An alternative technique for making approximate action choices is to gather together similar states, belief states or groups of observations, reducing the state space. In particular, in problems where a notion of proximity can be defined between states, an action can be decided for a new state based on experience of nearby states. Examples of the former include manual feature abstraction and hierarchies [15]. Examples of the latter include include neural networks [36], Kohonen maps [35] and belief compression via principal components analysis [31]. We leave investigation of such state aggregration to future work.

A further optimisation that has been used to good effect for modelling and responding to agents in partially observable domains is to restrict agent policies to the class of policies which can be described by *finite state machines* (sometimes called *finite state automata* or *regular automata*, and abbreviated to FSMs). Using this class of policies, which we describe in more detail in the next Section, it has been shown that it is possible to compactly represent good approximates to the optimal agent policy [6] [9]. More recent work has used finite state machines to find offline solutions in partially observable domains [23]. In the next Section we describe finite state machines in more detail and develop an online solution strategy to partially observable problems, using finite state machines to model the behaviour of the other agents.

3.2.2 FSMs

A finite state machine can be used to represent an agent's policy. We have discussed fully-observable MDPs in which the agent's policy is decided on its immediate observations, and partially-observable MDPs in which the agent state is a continuous, high-dimensional belief-state derived from its entire history. A finite state machine policy falls between these two: the agent state is based on a variable length history. A fixed and finite number of agent states, more than the number of possible observations, are defined in the finite state machine and the agent moves from state to state of the machine based on its observation. In the next Section we describe this model in more detail and explain how representing agent policies with finite state machines can be used to develop approximate online solutions in partially observable multi-agent problems.

4 Bayesian learning approximation using finite state machines

In this Section we detail how to model agent policies using finite state automata (4.1 and 4.2). In Section 4.3 we then explain how these models fit with the multi-agent POMDP solution techniques described above, giving an algorithm for online learning (4.4) and explaining how this model extends previous work in the area. First, however, we begin with the definition of a finite state machine.

4.1 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 1): 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.

Algorithm 1 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 world state $s$, supplying
  agents with observations $o$.
• The agent moves along the edge associated with $o$, arriving at a new node $N'$.
• Repeat.
```

Now, in order to use finite state machines as representations of agent policies in unknown multi-agent scenarios, we will do two things: (1) to learn 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 4.4.

4.2 Learning FSMs

In principle, learning a deterministic finite state machine from a set of observations can proceed as follows [6]:

- **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, 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 (after applying the above algorithm to the 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 two problems: one is that observation strings can be of indefinite length, i.e. we may find ourselves storing the entire observation history in order to accurately build the FSM. The second is that although the FSM is a deterministic model, the behaviour it is modelling may be neither deterministic nor static. (A third issue is that we do not in fact know the observation strings, 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.

A point to note is that although we do not know the strategies of others or their optimal strategies, because we do know the MDP and the observation function, we can make some judgements about how much observation history is likely to be important in making decisions, providing us with a way of judging the optimal size of the FSMs.

We propose to sample possible observation strings from our belief state, and construct a candidate FSM for each sample, using the following tactics in learning these candidate FSMs:

- Define a maximum number of nodes which can occur in the FSM
- Break the observation history into overlapping observation strings of length l
- Assign each observation string a *likelihood* based on the frequency of occurrence and its sample probability, weighting more recent occurrences more highly. Discard completely observation strings older than n_t timesteps.
- Rather than resolve inconsistencies by always creating new nodes, resolve inconsistencies by appealing to the likelihood of each of the inconsistent strings, and discarding the least likely

In the next sections we describe in more detail an algorithm for learning FSMs from observation strings.

4.2.1 A polynomial FSM learning algorithm

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* 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 [6]. However, it is possible to learn compact FSMs in polynomial time, for many practical problems. The US-L* algorithm [6] has polynomial running time and has been shown to be effective at finding compact models of agent behaviour on small agent coordination problems—we propose to test it on larger problems.

This algorithm models the FSM using a table, with rows corresponding to observation string prefixes s, columns corresponding to string suffixes e, and the table entries corresponding to actions σ. The alphabet of possible observations is Σ. The table is then partitioned into equivalence classes:

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

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

- **consistent**: $\forall s_1, s_2 \in S, [C(s_1) = C(s_2) \rightarrow \forall t \in \Sigma, C(s_1 t) = C(s_2 t)]$.
- **closed** : $\forall s \in S\Sigma, \exists s' \in S, s \in S, C(s) \in C(s')$

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 readjust 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 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.

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. The next Section explains how this is done.

4.3 Online solutions: Best response

Previous work [38], [6] has considered 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 for every step.

By contrast, in our work, the state is not known. This adds to the complexity of the situation (as previously discussed), since even if the policies of the other agents are known, we do not know what observations they may make and consequently cannot determine their actions. We consider first this idealised case where the policies of the other agents are known. Now, we can compute a best response to any belief-state b using the Bellman equations, as discussed in Section 3:

$$Q_i(b,a) = \sum_{s'} P(s'|b,a)[r(s') + \gamma V(b')] \tag{7}$$

$$\text{where} \quad P(b'(s')|b,a) = \sum_{\mathbf{n_j},s',s} P(b'(s')|s,a \circ Act(\mathbf{n_j}))P(\mathbf{n_j},s|b) \tag{8}$$

$$\text{and} \quad P(b'(n'_j)|b,a) = \sum_{n_j,o',s} P(\mathbf{n'_j}|o')P(o'|s,a \circ Act(\mathbf{n_j}))P(\mathbf{n_j},s|b) \tag{9}$$

$$\text{where} \quad P(o'|s,a \circ Act(\mathbf{n_j})) = \sum_{s'} P(s'|s,a \circ Act(\mathbf{n_j}))P(o'|s') \tag{10}$$

($P(\mathbf{n'_j}|o')$ is 1 or 0.)

In an online algorithm, $Q_i(b,a)$ can be approximated from the current belief state by projecting k steps into the future. On the kth step, we replace $V(b')$ in equation 7 with some heuristic value. Possible heuristics include 0, a Q_{MDP}-based heuristic [19] or some domain-specific heuristic (for example, the expected distance from any agent to a goal, or visible future rewards such as victims which can be saved, in a disaster problem). Algorithm 2 outlines this best response solution. Such finite horizon algorithms have been used in related belief-state problems in many cases: in observable problems with unknown parameters (the heuristic is to assume that the current parameters are correct, and solve the corresponding MDP [8]), in finding offline solutions for networked POMDPs [23], and in online partially observable stochastic games [14].

