Approximate Solutions For Partially Observable Stochastic Games with Common Payoffs

Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{*remery,ggordon,schneide*}*@cs.cmu.edu*

Sebastian Thrun
Stanford AI Lab
Stanford University
Stanford, CA 94305
thrun@stanford.edu

Abstract

Partially observable decentralized decision making in robot teams is fundamentally different from decision making in fully observable problems. Team members cannot simply apply single-agent solution techniques in parallel. Instead, we must turn to game theoretic frameworks to correctly model the problem. While partially observable stochastic games (POSGs) provide a solution model for decentralized robot teams, this model quickly becomes intractable. We propose an algorithm that approximates POSGs as a series of smaller, related Bayesian games, using heuristics such as Q_{MDP} to provide the future discounted value of actions. This algorithm trades off limited look-ahead in uncertainty for computational feasibility, and results in policies that are locally optimal with respect to the selected heuristic. Empirical results are provided for both a simple problem for which the full POSG can also be constructed, as well as more complex, robot-inspired, problems.

1. Introduction

Partially observable problems, those in which agents do not have full access to the world state at every timestep, are very common in robotics applications where robots have limited and noisy sensors. While partially observable Markov decision processes (POMDPs) have been successfully applied to single robot problems [11], this framework does not generalize well to teams of decentralized robots. All of the robots would have to maintain parallel POMDPs representing the teams' joint belief state, taking all other robot observations as input. Not only does this algorithm scale exponentially with the number of robots and observations, it requires unlimited bandwidth and instantaneous communication between the robots. If no communication is available to the team, this approach is no longer correct: each robot will acquire different observations, leading to different belief states that cannot be reconciled. Therefore, we conjecture that robots with limited knowledge of their teammates' observations must reason about all possible belief states that could be held by teammates and how that affects their own action selection. This reasoning is tied to the similarity in the relationship between partially observable stochastic games (POSGs) and POMDPs to the relationship between POMDPs and Markov decision processes (MDPs) (see Figure 1). Much in the same way that a POMDP cannot be solved by randomizing over solutions to different MDPs, it is not sufficient to find a policy for a POSG by having each robot solve parallel POMDPs with reasonable assumptions about teammates' experiences. Additional structure is necessary to find solutions that correctly take into account different experiences.

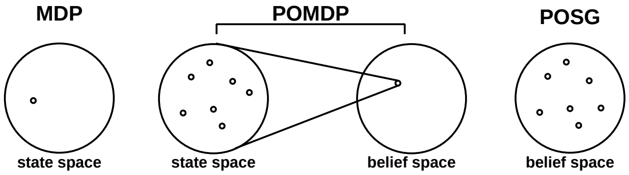

Figure 1. A representation of the relationship between MDPs, POMDPs and POSGs.

We demonstrate this with a simple example. Consider a team of paramedic robots in a large city. When a distress signal, e, is received by emergency services, a radio call, rc, is sent to the two members of the paramedic team. Each robot independently receives that call with probability 0.9 and must decide whether or not to respond to the call. If both robots respond to the emergency then it is dealt with (cost of 0), but if only one robot responds then the incident is unresolved with a known cost c that is either 1.0 or 100.0 with equal probability (low or high cost emergencies). Alternatively, a robot that receives a radio call may relay that a call to its teammate at a cost d of 1.0 and both robots then respond to the emergency. What should a robot do when it receives a radio call?

If the two robots run POMDPs in parallel in order to decide their policies then they have to make assumptions about their teammates' observation and policy in order to make a decision. First, robot 1 can assume that robot 2 always receives and responds to the radio call when robot 1 receives that call. Robot 1 should therefore also always respond to the call, resulting in a cost of 5.05 ($0.9c$). Alternatively, robot 1 can assume robot 2 receives no information and so it relays the radio call because otherwise it believes that robot 2 will never respond to the emergency. This policy has a cost of 1.0. A third possibility is for each robot to calculate the policy for both of these POMDPs and then randomize over them in proportion to the likelihood that each POMDP is valid. This, however, leads to a cost of 4.654. The robots can do better by considering the distribution over their teammate's belief state directly in policy construction. A policy of relaying a radio call when $d < 0.9c$ and otherwise just responding to the call has a cost of 0.55 which is better than any of the POMDP variants.

The POSG is a game theoretic approach to multi-agent decision making that implicitly models a distribution over other agents' belief states. It is, however, intractable to solve large POSGs, and we propose that a viable alternative is for agents to interleave planning and execution by building and solving a smaller approximate game at every timestep to generate actions. Our algorithm will use information common to the team (e.g. problem dynamics) to approximate the policy of a POSG as a concatenation of a series of policies for smaller, related Bayesian games. In turn, each of these Bayesian games will use heuristic functions to evaluate the utility of future states in order to keep policy computation tractable.

