Game-Theoretic Control for Robot Teams

Rosemary Emery-Montemerlo

12th August 2005

CMU-RI-TR-05-36

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Robotics.*

Thesis Committee:
Jeff Schneider (co-chair)
Sebastian Thrun (co-chair)
Geoff Gordon
Michael Littman, Rutgers, The State University of New Jersey

Keywords: Game Theory, Robot Teams, Partially Observable Stochastic Games, Decentralized Control, Planning and Execution

Abstract

Planning for a decentralized team of robots is a fundamentally different problem from that of centralized control. During decision making, robots must take into account not only their own observations of world state, but also the possible observations and actions of teammates. While the interconnectedness of such a reasoning process seems to require an infinite recursion of beliefs to be modelled by each member of the team, game theory provides an alternative approach. Partially observable stochastic games (POSGs) generalize notions of single-stage games and Markov decision processes to both multiple agents and partially observable worlds. Even if there is only limited communication between teammates, POSGs allow robots to come up with policies that still take into account possible teammate experiences without the need to explicitly model any recursive beliefs about those experiences.

While a powerful model of decentralized teams, POSGs are computationally intractable for all but the smallest problems. This dissertation proposes a Bayesian game approximation to POSGs in which game-theoretic reasoning about action selection is retained, but robots reason only a limited time ahead about uncertainty in world state and the experiences of their teammates. Planning and execution are interleaved to further reduce computational burdens: at each timestep, robots perform a step of full game-theoretic reasoning about their current action selection given any possible history of observations and a heuristic evaluation of the expected future value of those decisions.

The Bayesian game approximation algorithm (BaGA) is able to find solutions to much larger problems than previously solved. Further computational savings are gained by reasoning about groups of similar observation histories rather than single histories. Finally, efficiency and performance are also improved through the use of run-time communication policies that trade off expected gains in performance with the costs of using bandwidth. In this dissertation, the performance of BaGA is compared to policies generated for full POSGs as well as heuristics. BaGA is also used to develop real-time robot controllers for a series of simulated and physical robotic tag problems that gradually increase in realism.

Acknowledgments

I would like to thank my thesis advisors, Jeff Schneider and Sebastian Thrun, for giving me the chance to explore such a challenging topic and the guidance necessary to make headway. I would also like to thank Geoff Gordon for, amongst other things, so ably guiding me through the intricacies of game theory. I would like to thank my external committee member, Michael Littman, for his support and many helpful suggestions.

I am very grateful to the Natural Sciences and Engineering Research Council (and the taxpayers) of Canada who helped support my graduate research.

My life and studies at CMU have been enriched by so many wonderful people. Thanks to my first thesis advisor, Tucker Balch, who provided me with the invaluable RoboCup experiences that inspired this dissertation research. Special thanks to Suzanne Lyons Muth who, in addition to all that she does on a daily basis for RoboGrads, helped ease my transition between CMU and Stanford and for which I am extremely grateful. Thanks also to my fellow RoboGrads (and their spouses!): Nick Roy, Joelle Pineau, Aaron Courville, Drew Bagnell, Gita Sukthankar, Vandi Verma, Chris Urmson, Carl Wellington and Chris Dima, for many helpful comments and suggestions over the years as well as such wonderful dinners. I would also like thank Matt Rosencrantz and Curt Bererton for their help with robot wrangling. Special thanks go to Sarah Mihailovich who helped me not only get through undergrad, but also grad school.

Finally, I would like to thank my family, and my husband Michael, for their love, support and encouragement. Mike, despite claiming not to understand MDPs, you have been so helpful and have improved both my life and my sentences. As you yourself first put it so eloquently, I could not have done this without you.

For my wonderful husband.

Contents

List of Figures

List of Tables

List of Algorithms

Table of Notation

(I, S, A, Z, T, R, O)	a POSG
$I = \{1, 2, \ldots, n\}$	set of n robots/agents/players
A_i	set of actions for robot i
Z_i	set of observations for robot i
$A = A_1 \times \ldots \times A_n$	set of joint actions
S	set of states
$Z = Z_1 \times \ldots \times Z_n$	set of joint observations
$T : S \times A \rightarrow S$	the transition function
$R : S \times A \rightarrow \Re$	the reward function
$O : S \times A \times Z \rightarrow [0, 1]$	the observation emission probabilities
A_*	the largest action set for any robot i, $\|A_*\| \geq \|A_i\|, \forall i$
Z_*	the largest observation set for any robot i, $\|Z_*\| \geq \|Z_i\|, \forall i$
h_i	history of robot i, will include all observations and, sometimes, actions taken by i, h_i^t indicates a history covering t timesteps
θ_i	a type of robot i, for example a possible h_i
Θ_i	type space for robot i such that $\theta_i \in \Theta_i$
$\theta = \{\theta_1, \ldots, \theta_n\}$	a joint type
$\Theta = \{\Theta_1 \times \ldots \times \Theta_n\}$	the joint-type space
θ_{-i}	the type of all robots but i
$P \in \Delta(\Theta)$	probability distribution over the joint-type space Θ
$P_i \in \Delta(\Theta_i)$	marginal distribution over robot i's type
$P_i(\theta_{-i} \| \theta_i)$	conditional probability of θ_{-i} given θ_i
$u_i(a_i, a_{-i}, (\theta_i, \theta_{-i}))$	utility of robot i taking action a_i when it has type θ_i given the actions and types of other robots
$u = \{u_1, \ldots, u_n\}$	set of utility functions