Algorithm 2 Finite-horizon best response

- At timestep t, agent i has beliefs b over the state and the nodes n_j of the other agents j, and knows policies $n_j \rightarrow a_j$ for these agents.
- For some k, compute $Q_k(b,a)$ for each possible action a, using
  - $Q_k(b,a) = \sum_{s'} P(s'|b,a)[r(s') + \gamma V_{k-1}(b')]$
  - $V_k(b) = \max_a Q_k(b,a)$
  - $V_0(b,a) = \sum_{s'} P(s'|b,a)V_{heuristic}(s')$
- Execute the action a which maximises $Q_k(b,a)$

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 these policies, we propose to estimate the best response to the belief state by sampling from the possible policies to obtain a selection of sets of FSMs, $F = \{F_1, ...F_m\}$. For each sample FSM set F_s (containing a FSM for each other agent), the agent computes a best response action $BR_i(F_s,b)$. The action decision is then given by:

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

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

4.4 An online learning algorithm

We conclude this Section with a complete description of our algorithm, which brings together several of the techniques described above. This is an algorithm implemented by a single agent who 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 approximate models of the other agents in the form of finite state machines. A set of possible models is held in a belief state which is updated using Bayesian learning. At each step, the agent uses its observations to update each of the possible models and to update its beliefs about the world. It then computes the finite-horizon best response to each of these possible models and weights the possible responses with its belief in the corresponding model to decide on its action.

- 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, 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[1]
- **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 obser-

[1] 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.

vations 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 in Section 4.3.
- Perform the action a
- Repeat

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.

Thus, in this Section we have outlined a theoretical model for the online solution of partially observable multi-agent systems, based on the POMDP model, and then shown how we can approximate a particular (challenging) case of this model using finite state machines to model agent behaviour. In order to demonstrate the effectiveness of this model, we have implemented it on a rescue problem. In the next Section (5) we outline the problem before going on to describe our results (Section 6) and how they compare with the state of the art.

5 Model instantiation

In order to test the algorithm on a challenging problem, we implemented a rescue scenario involving coordinating ambulances. We compared our algorithm with a current state of the art algorithm and a hand-written solution for this problem. In this Section, we specify the problem as a multi-agent POMDP and explain how we simplify the observation space.

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). 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 5 shows one step on the grid for a 4x4 grid with three agents.

Specifically, taking the model of Section 3, the various parameters are instantiated in the following way:

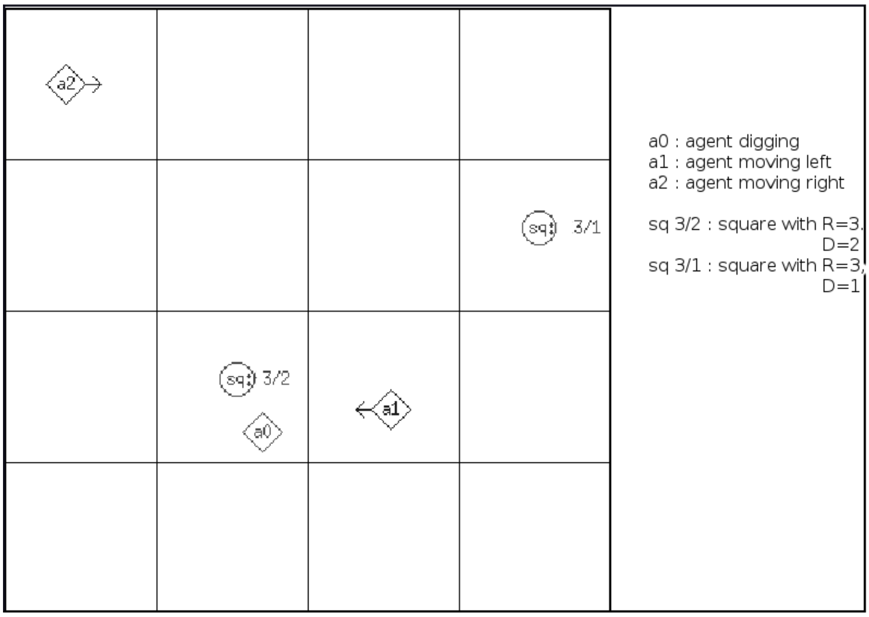


Fig. 5 One step of the rescue problem on a 4x4 grid with three agents

States: A state of this world is described by using a pair of variables for each of the grid squares, characterising the D and R values in the square (we 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 $l_d * l_r$ possible states, and for each agent there are $m * n$ possible states, making a total of $((l_d * l_r)^{(m*n)})$ possible states.

Agents: We assume that the number of agents, k, is fixed throughout each problem and known to each agent.

Locations: The location variable for each agent is its current square.

Actions: Agents may take Move actions (left, right, up or down), or Dig actions in their current square.

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”.

Transition function: Move actions move the agent one square in the requested direction, unless this is impossible in which case the action has no effect. Each square transitions (D,R) independently of other squares, so it is sufficient to define the transition function for one square. We use two global probabilities, p_d and p_r, to specify the probability of the D level changing (this is a constant probability independent of the action) and of the R level changing if there is a Dig action. If there is no dig action, R remains unchanged. We assume that if there are k digs in a square, they are concatenated. Finally, if a square is empty, 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,R) levels it has are determined with uniform probability (greater than 0).

Observation function: Agents are able to see the squares (deadness, rescue-level, and any other agents in the square) to the left and right, and above and below them, as well as their own square. 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 rescue-level R in the square with independent probability v. Since all agent actions are fully observable, we assume that we can also observe all agent locations. 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.

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.

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 i timesteps: for each property sq which may take values x,

$$P(sq_t = x_t | sq_{t-i} = x_{t-i}) = \sum_x P(sq_t = x_t | sq_{t-1} = x) P(sq_{t-1} = x | sq_{t-i} = x_{t-i})$$

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 $P(sq_0 = x_0)$ set to the problem-specific prior probabilities. Here, we assume that all squares are empty to begin with.

The above equations describe our beliefs about the world state: that is, 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. Using this model, we investigate the behaviour of our algorithm on the rescue problem.

6 Experimental evaluation

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 [14], described in Section 3 (“`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. Previous work on large dynamic rescue problems of a similar form [26] 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 [29]. It is therefore not an optimal strategy for the problem as we have stated it, but is a good approximation, thus providing a good target for our algorithm to meet.