This transformation is similar to the classic one-step lookahead strategy for fully-observable games [13]: the agents perform full, game-theoretic reasoning about their current knowledge and first action choice but then use a predefined heuristic function to evaluate the quality of the resulting outcomes. The resulting policies are always coordinated; and, so long as the heuristic function is fairly accurate, they perform well in practice. A similar approach, proposed by Shi and Littman [14], is used to find near-optimal solutions for a scaled down version of Texas Hold'Em.

2. Partially Observable Stochastic Games

Stochastic games, a generalization of both repeated games and MDPs, provide a framework for decentralized action selection [4]. POSGs are the extension of this framework to handle uncertainty in world state. A POSG is defined as a tuple (I, S, A, Z, T, R, O). $I = \{1, ..., n\}$ is the set of agents, S is the set of states and A and Z are respectively the cross-product of the action and observation space for each agent, i.e. $A = A_1 \times \cdots \times A_n$. T is the transition function, $T : S \times A \rightarrow S$, R is the reward function, $R : S \times A \rightarrow \Re$ and O defines the observation emission probabilities, $O : S \times A \times Z \rightarrow [0, 1]$. At each timestep of a POSG the agents simultaneously choose actions and receive a reward and observation. In this paper, we limit ourselves to finite POSGs with common payoffs (each agent has an identical reward function R).

Agents choose their actions by solving the POSG to find a policy σ_i for each agent that defines a probability distribution over the actions it should take at each timestep. We will use the Pareto-optimal Nash equilibrium as our solution concept for POSGs with common payoffs. A Nash equilibrium of a game is a set of strategies (or policies) for each agent in the game such that no one agent can improve its performance by unilaterally changing its own strategy [4]. In a common payoff game, there will exist a Pareto-optimal Nash equilibrium which is a set of best response strategies that maximize the payoffs to all agents. Coordination protocols [2] can be used to resolve ambiguity if multiple Pareto-optimal Nash equilibria exist.

The paramedic problem presented in Section 1 can be represented as a POSG with: $I = \{1, 2\}$, $S = \{e, \overline{e}\}$; $A_i = \{$*respond,relay-call,do-nothing*$\}$; and $Z_i = \{rc, \overline{rc}\}$. The reward function is defined as a cost c if both robots do not respond to an emergency and a cost d of relaying a radio call. T and O can only be defined if the probability of an emergency is given. If they were given, each robot could then find a Pareto-optimal Nash equilibrium of the game.

There are several ways to represent a POSG in order to find a Nash equilibrium. One useful way is as an extensive form game [4] which is an augmented game tree. Extensive form games can then be converted to the normal form [4] (exponential in the size of the game tree) or the sequence form [15, 6] (linear in the size of the game tree) and, in theory, solved. While this is non-trivial in arbitrary n-player games, to solve games with common payoff problems, we can use an alternating-maximization algorithm to find locally optimal best response strategies. Holding the strategies of all but one agent fixed, linear or dynamic programming is used to find a best response strategy for the remaining agent. We optimize for each agent in turn until no agent wishes to change its strategy. The resulting equilibrium point is a local optimum; so, we use random restarts to explore the strategy space.

Extensive form game trees provide a theoretically sound framework for representing and solving POSGs; however, because the action and observation space are exponential in the number of agents, for even the smallest problems these trees rapidly become too large to solve. Instead, we turn to more tractable representations that trade off global optimality for locally optimal solutions that can be found efficiently.

3. Bayesian Games

We will be approximating POSGs as a series of Bayesian games. As it will be useful to have an understanding of Bayesian games before describing our transformation, we first present an overview of this game theory model.

Bayesian games model single state problems in which each agent has private information about something relevant to the decision making process [4]. While this private information can range from knowledge about the number of agents in the game to the action set of other agents, in general it can be represented as uncertainty about the utility (or payoffs) of the game. In other words, utility depends on this private information.

For example, in the paramedic problem, the private information known by robot 1 is whether or not it has heard a radio call. If both robots know whether or not each other received the call, then the payoff of the game is known with certainty and action selection is straightforward. But as this is not the case, each robot has uncertainty over the payoff.

In the game theory literature, the private information held by an agent is called its *type*, and it encapsulates all non-commonly-known information to which the agent has access. The set of possible types for an agent can be infinite, but we limit ourselves to games with finite sets. Each agent knows its own type with certainty but not those of other agents; however, beliefs about the types of others are given by a commonly held probability distribution over joint types.

If $I = \{1, ..., n\}$ is the set of agents, θ_i is a type of agent i and Θ_i is the type space of agent i such that $\theta_i \in \Theta_i$, then a type profile θ is $\theta = \{\theta_i, ..., \theta_n\}$ and the type profile space is $\Theta = \Theta_1 \times \cdots \times \Theta_n$. θ_{-i} is used to represent the type of all agents but i, with Θ_{-i} defined similarly. A probability distribution, $p \in \Delta(\Theta)$, over the type profile space Θ is used to assign types to agents, and this probability distribution is assumed to be commonly known. From this probability distribution, the marginal distribution over each agent's type, $p_i \in \Delta(\Theta_i)$, can be recovered as can the conditional probabilities $p_i(\theta_{-i}|\theta_i)$.