π_i	policy or strategy for robot i
$P(a_i\|\theta_i)$= $\pi_i(\theta_i)$	probability distribution over actions assigned for type θ_i by robot i's policy
$(I, \Theta, A, P(\Theta), u)$	a Bayesian Game
π	a set of policies or strategies for a set of I robots, the solution to a Bayesian Game
θ_i^t	type of robot i at timestep t, similarly the action, observation, history, joint type, joint-type space and policies at timestep t can be defined
r^{θ_i}	reward profile for θ_i
$r_a^{\theta_i} = E(u(a, \theta)\|\theta_i)$	a-th element of the reward profile, it is the expected utility to i for doing action a given its belief about the state of the world induced by θ_i and the utility function u
c_i	a cluster of types
r^{c_i}	reward profile for cluster c_i
C_i^t	the set of all possible clusters for robot i at timestep t
BG^t	a Bayesian game approximating timestep t of a POSG
BGC_i^t	a Bayesian game approximating timestep t of a POSG, in which all robots know robot i's type with certainty
$\sigma^{i,t}$	the solution to BGC_i^t
BG'^t	a Bayesian game approximating timestep t in which the types of some robots are known with certainty and others are not based on the results of communication

Chapter 1

Introduction

In the field of multi-robot systems, researchers seek to understand and specify the interaction between robots as they cooperate to perform tasks in the real world. It is a field that has attracted researchers from a variety of backgrounds, each bringing a different flavour to their approach. Behaviour-based researchers use the notion of social behaviours to facilitate emergent cooperation in reactive systems (Matarić, 1995; Parker, 1998), while machine-learning researchers have examined issues of multi-agent adaption and learning (Weiß & Sen, 1996; Weiß, 1997; Stone & Veloso, 2000). Others have looked at the problem from the perspective of teamwork: the study of social intentions, commitments, beliefs and desires (Cohen & Levesque, 1991; Grosz & Kraus, 1996). Market-based approaches treat task allocation as an auction in which robots bid for tasks in an effort to maximize their individual rewards (Wellman & Wurman, 1998; Gerkey & Matarić, 2002; Dias & Stentz, 2003).

While it is a powerful multi-agent framework, game theory is rarely used to model multi-robot problems, in part due to its high computational complexity. In the absence of game-theoretic reasoning, however, each decision maker in a decentralized multi-robot team is faced with an infinite belief/knowledge hierarchy. In the case of two robots, Robot 1's actions are dependent on what it has seen, what it believes Robot 2 has seen, what it believes about Robot 2's beliefs about what Robot 1 has seen, and so on. These belief hierarchies arise from both uncertainty in the state of the world as well as in the action choices of teammates. By reasoning directly about all possible sequences of actions and observations across robot teams, game theory eliminates the need for robots to explicitly model the belief hierarchies of their teammates in partially observable domains.

In general, the full game-theoretic approach to the multi-agent problem is computationally infeasible. However, by creating approximate solutions to the games representing these problems, this approach can be used to create real-time robot controllers for reasonably sized domains. Additionally, by understanding how existing algorithms relate to the full game-theoretic approach, insight is gained into the tradeoff between optimality and performance in real multi-robot problems.

1.1 Problem Description

This dissertation is concerned with the problem of constructing policies for robots that must coordinate their action selection in tightly-coupled tasks within a partially observable Markovian environment with limited communication.

1.1.1 Tightly-Coupled Tasks

Distributed robot problems can range from loosely-coupled to tightly-coupled tasks. At one end of the spectrum, once each robot is assigned a task, no interaction is necessary between teammates during task completion. An example of this class would be robotic farming in which each tractor is assigned a different field to plow. This type of problem does not really require multi-robot algorithms because, once the initial task decomposition is set, the robots function as individual entities.

As task allocation takes on a more dynamic role, the level of interaction between robots starts to increase. One can imagine that, should a robotic tractor fail, its remaining fields must be reallocated amongst the remaining team. Or perhaps an ordering exists between sub-tasks, such as a rock collecting robot that cannot complete its task until one of its teammates has discovered the location of the rock. Robots must now negotiate between each other to insure that optimal task allocation and reallocation takes place.