Comparing against these two algorithms, and using the null policy in which agents move randomly, but never dig and so never effect any rescues (“`null`”) as a baseline, we investigate our algorithm, “`best response`” over different parameter settings on the rescue problem, and then focus on the scaling properties of the algorithm. Next, we identify the fixed parameters and then go on to our results.

6.1 Experimental setup

Following experimentation, we fix the following parameters: $l_d = l_r = 4$, $p_d = 0.15$, $p_r = 0.4$, $p_a = 0.05$, $v = 0.5$. In particular, we felt that the choice of four 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 and k as specified. We also experimented with increasing p_a towards problems where “dig nearby” becomes a viable strategy, and varying v. 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 [11]).

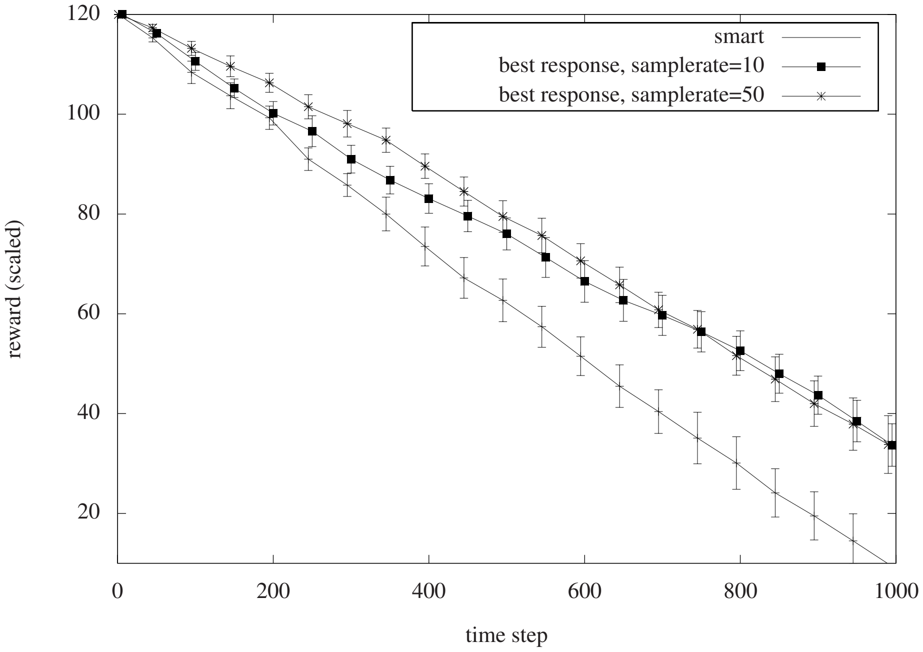
In every experiment, we carried out several runs of the problem, varying the initial placement of civilians and randomising their arrival and visibility. The same random seed was used to initialise each of the test algorithms in each run. The error bars included in the results show the 95% confidence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

6.2 Examining the learning rate

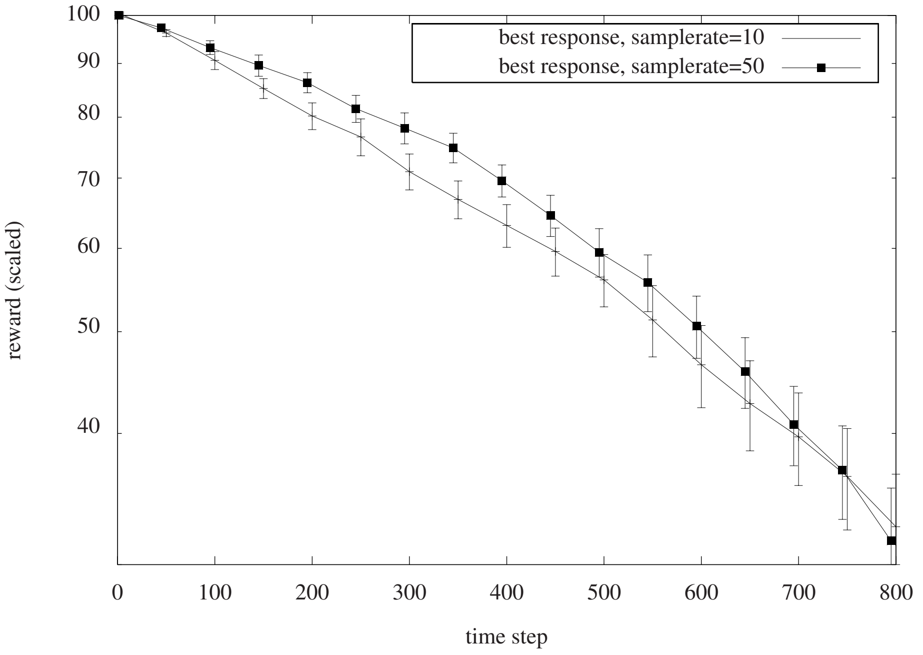
To begin with, we compared the algorithms over the course of 1000 steps on a 7x7 grid, with three agents. We found that the `POSG` algorithm, which is exponential in the number of agents, did not complete in any reasonable time (we consider one minute per step to be "reasonable") on this size of problem, taking ten minutes for one agent to complete a single step. Figure 6 shows the performance of 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 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(a) it is not clear that there is a large improvement in this advantage—that is, the lines are fairly straight. However, Figure 6(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



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



(b) Closeup of the first 800 steps

Fig. 6 Comparison of two algorithms over time on a 7x7 grid 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.

as the problem nature is changed. We also investigate parameters relating to the scale of the problem (number of agents, and size of grid). For each of these experiments we compare the total reward after 150 steps—from Figure 6 we can see that this is sufficient to show the differences between the algorithms or settings.