In the paramedic example $\Theta_i = \{rc, \overline{rc}\}$. The conditional probability $p_i(rc_j|rc_i) = 0.9$ and $p_i(\overline{rc_j}|rc_i) = 0.1$. The description of the problem does not give enough information to assign a probability distribution over $\Theta = \{\{rc, rc\}, \{rc, \overline{rc}\}, \{\overline{rc}, rc\}, \{\overline{rc}, \overline{rc}\}\}$.

The utility, u, of an action a_i to an agent is dependent on the actions selected by all agents as well as on their type profile and is defined as $u_i(a_i, a_{-i}, (\theta_i, \theta_{-i}))$. By definition, an agent's strategy must assign an action for every one of its possible types even though it will only be assigned one type. For example, although robot 1 in the paramedic example either receives a call, rc, or does not, its strategy must still define what it would do in either case. Let an agent's strategy be σ_i, and the probability distribution over actions it assigns for θ_i be given by $p(a_i|\theta_i) = \sigma_i(\theta_i)$.

Formally, a Bayesian game is a tuple (I, Θ, A, p, u) where $A = A_1 \times \cdots \times A_n$, $u = \{u_i, ..., u_n\}$ and I, Θ and p are as defined above. Given the commonly held prior $p(\Theta)$ over the distribution over agents' types, each agent can calculate a Bayesian-Nash equilibrium policy. This is a set of best response strategies σ in which each agent maximizes its expected utility conditioned on its probability distribution over the other agents' types. In this equilibria, each agent has a policy σ_i that, given σ_{-i}, maximizes the expected value $u(\sigma_i, \theta_i, \sigma_{-i}) = \sum_{\theta_{-i} \in \Theta_{-i}} p_i(\theta_{-i}|\theta_i) u_i(\sigma_i(\theta_i), \sigma_{-i}(\theta_{-i}), (\theta_i, \theta_{-i}))$. Agents know σ_{-i} because an equilibrium solution is defined as σ, not just σ_i.

Bayesian-Nash equilibria are found using similar techniques as for Nash equilbria in an extensive form game. For a common-payoff Bayesian game where $u_i = u_j \forall i, j$, we can find a solution by applying the sequence form and the alternating maximization algorithm as described in Section 2.

4. Bayesian Game Approximation

We propose an algorithm for finding an approximate solution to a POSG with common payoffs that transforms the original problem into a sequence of smaller Bayesian games that are computationally tractable. So long as each agent is able to build and solve the same sequence of games, the team can coordinate on action selection. Theoretically, our algorithm will allow us to handle finite horizon problems of indefinite length by interleaving planning and execution. While we will formally describe our approximation in Algorithms 1 and 2, we first present an informal overview of the approach.

In Figure 2, we show a high-level diagram of our algorithm. We approximate the entire POSG, shown by the outer triangle, by constructing a smaller game at each timestep. Each game models a subset of the possible experiences occurring up to that point and then finds a one-step lookahead policy for each agent contingent on that subset.

The question is now how to convert the POSG into these smaller slices. First, in order to model a single time slice, it is necessary to be able to represent each sub-path through the tree up to that timestep as a single entity. A single subpath through the tree corresponds to a specific set of observation and action histories up to time t for all agents in the team. If all agents know that a specific path has occurred, then the problem becomes fully observable and the payoffs of taking each joint action at time t are known with certainty. This is analogous to the utility in Bayesian games being conditioned on specific type profiles, and so we model each smaller game as a Bayesian game. Each path in the POSG up to time t is now represented as a specific type profile, θ^t, with the the type of each agent, corresponding to its own observation and actions, as θ_i^t. A single θ_i^t may appear in multiple θ^t. For example, in the paramedic problem, robot 1's type $\theta_1 = rc$ appears in both $\theta = \{rc, rc\}$ and $\theta = \{rc, \overline{rc}\}$.

Agents can now condition their policies on their individual observation and action histories; however, there are still two pieces of the Bayesian game model left to define before

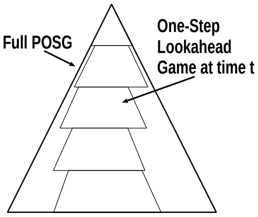

Figure 2. A high level representation of our algorithm for approximating POSGs.