If robots are located physically close in space, another type of interaction starts to occur; one at a lower level of behaviour. Robots must now re-plan the paths they take while performing a task in order to avoid bumping into their teammates.

At the tightly-coupled end of the spectrum, robots must coordinate their domain-level actions in order to achieve optimal performance. They start to function less like individuals with mutual goals and more like a single entity with a distributed controller. In multi-robot box pushing, the team must coordinate on actuator placement and force to ensure that the box takes the correct path (Gerkey & Matarić, 2002). In a sentry problem (e.g., guarding an art gallery or museum), robots must ensure that their joint paths through the galleries leave no area unmonitored.

It is this latter end of the spectrum with which this thesis is concerned. This dissertation is focused on distributed robots that, in parallel, select individual actions at every timestep of a problem. The problems consist of an overall task or goal with an associated reward function that reflects the need of individual robots to coordinate during action selection.[1]

1.1.2 Stochastic Environment

In the real world, robots must constantly deal with uncertainty. Part of this uncertainty comes from robot actuators and the effects they have on actions. Generally, these effects can be captured through some sort of probabilistic model governing the transitions between states in the environment. Furthermore, for many problems the Markov property holds. The Markov property states that if the current world state is known, then the future state is not dependent on any of the previous states. That is, all information necessary to determine a distribution over the next set of world states is contained entirely within the current state description. This means that, given a history of past states s^i and actions a^i, $P(s^{t+1}|s^t, a^t, s^{t-1}, a^{t-1}, \ldots, s^0, a^0) = P(s^{t+1}|s^t, a^t)$.

This dissertation is concerned with problems that obey the Markov assumption and for which probabilistic world models can be derived. It will be shown, however, that existing single-robot methods for these problems, and their extensions to handle partial observability, are not sufficient for modelling decision making in a team of robots with limited communication.

1.1.3 Partial Observability

The second type of uncertainty comes from how robots sense their environments. Sensors are not perfect and while impressive algorithms for localization exist, there is always some uncertainty inherent in determining position (Fox et al., 1999). Even with extremely accurate sensors, components of world state can remain unobservable due to perceptual

[1] The methods for generating policies that will be explored in this dissertation can still be applied to task-level coordination; one can replace actions with specific tasks, modify reward functions appropriately and then use the framework to assign tasks at each timestep. Robots can then use precalculated policies for implementing those tasks much as they would with frameworks such as market-based approaches (Dias & Stentz, 2003) and theories of teamwork (Jennings, 1995; Tambe, 1997).

aliasing of sensed features or sensor limitations. For example, a robot cannot observe what is around a corner until it actually moves around the corner itself.

In partially observable domains, if some aspect of state is not directly observable then it is considered hidden state. While a robot cannot directly identify this hidden state, its observations will contain information that allow it to infer a set of possible world states and to generate a probability distribution over these possible states. This probability distribution, also known as a belief, is updated by the robot based upon its actions and observations as it moves through the environment. Robots can therefore use their beliefs to guide action selection, much as they would use world state in a fully observable problem.

In robot teams there are two main categories of partial observability: collective observability and collective partial observability (Pynadath & Tambe, 2002).[2] With collective observability, if robots can share all of their observations then world state is known with certainty; however, with collective partial observability no assumptions are made about the joint observations of the team. Unless all observations can be shared, it is not necessarily easier to find solutions to collectively observable domains.

If full communication is available within a team of robots then any observations of world state can be shared between the team. Limited communication adds another type of partial observability. If robots are not able to share their observations every timestep they can only infer a distribution over the possible experiences of their teammates. As a result, the true observation history of each teammate can be considered another part of the problem's hidden state.

The problems addressed in this dissertation fall into the category of collective partial observability. More specifically, these problems will all include partial observability as a result of limited communication.

1.1.4 Limited Communication

In general, communication bandwidth is limited and in some domains not available at all. For example, robots that function underground or on the surface of another planet may be limited to point-to-point communication. In these problems, robots must evaluate the benefit of communication against the associated costs of moving into a position suitable for initiating communication. Even if communication is more readily available, such as

[2] One could also include the case of complete, or individual, observability in this set. If a problem has complete observability, then each robot knows the current world state with certainty.

with indoor mobile robots, bandwidth limitations often result in latency in communication between the robots, and critical information may not be received in time to make decisions. While bandwidth requirements for full communication may be feasible for small robot teams, scaling up the number of robots in such a system will eventually saturate the communication network.