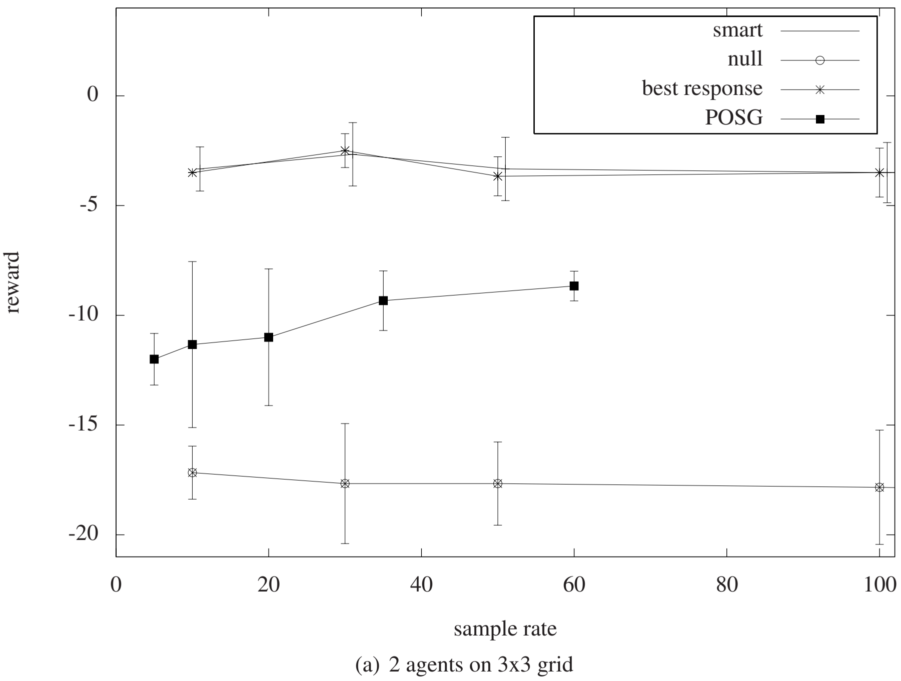
6.3 Varying the sampling rate

In order to examine how the best response algorithm will perform on challenge problems 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 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.

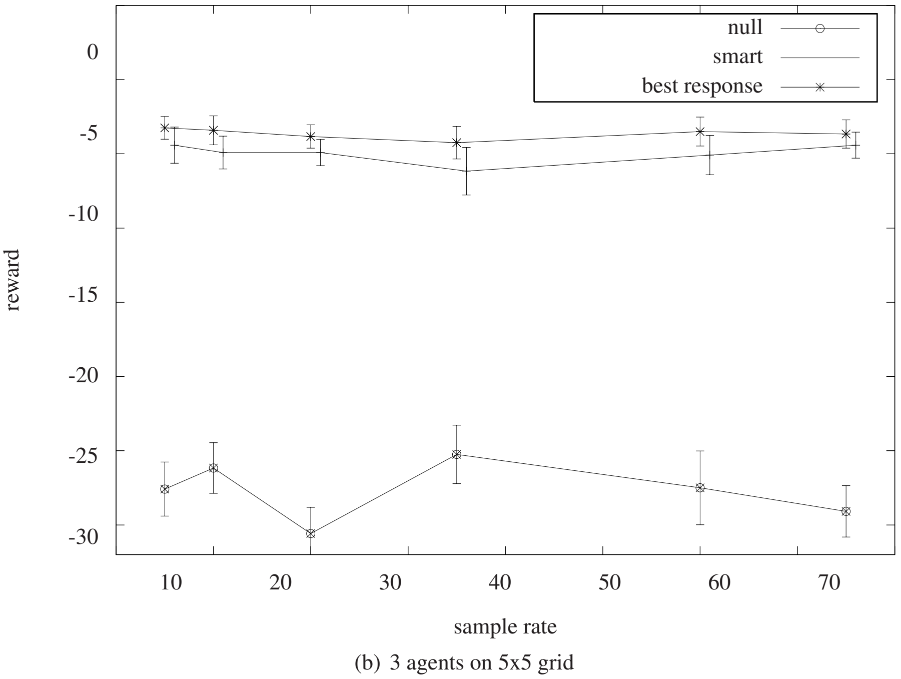
For the POSG algorithm, on a 3x3 grid with two agents, table 8(a) shows the time/sample-rate ratios for 100 steps (to the nearest minute). Since our cut-off was one step per minute, we did not run any tests on the POSG algorithm beyond a sample rate of 75, the null policy and the smart policy do not do any sampling. For our own policy, which does not need to iterate over all *joint* policies, the scaling factor was much better: table 8(b) shows the equivalent rates. The POSG algorithm is exponential in the number of agents, since it iterates over all joint actions. It therefore scales badly as the number of agents is increased. By contrast, table 8(c) shows the times for the best response algorithm running on the larger problem of a 7x7 grid with three agents. Even on this larger grid 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 indicated that the best response algorithm is able to perform quite well even at low sample rates.

To this end, Figure 7 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. By contrast, the POSG algorithm performs noticeably better as the number of samples is increased, and the error around the points reduces.

These results indicate that similar actions are selected even with a small number of samples, perhaps because the best response can be estimated well, and the best response 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. We do not investigate the POSG algorithm in the larger version of the problem (Figure 7(b)) but we see that as for the larger problems above, the best response algorithm slightly outperforms