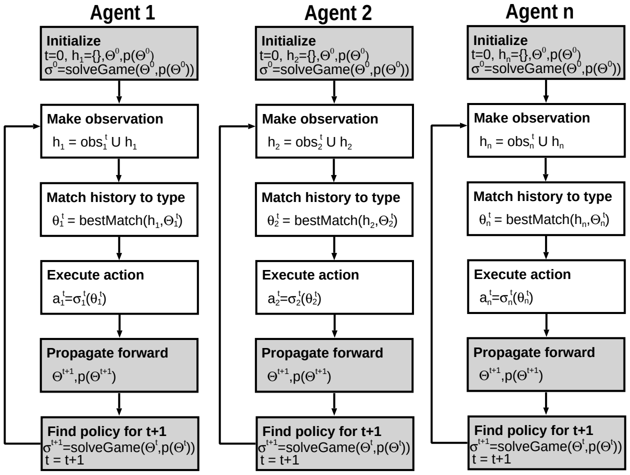
Agent 1
Initialize
$t=0,\ h_1=\{\},\Theta^0,p(\Theta^0)$
$\sigma^0=\mathrm{solveGame}(\Theta^0,p(\Theta^0))$
Make observation
$h_1 = obs_1^t \cup h_1$
Match history to type
$\theta_1^t = \mathrm{bestMatch}(h_1,\Theta_1^t)$
Execute action
$a_1^t=\sigma_1^t(\theta_1^t)$
Propagate forward
$\Theta^{t+1},p(\Theta^{t+1})$
Find policy for t+1
$\sigma^{t+1}=\mathrm{solveGame}(\Theta^t,p(\Theta^t))$
$t = t+1$
Agent 2
Initialize
$t=0,\ h_2=\{\},\Theta^0,p(\Theta^0)$
$\sigma^0=\mathrm{solveGame}(\Theta^0,p(\Theta^0))$
Make observation
$h_2 = obs_2^t \cup h_2$
Match history to type
$\theta_2^t = \mathrm{bestMatch}(h_2,\Theta_2^t)$
Execute action
$a_2^t=\sigma_2^t(\theta_2^t)$
Propagate forward
$\Theta^{t+1},p(\Theta^{t+1})$
Find policy for t+1
$\sigma^{t+1}=\mathrm{solveGame}(\Theta^t,p(\Theta^t))$
$t = t+1$
Agent n
Initialize
$t=0,\ h_n=\{\},\Theta^0,p(\Theta^0)$
$\sigma^0=\mathrm{solveGame}(\Theta^0,p(\Theta^0))$
Make observation
$h_n = obs_n^t \cup h_n$
Match history to type
$\theta_n^t = \mathrm{bestMatch}(h_n,\Theta_n^t)$
Execute action
$a_n^t=\sigma_n^t(\theta_n^t)$
Propagate forward
$\Theta^{t+1},p(\Theta^{t+1})$
Find policy for t+1
$\sigma^{t+1}=\mathrm{solveGame}(\Theta^t,p(\Theta^t))$
$t = t+1$

Figure 3. A diagram showing the parallel nature of our algorithm. Shaded boxes indicate steps that have the same outcome for all agents.

these policies can be found. First, the Bayesian game model requires agents to have a common prior over the type profile space Θ. If we assume that all agents have common knowledge of the starting conditions of the original POSG, i.e. a probability distribution over possible starting states, then our algorithm can iteratively find Θ^{t+1} and $p(\Theta^{t+1})$ using information from Θ^t, $p(\Theta^t)$, A, T, Z, and O. Additionally, because the solution of a game, σ, is a set of policies for all agents, this means that each agent not only knows its own next-step policy but also those of its teammates and this information can be used to update the type profile space. As the set of all possible histories at time t can be quite large, we discard all types with prior probability less than some threshold and renormalize $p(\Theta)$.

Second, it is necessary to define a utility function u that represents the payoff of actions conditioned on the type profile space. Ideally, a utility function u should represent not only the immediate value of a joint action but also its expected future value. In finite horizon MDPs and POMDPs, these future values are found by backing up the value of actions at the final timestep back through time. In our algorithm, this would correspond to finding the solution to the Bayesian game representing the final timestep T of the problem, using that as the future value of the game at the time $T-1$, solving that Bayesian game and then backing up that value through the tree. Unfortunately, this is intractable; it requires us to do as much work as solving the original POSG because we do not know a specific probability distribution over Θ^T until we have solved the game for timesteps 0 through $T-1$ and we cannot take advantage of any pruning of Θ^t. Instead, we will perform our one-step lookahead using a heuristic function for u, resulting in policies that are locally optimal with respect to that heuristic. We will use heuristic functions that try to capture some notion of future value while remaining tractable to calculate.

Now that the Bayesian game is fully defined, the agents can iteratively build and solve these games for each timestep. The t-th Bayesian game is converted into its corresponding sequence form and a locally optimal Bayes-Nash equilibrium found by using the alternating-maximization algorithm. Once a set of best-response strategies is found at time t, each agent i then matches its true history of observations and actions, h_i, to one of the types in its type space Θ_i and then selects an action based on its policy, σ_i^t, and type, θ_i^t.[1]

This Bayesian game approximation runs in parallel on all agents in the team as shown in Figure 3. So long as a mechanism exists to ensure that each agent finds the same set of best-response policies σ^t, then each agent will maintain the same type profile space for the team as well as the same probability distribution over that type space without having to communicate. This ensures that the common prior assumption is always met. Shading is used in Figure 3 to indicate which steps of the algorithm should result in the same output for each agent. Agents run the algorithm in lock-step and a synchronized random-number generator is used to ensure all agents come up with the same set of best-response policies using the alternating-maximization algorithm. Only common knowledge arising from these policies or domain specific information (e.g. agents always know the position of their teammates) is used to update individual and joint type spaces. This includes pruning of types due to low probability as pruning thresholds are common to the team. The non-shaded boxes in Figure 3 indicate steps in which agents only use their own local information to make a decision. These steps involve matching an agent's true observation history, h_i to a type modeled in the game, θ_i^t, and then executing an action based on that type.