Instead of sharing complete histories of their actions and observations, robots should be able to independently reason about appropriate actions to take in a decentralized fashion that takes into account both their individual experiences in the world and beliefs about the possible experiences of teammates. If no communication is available, then this type of reasoning is necessary for the entire duration of the problem. If communication is available, but is periodic or has latency, this reasoning allows robots to generate optimal policies between synchronization episodes (Nair et al., 2004).

This dissertation is concerned with problems that have limited or no communication available to the robot team. Robots must be able to arrive at policies that are optimal given the information they have available about the experiences of their teammates and the environment. If communication is available, robots should be able to generate communication policies that trade off the benefits of additional information with the costs of communication.

1.2 Probabilistic Frameworks

This section discusses how the probabilistic frameworks of Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs) can be used to solve decentralized robot problems if certain assumptions are made about the world. As the framework moves from assumptions of full observability to partial observability, more realistic problems can be represented; however, even POMDPs are still insufficient for modelling the original problem.

1.2.1 MDP Solution

One of the most restrictive assumptions typically applied to distributed robot control is that the world is fully observable. This assumption means that, at each timestep, every member of the team knows the current world state with certainty. This problem can be modeled as an MDP.

An MDP is a tuple (S, A, T, R) where S is a set of possible world states, A is a set of actions, $T : S \times A \rightarrow S$ is a probabilistic transition function that describes how state evolves as a function of actions and $R : S \times A \rightarrow \Re$ is the reward function. MDPs can be defined with a finite planning horizon in which an agent has a fixed number of timesteps in which to maximize its reward. Alternatively, they can be formulated as infinite-horizon problems in which the agent continues to act until some terminal state is reached.

Using the MDP model, a policy is found for the agent using dynamic programming. This policy is a mapping between the states in S and the actions in A (as shown in Figure 1.1 (a)) that attempts to maximize cumulative reward. For a finite horizon problem, the agent takes the action at every timestep that maximizes its current reward plus the sum of discounted expected future rewards over the remaining timesteps. If $\pi^t(s^t)$ is the action given by policy π for state s at timestep t, then the value of starting in state s and executing π for the remaining $T - t$ steps is given by the finite-horizon Bellman's Equation:

$$V_\pi^t(s^t) = R(s^t, \pi^t(s^t)) + \gamma \sum_{s^{t+1}} T(s^t, \pi^t(s^t), s^{t+1}) V_\pi^{t+1}(s^{t+1}) \tag{1.1}$$

where γ is a discount factor. The optimal policy, π_*, and its value function, V_*, can be found by maximizing Equation 1.1 and applying backwards induction from $t = T$ to $t = 0$. For each state s:

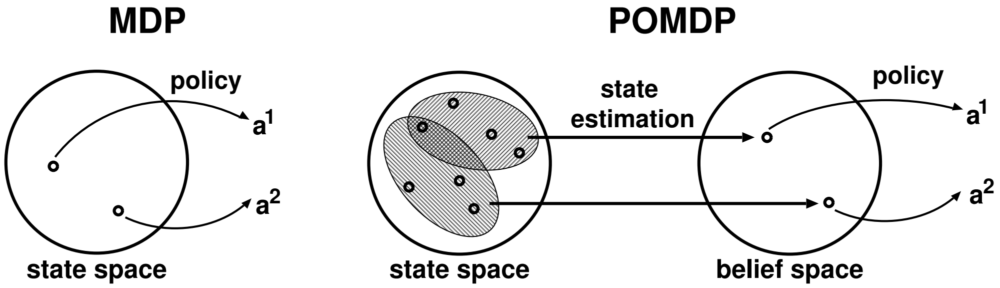
$$V_*^t(s^t) = \max_a (R(s^t, a) + \gamma \sum_{s^{t+1}} T(s^t, a, s^{t+1}) V_*^{t+1}(s^{t+1})) \tag{1.2}$$

and the optimal policy π_* is given by:

$$\pi_*^t(s^t) = \arg\max_a (R(s^t, a) + \gamma \sum_{s^{t+1}} T(s^t, a, s^{t+1}) V_*^{t+1}(s^{t+1})). \tag{1.3}$$

For an infinite horizon problem, Equations 1.1, 1.2 and 1.3 can be modified to reflect that an agent takes the action that maximizes its current reward plus its discounted expected future reward. Rather than finding a policy dependent on time, the agent is now trying to find the stationary value function V^* that maximizes Bellman's equation:

$$V^*(s) = \max_a (R(s, a) + \gamma \sum_{s'} T(s, a, s') V^*(s')). \tag{1.4}$$



(a) MDP relationship between state space and actions.

(b) POMDP relationship between state space, belief space and actions.

Figure 1.1: Relationship between MDPs and POMDPs. In an MDP, a policy is a direct mapping between states and actions. In POMDPs, an agent no longer knows the current state with certainty but instead constructs a probability distribution over all possible states. A POMDP policy can then be thought of as a direct mapping between these probability distributions (a belief) and actions. That is, a POMDP can be thought of as an MDP in belief space rather than state space.