(a) 2 agents on 3x3 grid



(b) 3 agents on 5x5 grid

Fig. 7 Effects of changing the sampling rate with two and three agents

3x3 grid, 2 agents	
Sample rate	Time
10	12 minutes
20	19 minutes
35	38 minutes
60	67 minutes
75	113 minutes

(a) POSG algorithm

3x3 grid, 2 agents	
Sample rate	Time
10	7 seconds
30	18 seconds
50	27 seconds
100	50 seconds
500	4 minutes

(b) best response algorithm

7x7 grid, 3 agents	
Sample rate	Time
10	18 seconds
35	48 seconds
60	5 minutes

(c) best response algorithm

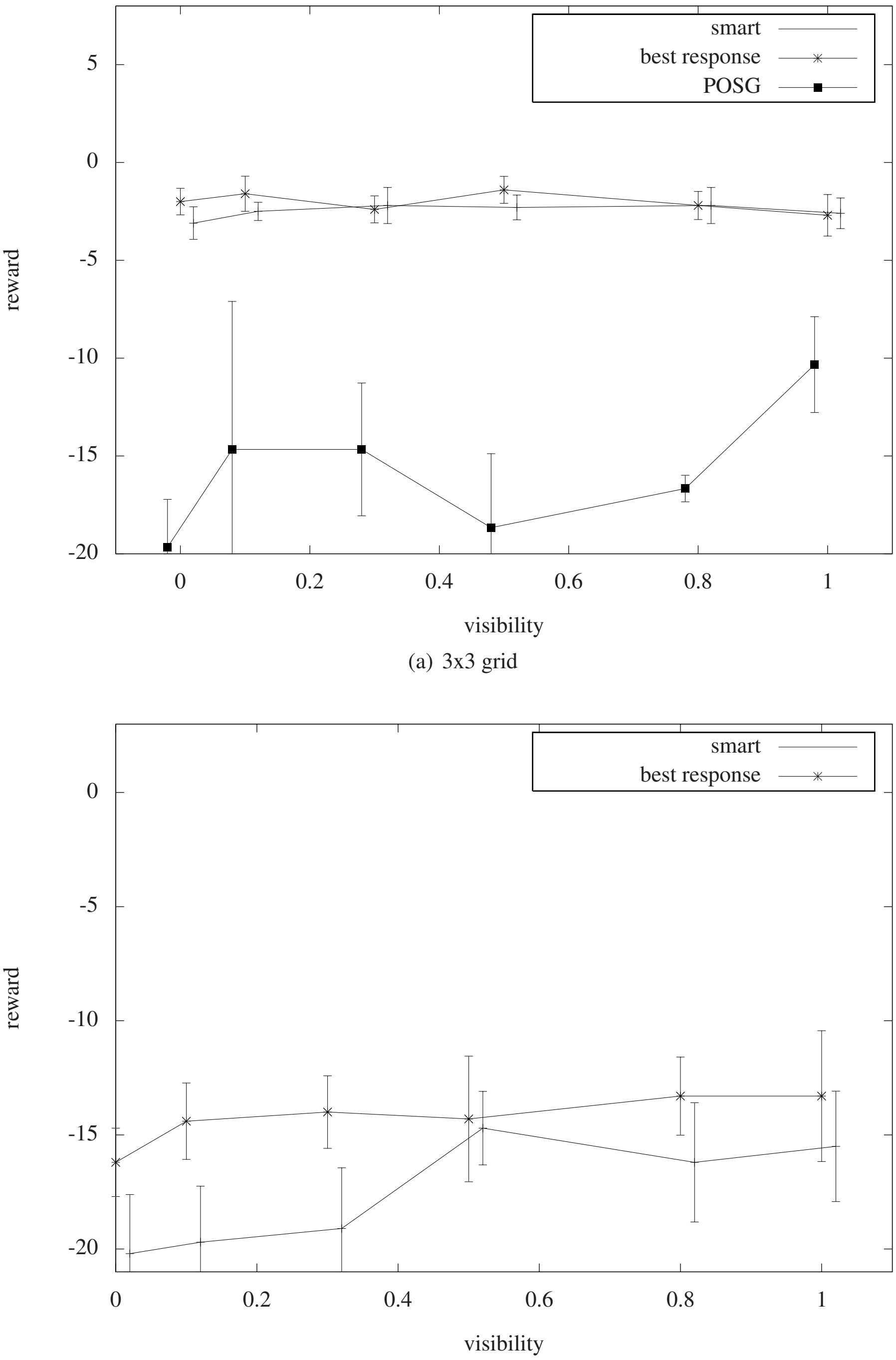
Fig. 8 Time taken to complete one run of 150 steps

the smart policy, due to its better handling of imperfect visibility. The next Section investigates the effects of visibility in more detail.

6.4 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% thus does not do any exploration actions.

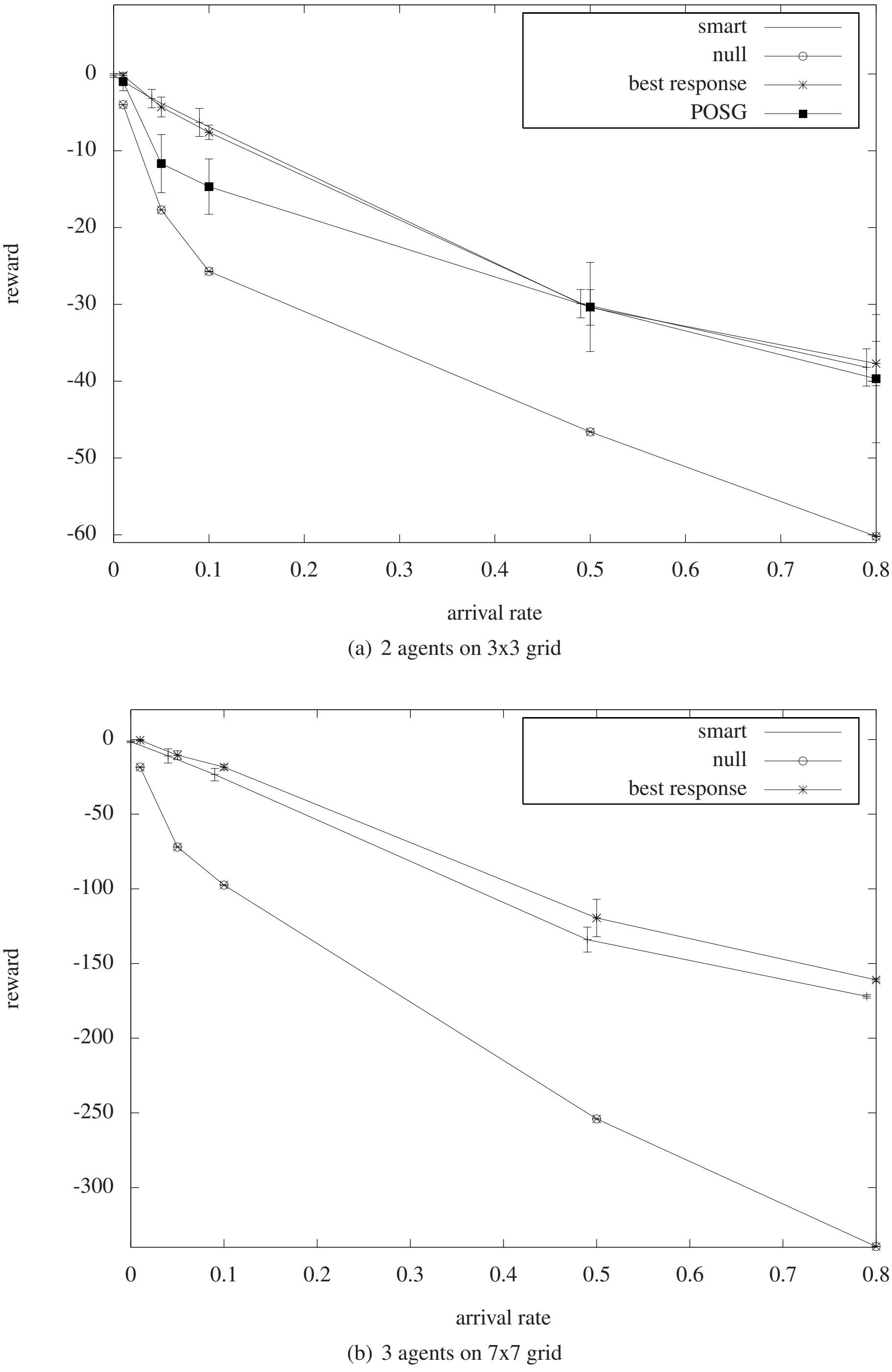
Figure 9 demonstrates the effects of varying visibility on a 3x3 grid and on a 7x7 grid, each with three agents. In Figure 9(a) we see the performance of the POSG algorithm is much worse than either the smart policy or the best response policy and fluctuating at lower visibilities, but noticeably improving 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 grid can do fairly well using the very simple strategy of digging where they see victims and can probably directly observe most of the grid between them. By contrast, Figure 9(b) shows the performance on the larger grid. We do not show the slow POSG algorithm on this problem; 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 performance of the best response policy as the visibility increases, although the best response policy continues to outperform the smart policy.