In practice there are two possible difficulties with constructing and solving our Bayesian game approximation of the POSG. The first is that the type space Θ can be extremely large, and the second is that it can be difficult to

[1] If the agent's true history was pruned, we map the actual history to the nearest non-pruned history using Hamming distance as our metric. This minimization should be done in a more appropriate space such as the belief space, and is future work.

Algorithm 1: $PolicyConstructionAndExecution$

```
$I, \Theta^0, A, p(\Theta^0), Z, S, T, R, O$
**Output**: $r, s^t, \sigma^t, \forall t$
**begin**
    $h_i \longleftarrow \emptyset, \forall i \in I$
    $r \longleftarrow 0$
    $initializeState(s^0)$
    **for** $t \leftarrow 0$ ***to*** $t_{max}$ **do**
        **for** $i \in I$ **do (in parallel)**
            $setRandSeed(rs^t)$
            $\sigma^t, \Theta^{t+1}, p(\Theta^{t+1}) \leftarrow$
            $BayesianGame(I, \Theta^t, A, p(\Theta^t), Z, S, T, R, O, rs^t)$
            $h_i \leftarrow h_i \cup z_i^t \cup a_i^{t-1}$
            $\theta_i^t \leftarrow matchToType(h_i, \Theta_i^t)$
            $a_i^t \leftarrow \sigma_i^t(\theta_i^t)$
        $s^{t+1} \leftarrow T(s^t, a_1^t, ..., a_n^t)$
        $r \leftarrow r + R(s^t, a_1^t, ..., a_n^t)$
**end**
```

find a good heuristic for u.

We can exert some control over the size of the type space by choosing the probability threshold for pruning unlikely histories. The benefit of using such a large type space is that we can guarantee that each agent has sufficient information to independently construct the same set of histories Θ^t and the same probability distribution over Θ^t at each timestep t. In the future, we plan on investigating ways of sparsely populating the type profile space or possibly recreating comparable type profile spaces and probability distributions from each agent's own belief space.

Providing a good u that allows the Bayesian game to find high-quality actions is more difficult because it depends on the system designer's domain knowledge. We have had good results in some domains with a Q_{MDP} heuristic [7]. For this heuristic, $u(a, \theta)$ is the value of the joint action a when executed in the belief state defined by θ assuming that the problem becomes fully observable after one timestep. This is equivalent to the Q_{MDP} value of that action in that belief state. To calculate the Q_{MDP} values it is necessary to find Q-values for a fully observable version of the problem. This can be done by dynamic programming or Q-learning with either exact or approximate Q-values depending upon the nature and size of the problem.

The algorithms for the Bayesian game approximation are shown in Algorithm 1 and 2. Algorithm 1 takes the parameters of the original POSG (the initial type profile space is generated using the initial distribution over S) and builds up a series of one-step policies, σ^t, as shown in Figure 3. In parallel, each agent will: solve the Bayesian game for timestep t to find σ^t; match its current history of experiences to a specific type θ_i^t in its type profile; match this type to an action as given by its policy σ_i^t; and then execute that action. Algorithm 2 provides an implementation of how to solve a Bayesian game given the dynamics of the overall POSG and a specific type profile space and distribution over that space. It also propagates the type profile space and its prior probability forward one timestep based on the solution to that Bayesian game.

Algorithm 2: $BayesianGame$

```
**Input**: $I, \Theta, A, p(\Theta), Z, S, T, R, O, randSeed$
**Output**: $\sigma, \Theta', p(\Theta')$
**begin**
    $setSeed(randSeed)$
    **for** $a \in A, \theta \in \Theta$ **do**
        $u(a, \theta) \leftarrow qmdpValue(a, beliefState(\theta))$
    $\sigma \leftarrow findPolicies(I, \Theta, A, p(\Theta), u)$
    $\Theta' \longleftarrow \emptyset$
    $\Theta'_i \longleftarrow \emptyset, \forall i \in I$
    **for** $\theta \in \Theta$,$z \in Z$, $a \in A$ **do**
        $\phi \leftarrow \theta \cup z \cup a$
        $p(\phi) \leftarrow p(z, a|\theta)p(\theta)$
        **if** $p(\phi) >$ *pruningThreshold* **then**
            $\theta' \leftarrow \phi$
            $p(\theta') \leftarrow p(\phi)$
            $\Theta' \leftarrow \Theta' \cup \theta'$
            $\Theta'_i \leftarrow \Theta_i \cup \theta'_i$,$\forall i \in I$
**end**
```