Dynamic programming can be used to find the fixed point, V^*, of Equation 1.4 and its associated optimal policy π^* (Bertsekas & Tsitsiklis, 1996).

To model a multi-robot problem using the MDP framework, the set of actions A becomes the set of joint actions $A = A_1 \times \cdots \times A_n$; that is, the robots are treated as a larger meta-agent with multiple actuators. For each state in S, the meta-agent selects an action from the set A according to its policy and then 'tells' each of its actuators which individual-level action to implement. In practice, it is not necessary for a central computer to make these decisions. Because the problem is fully observable, each robot can store a copy of the computed policy and implement the appropriate portion of the selected joint actions at run time (Boutilier, 1999).

1.2.2 POMDP Solution

In this section, the problem is made more realistic by removing the assumption that the state of the world is fully observable by each robot and, instead, robots only receive noisy observations about the world. Although the problem is no longer fully observable, it is assumed that full communication is available to the team; that is, any observation made by one robot is also known by the other robots, without latency or further signal degradation.

This assumption of full communication allows the problem to be treated as a POMDP.

Formally, a POMDP is a tuple (S, A, Z, T, R, O) where S, A, T and R are as defined for an MDP, Z is a set of observations and $O : S \times A \times Z \rightarrow [0, 1]$ is the observation emission probability function. Given an initial belief about the world state b^0, the current belief state b (the probability that an agent is in state s) can be calculated from the history of actions and observations taken by each agent: $b^t = P(s^t|z^t, a^t, z^{t-1}, a^{t-1}, \ldots, z^0, a^0)$. Updating the current belief b is known as state tracking or filtering, and the new belief can be constructed from the transition and emission functions as follows:

$$b'(s') \quad = P(s'|z, a, b) \quad = \frac{O(s', a, z) \sum_{s'} T(s, a, s') b'(s')}{P(z|b, a)} \tag{1.5}$$

where $b(s) = P(s|b)$ and $P(z|b, a)$ is a normalization factor equivalent to $\sum_{s'} \big(O(s', a, z) \sum_s T(s, a, s') b(s)\big)$.

Like MDP policies, POMDP policies are universal plans: they specify an action to take for any possible belief state. The addition of partial observability, however, makes it significantly harder to find policies for POMDPs than it is for MDPs. Every POMDP can be thought of as an equivalent belief space MDP whose state set is the set of reachable beliefs of the POMDP (Kaelbling et al., 1998). Rather than plan in state space, an agent plans in belief space as is shown in Figure 1.1 (b). The value function and optimal policy equations in Section 1.2.1 can be re-written to take this dependence into account. Given a belief b^t:

$$V_*^t(b^t) = \max_a \big(\sum_{s^t} b^t(s^t) R(s^t, a) + \gamma \sum_{b^{t+1}} P(b^{t+1}|b^t, a) V_*^{t+1}(b^{t+1})\big) \tag{1.6}$$

where the transition and observation emission functions are used to calculate $P(b^{t+1}|b^t, a)$.

While exact planning algorithms exist for POMDPs, (e.g., the Witness algorithm of Kaelbling et al. (1998)), finite-horizon POMDPs themselves are PSPACE-complete (Papadimitriou & Tsitsiklis, 1987). As a result, a number of approximate algorithms for solving POMDPs have been developed such as those of Hauskrecht (2000) and Pineau et al. (2003a).

For the multi-robot version of this problem, as with the MDP solution, a policy for a larger meta-agent with multiple actuators can be found. In this solution, the action set A consists of all joint actions $A_1 \times \cdots \times A_n$, and Z consists of all joint observations $Z_1 \times \cdots \times Z_n$. An appropriate solution technique can then be applied (e.g., an exact or approximate method depending on the size of the problem). At run time, each robot receives an observation z_i^t, transmits that signal to its teammates, updates its belief state b_i^t with all observations $z_j^t \in z^t$ and the previous joint action, and then looks up the optimal joint action for the new belief. Finally, it implements its part of that joint action. Because the robots share all of their observations at every timestep, the robots will always select the same joint action a^t (with ties broken consistently).

Collectively fully observable POMDPs represent a special case of multi-robot POMDPs. For these problems the global state of the system can be uniquely identified from each set of observations $\{z_1^t, \ldots, z_n^t\}$. Full communication allows problems that are collectively fully observable to be modeled as an MDP rather than a POMDP.