(a) 3x3 grid

(b) 7x7 grid

Fig. 9 Effects of varying visibility



(a) 2 agents on 3x3 grid

(b) 3 agents on 7x7 grid

Fig. 10 Effects of varying victim arrival rate

6.5 Varying the victim arrival rate

As well as varying the visibility, we can vary the problem by adjusting the 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. The reward for the `null` policy drops sharply—this is because there are more victims dying.

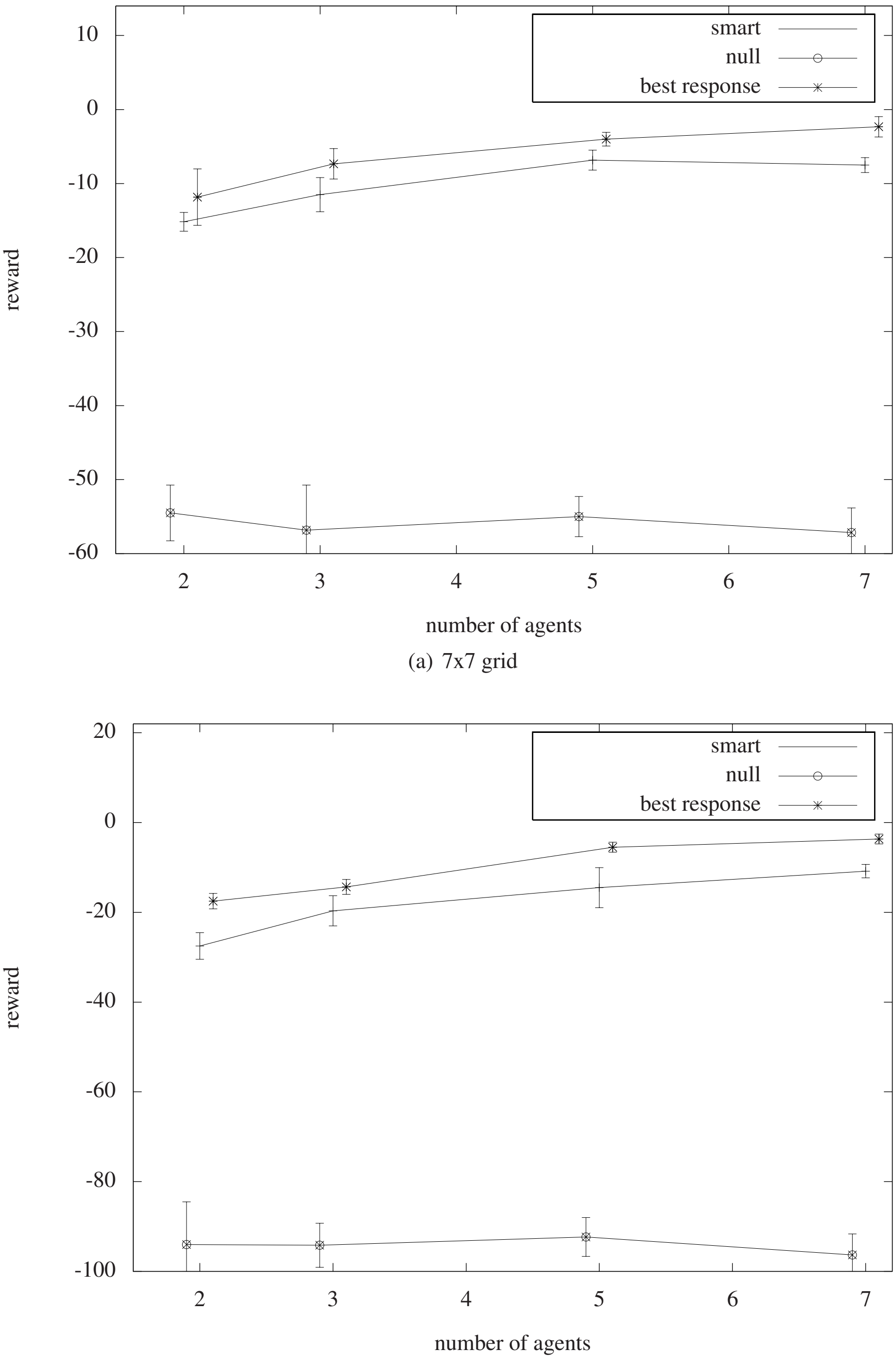
As the victim density increases, the optimal strategy approaches the very simple strategy of digging if there are any nearby victims. The point at which the simple strategy becomes optimal is indicated by the point where the `smart` policy stops making improvements over the `null` policy: between the 0.1 and 0.5 arrival rate on the small problem. The `best response` policy has matched the `smart` policy, and the `POSG` policy also catches up by the 0.5 data point. On the larger problem (Figure 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 6.1. We go on to investigate the scaling properties of the algorithms.

6.6 Varying scaling factors

The difficulty of the rescue problem scales exponentially with the size of the grid 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 agents were running on the same machine as one another and the environment; consequently, the memory requirements of the implementation scaled linearly with the number of agents. Nonetheless, we were able to test our algorithm on grids of up to 12x12 (2^{173} states), and with up to 7 agents (80,000 joint actions).