Algorithm 3: $findPolicies$

```
**Input**: $I, \Theta, A, p(\Theta), u$
**Output**: $\sigma_i, \forall i \in I$
**begin**
    **for** $j \leftarrow 0$ ***to*** $maxNumRestarts$ **do**
        $\pi_i \leftarrow random, \forall i \in I$
        **while** $!converged(\pi)$ **do**
            **for** $i \in I$ **do**
                $\pi_i \leftarrow argmax[\sum_{\theta \in \Theta} p(\theta) *$
                $u([\pi_i(\theta_i), \pi_{-i}(\theta_{-i})], \theta)]$
        **if** *bestSolution* **then**
            $\sigma_i \leftarrow \pi_i, \forall i \in I$
**end**
```

Additional variables used in these algorithms are: r is the reward to the team, rs is a random seed, π_i is a generic strategy, ϕ is a generic type, and superscripts refer to the timestep under consideration. The function $beliefState(\theta)$ returns the belief state over S given by the history of observations and actions made by all agents as defined by θ. In Algorithm 2 we specify a Q_{MDP} heuristic for calculating utility, but any other heuristic could be substituted.

For alternating-maximization, shown in its dynamic programming form in Algorithm 3, random restarts are used to move the solution out of local maxima. In order to ensure that each agent finds the same set of policies, we synchronize the random number generation of each agent.

5. Experimental Results

5.1. Lady and The Tiger

The *Lady and The Tiger* problem is a multi-agent version of the classic tiger problem [3] created by Nair et al. [8]. In this two state, finite horizon problem, two agents are faced with the problem of opening one of two doors, one of which has a tiger behind it and the other a treasure. Whenever a door is opened, the state is reset and the game continues un-

Time	Full POSG		Bayesian Game Approx.		*selfish*
Horizon	Time(ms)	Exp. Reward	Time(ms)	Avg. Reward	Avg. Reward
3	50	5.19	1	5.18 ± 0.15	5.14 ± 0.15
4	1000	4.80	5	4.77 ± 0.07	-28.74 ± 0.46
5	25000	7.02	20	7.10 ± 0.12	-6.59 ± 0.34
6	900000	10.38	50	10.28 ± 0.21	-42.59 ± 0.56
7	—	—	200	10.00 ± 0.17	-19.86 ± 0.46
8	—	—	700	12.25 ± 0.19	-56.93 ± 0.65
9	—	—	3500	11.86 ± 0.14	-34.71 ± 0.56
10	—	—	9000	15.07 ± 0.23	-70.85 ± 0.73

Table 1. Computational and performance results for *Lady and the Tiger*, $|S| = 2$. Rewards are averaged over 10000 trials with 95% confidence intervals shown.

til the fixed time has passed. The crux of the *Lady and the Tiger* is that neither agent can see the actions or observations made by their teammate, nor can they observe the reward signal and thereby deduce them. It is, however, necessary for an agent to reason about this information. See Nair et al. [8] for $S, A, Z, T, R,$ and O.

While small, this problem allows us to compare the policies achieved by building the full extensive form game version of the POSG to those from our Bayesian game approximation. It also shows how even a very small POSG quickly becomes intractable. Table 1 compares performance and computational time for this problem with various time horizons. The full POSG version of the problem was solved using the dynamic programming version of alternating-maximization with 20 random restarts (increasing the number of restarts did not result in higher valued policies).

For the Bayesian game approximation, each agent's type at time t consists of its entire observation and action history up to that point. Types with probability less than 0.000005 were pruned. If an agent's true history is pruned, then it is assigned the closest matched type in Θ_i^t for action selection (using Hamming distance). The heuristic used for the utility of actions was based upon the value of policies found for shorter time horizons using the Bayesian game approximation. Our algorithm was able to find comparable policies to the full POSG in a much shorter time.

Finally, to reinforce the importance of using game theory, results are included for a *selfish* policy in which agents solve parallel POMDP versions of the problem using Cassandra's publicly available *pomdp-solve*.[2] In these POMDPs, each agent assumes that its teammate never receives observations that would lead it to independently open a door.

5.2. Robotic Team Tag

This example is a two-robot version of Pineau et al.'s *Tag* problem [10]. In *Team Tag*, two robots are trying to herd and then capture an opponent that moves with a known stochastic policy in the discrete environment depicted in Figure 4. Once the opponent is forced into the goal cell, the two robots must coordinate on a tag action in order to receive a reward and end the problem.

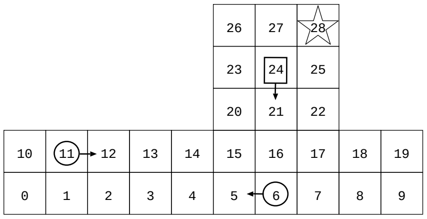

Figure 4. The discrete environment used for *Team Tag*. A star indicates the goal cell.