If the assumption of full communication is removed, however, then the problem suddenly becomes much harder, even if it is collectively fully observable. If no communication is available, each robot must make action decisions based only on its own history of observations $\{z_i^0, \ldots, z_i^t\}$, actions $\{a_i^0, \ldots, a_i^t\}$, and known problem dynamics (i.e., T, R and O). If it knows its teammates' policies, then it can fold that information into the transition and reward functions and model the problem as a single-robot POMDP. However, because robots are finding their policies simultaneously and those policies are interdependent, if each robot optimizes its own policy while holding those of its teammates fixed, it cannot be guaranteed that the resulting policies are consistent with each other. I.e., if robot i assumes a set of fixed policies σ for its teammates and then finds an 'optimal' policy π_i for itself, it cannot be guaranteed that this π_i is identical to the policy that its teammates assign to it during their own optimization process. As a result, robot i's teammates can find policies for themselves that are different from those in σ and so the assumptions that robot i made about σ are violated. Instead, when choosing actions, each robot must reason about how the actions its teammates choose would affect its own action selection and, in turn, how they would effect the teammates' action selection.

This type of reasoning requires a recursive modelling approach in which each robot builds up an infinite belief hierarchy about what it believes is the state of the world, what it believes its teammates believe is the state of the world, what it believes about what its teammates believe it believes is the state of the world, and so on. Because communication is not available to establish observations as common knowledge, this hierarchy is infinite even if the POMDP itself is finite. A single POMDP is no longer a rich enough framework to properly model what is occurring and so, game-theoretic models must be considered in order to break the infinite belief hierarchy.

1.3 Partially Observable Stochastic Games

Stochastic games, first introduced by Shapley (1953), are a generalization of both MDPs to the multi-robot case and single-stage games to games with state (Fudenberg & Tirole, 1991). Instead of modelling a set of agents as a single agent that executes joint actions, in a stochastic game each agent is modeled as a self-interested player who chooses actions that maximize its reward with respect to a reward function. Both reward and the transition between world states, however, are dependent on the actions selected by all agents. Game theory concepts like Nash equilibria are therefore used to simultaneously define sets of policies for all of the agents. Just as POMDPs are the extension of MDPs to partially observable domains, partially observable stochastic games (POSGs) are the extension of stochastic games to the case of uncertainty in world state.

A POSG is defined as a tuple (I, S, A, Z, T, R, O). $I = \{1, \ldots, n\}$ is the set of agents, and S is the set of states. It is important to note that S is not limited to the cross-product of the states of the individual agents, but can include additional information. For example, in a multi-robot mapping task the state of the world will include the positions of all of the robots as well as positions of any obstacle in the environment. A and Z are respectively the cross-product of the action and observation space for each agent, i.e., $A = A_1 \times \cdots \times A_n$ and $Z = Z_1 \times \cdots \times Z_n$. 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. These actions are given by the solution to the POSG, which is a set of conditional policies, $\pi = \{\pi_1, \ldots, \pi_n\}$. This thesis addresses finite POSGs with common payoffs, i.e., $R_i = R_j \forall i, j$; however, the set of general POSG problems includes arbitrary reward functions and infinite horizon problems.[3]

In the POSG model, a policy for each robot in a team can be found without the need to do any sort of infinite reasoning or deduction over an infinite belief hierarchy.[4] A POSG policy, π_i, is a universal, or conditional, plan that tells a robot what to do at each timestep as it

[3] While this formalism may appear similar to that of multi-agent POMDPs, a key difference is that in multi-agent POMDPs each agent is treated as an actuator of a larger meta-agent. This means that the observation z^t is known by all agents and a policy is a mapping from a joint belief to a joint action. In a POSG, however, each agent only has access to z_i^t. As a result, different solution methods are necessary and the policy of an agent is now a mapping from a history of individual observations to an individual action.

[4] The proof that the POSG framework is able to implicitly represent any possible infinite belief hierarchy is quite complex and beyond the scope of the game-theoretic concepts discussed in this thesis. Interested readers are directed to Harsanyi (1968) and Samet (1997) for an in-depth discussion of the relationship between belief/knowledge hierarchies and common priors. Informally, if there is a common prior held by all agents over the possible set of states of the game at any timestep (i.e., common knowledge as to the POSG model

acquires additional observations. At each timestep t, a robot has a sequence of observations h_i^t and will execute the action given by $\pi_i(h_i^t)$, i.e., $\pi_i : H_i^t \rightarrow A_i$. Alternatively, π_i can be thought of as a policy tree as shown in Figure 1.2: the nodes of the tree indicate actions while branches represent different observations. The robot starts at the root of the tree and, as it gathers more observations, follows the associated path down through the tree, executing the action given by each node it passes through. Note that in either case, while the policy is conditioned on only a history of observations h_i^t, due to the nature of policy construction, it also implicitly includes information about past action choices. This is because the policy is a deterministic function of history and so, given a history h_i^t, the actions taken at timesteps 0 through $t-1$ can be recovered with certainty.[5]