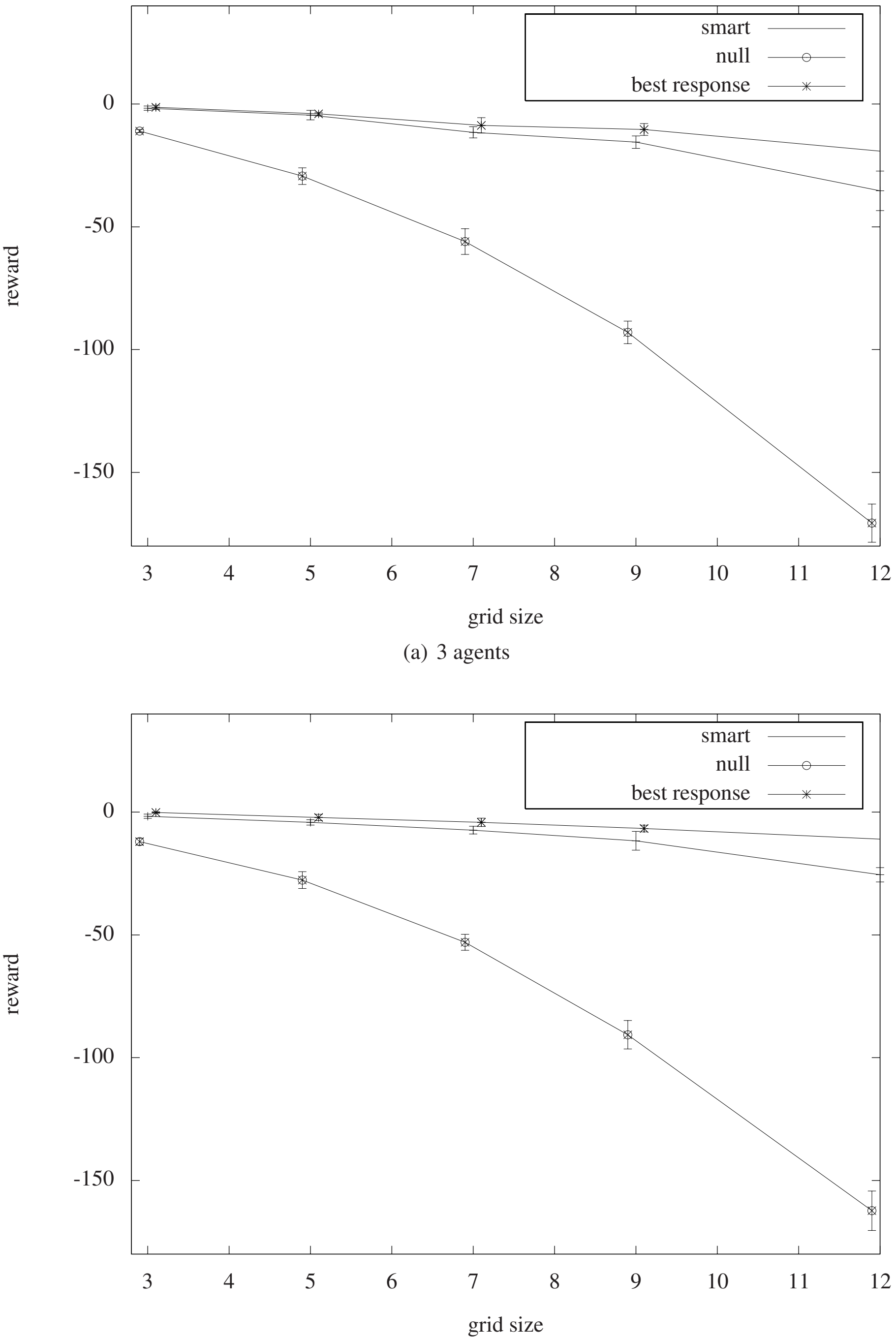
Now, although 7 agents is not a huge number for an algorithm which we would like to scale into dozens of agents, the primary limiting factor was the memory requirement for our implementation. Figure 11 shows the effect of increasing the number of agents on two larger grids, a 7x7 grid and a 9x9 grid. We observe that on the 7x7 grid 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 grid, although the `smart` policy does not saturate so much on the 9x9 grid—the larger problem space provides more room for improvement. Future work should involve a more efficient implementation, dividing the agents among several machines. We expect that the `best response` algorithm will then scale well as the number of agents is increased.



(a) 7x7 grid

(b) 9x9 grid

Fig. 11 Effects of increasing the number of agents on the results for two large grids



(a) 3 agents

(b) 5 agents

Fig. 12 Effects of changing the grid size on the results for 3 and for 5 agents

The `best response` algorithm also performs well on the large grids 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) grid. 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 grid size increases, a consequence of the way in which the `best response` policy incorporates uncertainty and the need for search on larger grids. The results are very similar for both three agents (Figure 12(a)) and five agents (Figure 12(b)) although, as expected, five agents are able to make more rescues than three agents (the lines are slightly flatter).

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. Although the best response algorithm typically has similar performance to the handwritten strategy, it is consistently outperforming it. 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. Since in general we anticipate our algorithm to be useful in scenarios where no good handwritten strategy is available, especially as the problem scales, the `best response` algorithm seems promising. Further improvements are discussed in the next Section.

7 Conclusions and Future Work

In summary, we have considered the problem of agent coordination in uncertain and partially observable systems. We developed an approach to this problem using a Bayesian learning mechanism, extending previous work on learning models of other agents, and demonstrated its effectiveness on a cooperative scenario from the disaster response domain. To emphasize, the novelties in this work lie in an extension of online model-based learning techniques into partially observable domains, using finite automata. As part of our theory, we outline a general Bayesian model of which our model forms a specific instantiation and show how other techniques, such as POMDPs and Bayesian learning, fit into this same model.

We have examined the performance of our 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. We therefore propose that future work should investigate more efficient implementations, and ways of distributing the problem across several machines—this is in any case a more accurate model of the problems of interest to us.

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. We discuss each of these in turn below.

Firstly, we propose to improve upon the learning of the FSM, using automatic state clustering. In the rescue problem, and in many other problems, groups of states can be considered equivalent by the agents. As a simple demonstration, note that there are several symmetries in our example problem: at every step the grid can be rotated until our agent is towards, say, the bottom right, dividing state space into equivalence classes with four states in each class, one corresponding to each rotation $(90^o, 180^o, 270^o, 0^o)$. More generally, we need only as many abstract states as there are joint actions, associating every underlying state with its optimal joint action. However, in practice, particularly if we plan to re-use parts of our model, reducing it purely to joint actions will be too abstract. An appropriate abstraction algorithm should be adaptable, allowing us to change our mind about which action is associated with a particular state, should allow us to update clusters incrementally and should not tie us to any predefined set of clusters. We propose to use a form of statistical clustering based on that described in [18] for this purpose.

A second area of improvement is to better exploit the information available to agents. We are investigating a complex problem domain in which some domain knowledge can be assumed. We may also be able to assume some level of rationality in the other agents (akin to coordination *conventions*). As we develop our models of the agents, we have discussed how we can use these models to improve our beliefs about the agents' observations, applying Bayes' rule. However, it may be possible to make more sophisticated belief updates by considering the observations which we make and the observations which other agents will make to be correlated streams of information. Techniques such as the Kalman Filter [40] are able to operate over correlated streams of information to make more accurate estimates about the value of any particular point and to estimate missing data [25]. These techniques could be applied (with caution) to our estimates of the observations of the other agents and of the current state.