The state space for this problem is the cross-product of the two robot positions $robot_i = \{s_o, ..., s_{28}\}$ and the opponent position $opponent = \{s_o, ..., s_{28}, s_{tagged}\}$. All three agents start in independently selected random positions and the game is over when $opponent = s_{tagged}$. Each robot has five actions, $A_i = \{North, South, East, West, Tag\}$. A shared reward of -1 per robot is imposed for each motion action, while the Tag action results in a +10 reward if $robot_o = robot_1 = opponent = s_{28}$ and if both robots select Tag. Otherwise, the Tag action results in a reward of -10. The positions of the two robots are fully observable and motion actions have deterministic effects. The position of the opponent is completely unobservable unless the opponent is in the same cell as a robot and robots cannot share observations. The opponent selects actions with full knowledge of the positions of the robots and has a stochastic policy that is biased toward maximizing its distance to the closest robot.

A fully observable policy was calculated for a team that could always observe the opponent using dynamic programming for an infinite horizon version of the problem with a discount factor of 0.95. The Q-values generated for this policy were then used as a Q_{MDP} heuristic for calculating the utility of the Bayesian game approximation. Each robot's type is its observation history with types having probability less than 0.00025 pruned.

Two common heuristics for partially observable problems were also implemented. In these two heuristics, each robot maintains an individual belief state based only upon its own observations (as there is no communication) and uses that belief state to make action selections. In the Most Likely State (MLS) heuristic, the robot selects the most likely opponent state and then applies its half of the best joint action for that state as given by the fully observable policy. In the Q_{MDP} heuristic, the robot implements its half of the joint action that maximizes its expected reward given its current belief state. MLS performs better than Q_{MDP} as it does not exhibit an oscillation in action selection. As can be seen in Table 2, the Bayesian game approximation per-

[2] http://www.cs.brown.edu/research/ai/pomdp/code/index.html

Algorithm	Avg. Discounted Value	Avg. Timesteps	%Success
With Full Observability of Teammate's Position			
Fully Observable	-8.08 ± 0.11	14.39 ± 0.11	100%
Most Likely State	-23.44 ± 0.23	49.43 ± 0.65	78.10%
Q_{MDP}	-29.83 ± 0.23	63.82 ± 0.7	56.61%
Bayesian Approx.	-21.23 ± 0.76	43.84 ± 1.97	83.30%
Without Fully Observability of Teammate's Position			
Most Likely State	-29.10 ± 0.25	67.31 ± 0.73	46.12%
Q_{MDP}	-48.35 ± 1.18	89.02 ± 0.52	16.12%
Bayesian Approx.	-22.21 ± 0.80	43.53 ± 1.00	83.20%

Table 2. Results for *Team Tag*, $|S| = 25230$. Results are averaged over 1000 trials for the Bayesian approximation and 10000 trials for the others, with 95% confidence intervals shown. Success rate is the percentage of trials in which the opponent was captured within 100 timesteps.

forms better than both MLS and Q_{MDP}.

The *Team Tag* problem was then modified by removing full observability of both robots' positions. While each robot maintains full observability of its own position, it can only see its teammate if they are located in the same cell. The MLS and Q_{MDP} heuristics maintain an estimate of the teammate's position by assuming that the teammate completes the other half of what the robot determines to be the best joint action at each timestep (the only information to which it has access). This estimate can only be corrected if the teammate is observed. In contrast, the Bayesian game approximation maintains a type space for each robot that includes its position as well as its observation at each timestep. The probability of such a type can be updated both by using the policies computed by the algorithm and through a mutual observation (or lack thereof).

This modification really shows the strength of reasoning about the actions and experiences of others. With an opponent that tries to maximize its distance from the team, a good strategy is for the team to go down different branches of the main corridor and then herd the opponent toward the goal state. With MLS and Q_{MDP}, the robots frequently go down the same branch because they both incorrectly make the assumption that their teammate will go down the other side. With the Bayesian game approximation, however, the robots were able to better track the true position of their teammate. The performance of these approaches under this modification are shown in Table 2, and it can be seen that while the Bayesian approximation performs comparably to how it did with full observability of both robot positions, the other two algorithms do not.

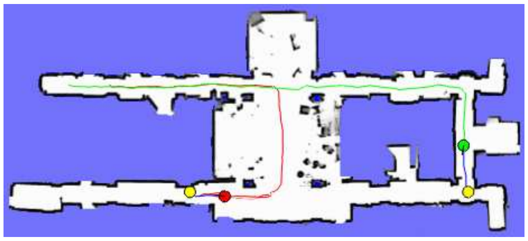

Figure 5. The robot team running in simulation with no opponent present. Robot paths and current goals (light coloured circles) are shown.

Algorithm	Avg. Discounted Value	Avg. Timesteps	%Success
Fully Observable	2.70 ± 0.09	5.38 ± 0.06	100%
Most Likely State	-7.04 ± 0.21	16.47 ± 0.28	99.96%
Q_{MDP}	-9.54 ± 0.27	25.29 ± 0.54	95.37%
Bayesian Approx.	-6.01 ± 0.21	15.20 ± 0.27	99.94%

Table 3. Results for *Tag 2*, $|S| = 18252$. Results are averaged over 10000 trials with 95% confidence intervals shown.