In order to coordinate their actions, a team of robots must find policies that take into account possible teammate observations and actions. However, the need for reasoning about an infinite belief hierarchy is avoided in POSGs by finding policies for all robots simultaneously. Given a specific set of policies π, it is possible to evaluate the resulting expected reward to the team. As shown by Nair et al. (2003), for a 2-robot general-sum problem, the value to robot i of starting in state s with the observation histories h_1^t and h_2^t, given that both robots will execute π for the remaining $T-t$ steps, is:

$$\begin{aligned} V_{\pi,i}^t(s^t, \{h_1^t, h_2^t\}) = R_i(s^t, \{\pi_1(h_1^t), \pi_2(h_2^t)\}) + \gamma \sum_{s^{t+1}} T(s^t, \{\pi_1(h_1^t), \pi_2(h_2^t)\}, s^{t+1}) \cdot \\ \big[\sum_{z_1 \in Z_1} \sum_{z_2 \in Z_2} O(s^{t+1}, \{\pi_1(h_1^t), \pi_2(h_2^t)\}, \{z_1, z_2\}) V_{\pi,i}^{t+1}(s^{t+1}, \{h_1^t \cup z_1, h_2^t \cup z_2\}) \big]. \quad (1.7) \end{aligned}$$

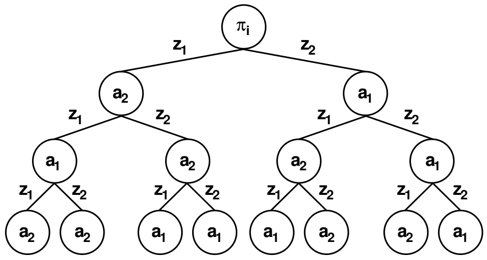
The subscript i is used to indicate that this value function is defined for robot i. As with Equation 1.1, dynamic programming can be used to find $V_{\pi,i}^t$ for all t, s, h_1^t and h_2^t. Note that this process requires iterating over all possible one-step extensions of h_1^t and h_2^t, yielding a computational cost of $O(|S||Z_1||Z_2|)$ for evaluating each $V_{\pi,i}^t(s^t, \{h_1^t, h_2^t\})$ pair and a cost of $O(|S|^2|Z_1|^{t+1}|Z_2|^{t+1})$ for finding $V_{\pi,i}^t$. Equation 1.7 can of course be generalized to the n-robot case.

and policies of each agent), then this common prior can be decomposed into an infinite hierarchy by using it to calculate the posteriors over all the possible beliefs in the hierarchy. However, in order to prove that a common prior is able to capture any infinite hierarchy of beliefs, the reverse must also be true. It turns out that so long as an infinite hierarchy of beliefs is consistent with itself (i.e., does not include beliefs that are contradictory), then it can be represented exactly as a common prior.

[5] For a POSG with common payoffs, it is only necessary to reason about deterministic policies due to the cooperative nature of the team.

$$
\begin{array}{ll}
\pi_i(h_i^1 = \{z_1\}) = a_2 & \pi_i(h_i^3 = \{z_1, z_1, z_1\}) = a_2 \\
\pi_i(h_i^1 = \{z_2\}) = a_1 & \pi_i(h_i^3 = \{z_1, z_1, z_2\}) = a_2 \\
 & \pi_i(h_i^3 = \{z_1, z_2, z_1\}) = a_1 \\
\pi_i(h_i^1 = \{z_1, z_1\}) = a_1 & \pi_i(h_i^3 = \{z_1, z_2, z_2\}) = a_1 \\
\pi_i(h_i^2 = \{z_1, z_2\}) = a_2 & \pi_i(h_i^3 = \{z_2, z_1, z_1\}) = a_1 \\
\pi_i(h_i^2 = \{z_2, z_1\}) = a_2 & \pi_i(h_i^3 = \{z_2, z_1, z_2\}) = a_2 \\
\pi_i(h_i^2 = \{z_2, z_2\}) = a_1 & \pi_i(h_i^3 = \{z_2, z_2, z_1\}) = a_2 \\
 & \pi_i(h_i^3 = \{z_2, z_2, z_2\}) = a_1
\end{array}
$$

(a) A policy π_i for a 3-step problem.



(b) π_i as a policy tree.

Figure 1.2: An example of a policy for robot i in a simple 3-step problem. Robot i has two possible observations: z_1 and z_2, and two possible actions: a_1 and a_2. In (a), π_i is enumerated as a function that returns an action given a history h_i^t of observations of length t. In (b), π_i is shown as a policy tree. Robot i starts at the root of the policy tree and, as it gathers more observations on each timestep, moves through the tree, executing the appropriate action for each of the nodes it reaches.