Thirdly, 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, 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. Indeed, related work [8] [30] has done this for some special cases.

In this context, given the uncertainties of our domain, it is clear that if the behaviour of the other agents is completely unknown, **and** the current state is unknown, **and** the environmental parameters are completely unknown, an agent must stumble around “in the dark” for some considerable time before it can begin to get a handle on good or optimal behaviour. However, in the typical scenarios motivated by our example domain of disaster response, an agent will have strong prior information about some or all of the unknown parameters. For example, the other agents may be assumed to be rational and cooperative, thus likely to behave in a near-optimal way. In our example problem, the form of the transition function may be known, but not the exact values of every parameter. By incorporating all the information available to the agent into its model, and particularly by correlating information, we anticipate that our model will be able to handle problems in which the environmental dynamics are not completely known using the theoretical form laid out in Section 3.

Following on from this, our model will easily handle scenarios in which the number of agents changes (but is known to our agent) over time. Since the best response is computed at each step, there will be no difficulty in computing a best response over a subset of the other agents, or in adding a new agent model to the collection. Our agent will adapt continually during the problem run. Similarly, if the agent is learning the environmental dynamics, and those dynamics change, the agent should adjust its model smoothly.

With these improvements, we anticipate that the model of Section 3 can be used as the basis of an algorithm capable of solving medium-sized distributed collaborative problems in the real world, such as traffic management, controlling search robots in a building after a fire, or distributing ambulances during a disaster. Similar algorithms could also be included in software which could be loaded onto handheld devices to aid human decision-making during critical situations such as war or a large-scale disaster.

Acknowledgements Thanks to Georgios Chalkiadakis and Zinovi Rabovich for discussions on the early versions of this work.

References

1. D. Aberdeen and J. Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the 19th International Conference on Machine Learning*, volume 2, pages 3–10, Sydney, Australia, 2002. Morgan Kaufmann.
2. O. Abul, F. Polat, and R. Alhajj. Multiagent reinforcement learning using function approximation. In *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, volume 30, pages 485–497, 2000.
3. C. Amato, D. S. Bernstein, and S. Zilberstein. Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 341–343, New York, NY, USA, 2006. ACM Press.
4. C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
5. M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *International Joint Conferences on Artificial Intelligence*, pages 1021–1026, 2001.

6. D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 2, pages 62–67, Portland, Oregon, 1996.
7. A. Cassandra, M. Littman, and N. Zhang. 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*, pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann.
8. G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: a Bayesian approach. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 709–716, New York, NY, USA, 2003. ACM Press.
9. A. Clark and F. Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, 2004.
10. C. Claus and C. Boutilier. 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*, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.
11. R. Dearden, N. Friedman, and D. Andre. Model-based Bayesian exploration. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 150–15, San Francisco, CA, 1999. Morgan Kaufmann.
12. E. H. Durfee. Practically coordinating. *AI Magazine*, 20(1):99–116, 1999.
13. P. S. Dutta, S. Dasmahapatra, S. R. Gunn, N. Jennings, and L. Moreau. Cooperative information sharing to improve distributed learning. In *Proceedings of the AAMAS 2004 workshop on Learning and Evolution in Agent-Based Systems*, pages 18–23, 2004.
14. R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 136–143, Washington, DC, USA, 2004. IEEE Computer Society.
15. F. Fischer, M. Rovatsos, and G. Weiss. Hierarchical reinforcement learning in communication-mediated multiagent coordination. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1334–1335, Washington, DC, USA, 2004. IEEE Computer Society.
16. D. Fitoussi and M. Tennenholtz. Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, 119(1-2):61–101, 2000.
17. D. Fudenberg and D. K. Levine. *The Theory of Learning in Games*. MIT Press, 1998.
18. J. Hoar. Reinforcement learning applied to a real robot task. DAI MSc Dissertion, University of Edinburgh, 1996.
19. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.
20. Y. Kim, R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Exploiting locality of interaction in networked distributed pomdps. In *Proceedings of the AAAI Spring Symposium on "Distributed Plan and Schedule Management"*, 2006.
21. D. Leslie. *Reinforcement learning in games*. PhD thesis, University of Bristol, 2004.
22. M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann.
23. J. Marecki, T. Gupta, P. Varakantham, and M. Tambe. 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*, 2008.
24. National Research Council. *Summary of a Workshop on Using Information Technology to enhance Disaster Management*. National Academies Press, 2005.
25. M. A. Osborne, A. Rogers, S. Ramchurn, S. J. Roberts, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In *International Conference on Information Processing in Sensor Networks*, pages 109–120, April 2008.

26. S. Paquet, L. Tobin, and B. Chaib-draa. An online POMDP algorithm for complex multiagent environments. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 970–977, New York, NY, USA, 2005. ACM Press.
27. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, pages 1025 – 1032, August 2003.
28. P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 697–704, New York, NY, USA, 2006. ACM.
29. K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Compututers*, 38(8):1110–1123, 1989.
30. S. Ross, B. Chaib-draa, and J. Pineau. Bayes-adaptive POMDPs. In *Neural Information Processing Systems*, page In press, 2008.
31. N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing*, pages 1043–1049, Vancouver, Canada, December 2002.
32. P. Scerri, E. Liao, Y. Xu, M. Lewis, G. Lai, and K. Sycara. Coordinating very large groups of wide area search munitions. *Theory and Algorithms for Cooperative Systems*, 2005.
33. P. Scerri, K. Sycara, and M. Tambe. Adjustable autonomy in the context of coordination. In *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, 2004. Invited Paper.
34. G. Shani, R. I. Brafman, and S. E. Shimony. Model-based online learning of POMDPs. In *European Conference on Machine Learning*, pages 353–364, 2005.
35. A. J. Smith. Dynamic generalisation of continuous action spaces in reinforcement learning: A neurally inspired approach, 2002. Ph.D. thesis, Division of Informatics, Edinburgh University, UK.
36. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.
37. M. Tambe, J. Adibi, Y. Alonaizon, A. Erdem, G. A. Kaminka, S. Marsella, and I. Muslea. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110(2):215–239, 1999.
38. T. Vu, R. Powers, and Y. Shoham. Learning against multiple opponents. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 752–759, New York, NY, USA, 2006. ACM.
39. F. Wang. Self-organising communities formed by middle agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1333–1339, New York, NY, USA, 2002. ACM Press.
40. G. Welch and G. Bishop. An introduction to the Kalman filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.
41. M. Wooldridge. *An Introduction to Multi-agent Systems.* Wiley, 2002.

Index