5.3. Robotic Tag 2

In order to test the Bayesian game approximation as a real-time controller for robots, we implemented a modified version of *Team Tag* in a portion of Stanford University's Gates Hall as shown in Figure 5. The parameters of this problem are similar to those of *Team Tag* with full teammate observability except that: the opponent moves with Brownian motion; the opponent can be captured in any state; and only one member of the team is required to tag the opponent. As with *Team Tag*, the utility for the problem was calculated by solving the underlying fully-observable game in a grid-world. MLS, Q_{MDP} and the Bayesian game approximation were then applied to this grid-world to generate the performance statistics shown in Table 3. Once again, the Bayesian approximation outperforms both MLS and Q_{MDP}.

The grid-world was also mapped to the real Gates Hall and two Pioneer-class robots using the Carmen software package for low-level control.[3] In this mapping, observations and actions remain discrete and there is an additional layer that converts from the continuous world to the grid-world and back again. Localized positions of the robots are used to calculate their specific grid-cell positions, with each cell being roughly 1.0m×3.5m. Robots navigate by converting the actions selected by the Bayesian game approximation into goal locations. For runs in the real environment, no opponent was used and the robots were always given a null observation. (As a result, the robots continued to hunt for an opponent until stopped by a human operator.) Figure 5 shows a set of paths taken by robots in a simulation of this environment as they attempt to capture and tag the (non-existent) opponent. Runs on the physical robots generated similar paths for the same initial conditions.

[3] http://www.cs.cmu.edu/˜carmen

6. Discussion

We have presented an algorithm that deals with the intractability of POSGs by transforming them into a series of smaller Bayesian games. Each of these games can then be solved to find one-step policies that, together, approximate the globally optimal solution of the original POSG. The Bayesian games are kept efficient through pruning of low-probability histories and heuristics to calculate the utility of actions. This results in policies for the POSG that are locally optimal with respect to the heuristic used.

There are several frameworks that have been proposed to generalize POMDPs to distributed, multi-agent systems. DEC-POMDP [1], a model of decentralized partially observable Markov decision processes, and MTDP [12], a Markov team decision problem, are both examples of these frameworks. They formalize the requirements of an optimal policy; however, as shown by Bernstein et al., solving decentralized POMDPs is NEXP-complete [1]. I-POMDP [5] also generalizes POMDPs but uses Bayesian games and decision theory to augment the state space to include models of other agents' behaviour. There is, however, little to suggest that these I-POMDPs can then be solved optimally. These results reinforce our belief that locally optimal policies that are computationally efficient to generate are essential to the success of applying the POSG framework to the real world.

Algorithms that attempt to find locally optimal solutions include POIPSG [9] which is a model of partially observable identical payoff stochastic games in which gradient descent search is used to find locally optimal policies from a limited set of policies. Rather than limit policy space, Xuan et. al. [16] deal with decentralized POMDPs by restricting the type of partial observability in their system. Each agent receives only local information about its position and so agents only ever have complementary observations. The issue in this system then becomes the determination of when global information is necessary to make progress toward the goal, rather than how to resolve conflicts in beliefs or to augment one's own belief about the global state.

The dynamic programming version of the alternating-maximization algorithm we use for finding solutions for extensive form games is very similar to the work done by Nair et. al. but with a difference in policy representation [8]. This alternating-maximization algorithm, however, is just one subroutine in our larger Bayesian game approximation; our approximation allows us to find solutions to much larger problems than any exact algorithm could handle.

The Bayesian game approximation for finding solutions to POSGs has two main areas for future work. First, because the performance of the approximation is limited by the quality of the heuristic used for the utility function, we plan to investigate heuristics, such as policy values of centralized POMDPs, that would allow our algorithm to consider the effects of uncertainty beyond the immediate action selection. The second area is in improving the efficiency of the type profile space coverage and representation.

Acknowledgments

Rosemary Emery-Montemerlo is supported in part by a Natural Sciences and Engineering Research Council of Canada postgraduate scholarship, and this research has been sponsored by DARPA's MICA program.

References

[1] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *UAI*, 2000.

[2] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge*, 1996.

[3] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, 1994.

[4] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[5] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. In *Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics*, Ft. Lauderdale, Florida, January 2004.

[6] D. Koller, N. Megiddo, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, June 1996.

[7] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, 1995.

[8] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, 2003.

[9] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *UAI*, 2000.

[10] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *UAI*, 2003.

[11] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, March 2003.

[12] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 2002.

[13] S. Russell and P. Norvig. Section 6.5: Games that include an element of chance. In *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[14] J. Shi and M. L. Littman. Abstraction methods for game theoretic poker. *Computers and Games*, pages 333–345, 2000.

[15] B. von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, June 1996.

[16] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Agents-01*, 2001.