Given Equation 1.7, an exhaustive search through the space of all possible joint policies will guarantee finding the optimal joint policy π_*. In a common-payoff game, $V^t_{\pi,i}(s^t, \{h_1^t, h_2^t\}) = V^t_{\pi,j}(s^t, \{h_1^t, h_2^t\})\ \forall i, j$, and so the optimal policy π_* will simply be the policy $\pi_* = \{\pi_1, \pi_2\}$ that maximizes the value function for each $(s^t, \{h_1^t, h_2^t\})$ pair.[6] However, as each robot has $|A_i|^{\frac{|Z_i|^{T+1}-1}{|Z_i|-1}-1}$ possible individual policies, this search is doubly exponential in the planning horizon of the problem.

While it is possible to solve a POSG by performing exhaustive search over all possible joint policies, it seems less elegant than how policies are found in MDPs and POMDPs. In MDPs, a policy is conditioned on the state s and so it is possible to perform dynamic programming directly on Equation 1.2 to find a mapping between states and actions. In POMDPs, an agent does not have full access to the current world state and so a policy that is conditioned on state alone is not useful. Instead, the agent has a history of observations and actions that defines a belief over the world state b. If the policy is conditioned on this belief, then the POMDP must be converted into a belief space MDP before dynamic programming can be performed to find this policy.[7] With a POSG, however, the policy for each agent is conditioned on its history h_i^t. By itself, h_i^t does not define a unique

[6] In general, game-theoretic principles such as Nash equilibria, can be used to suggest policies for each robot given a set of $V^t_{\pi,i}(s^t, h^t)$ values. Game theory and Nash equilibria are presented in more detail in Chapter 2.

[7] While a POMDP policy can be conditioned on a history of observations (and actions), this representation is unnecessary as the belief state is a sufficient statistic of this information.

joint belief over the world state because such a belief is dependent on the observations and actions of all agents. A joint belief b can only be defined by $h^t = \{h_1^t, \ldots, h_n^t\}$. At best, therefore, h_i^t defines a probability distribution, $d_{h_i^t}$, over all possible joint beliefs in which $d_{h_i^t}(b) = P(b|h_i^t)$. Following the analogy of MDPs and POMDPs, it would seem possible to rewrite the POSG as a d-state MDP in which agents plan using a state set equivalent to all possible distributions over joint beliefs. The corresponding optimal policy will be one in which, for each distribution d^t, the agents take the joint action that maximizes their immediate expected reward and the sum of future expected reward over the remaining timesteps. That is, the optimal policy satisfies:

$$\sigma_*^t(d^t) = \arg\max_a \big(\sum_{b^t} d^t(b^t)\big[\sum_{s^t} b^t(s^t)R(s^t,a)\big] + \gamma \sum_{d^{t+1}} P(d^{t+1}|d^t,a)V_*^{t+1}(d^{t+1})\big). \tag{1.8}$$

Unlike π_*^t, σ_*^t is a mapping from a probability distribution d^t to a joint action $a^t = \{a_1, \ldots, a_n\}$. At run time, a robot would construct d_i^t , look up the associated a^t using σ_*^t and implement its part of that joint action. The problem now becomes, how to construct d_i^t. As is shown in Figure 1.3, choosing d_i^t conditioned on h_i^t results in miscoordination between robots even if they are in agreement as to σ_*^t. This miscoordination happens because, as each robot has a different history h_i^t, it will generate a different d_i^t and therefore select a different joint action.

Therefore, the use of Equation 1.8 is only valid if each robot uses the same distribution d^t at run time. Coordination can be insured by allowing robots to only use common information (such as the problem dynamics given by T and O) to generate d^t. However, the robots would then come up with a policy σ_*^t that does not make use of any observations received by team members during decision making. While such a policy is technically coordinated, for more powerful policies agents must be allowed to integrate their own information into the action selection process. However, modifying Equation 1.8 to condition policies on h_i^t and not d^t, effectively yields an exhaustive search on Equation 1.7:

$$\pi_*^t(h^t) \;=\; \begin{cases} \pi_{*,1}^t(h_1^t) = \arg\max_{\pi_1^t} \big(\sum_{k^t \in H^t} P(k^t|h_1^t) \big(\sum_{s^t} P(s^t|k^t) R(s, \pi^t(k^t)) + \\ \qquad \gamma \sum_{k^{t+1} \in H^{t+1}} P(k^{t+1}|k^t) V_{\pi_*^{t+1}}^{t+1}(k^{t+1}) \big) \big) \\ \vdots \\ \pi_{*,n}^t(h_n^t) = \arg\max_{\pi_n^t} \big(\sum_{k^t \in H^t} P(k^t|h_n^t) \big(\sum_{s^t} P(s^t|k^t) R(s, \pi^t(k^t)) + \\ \qquad \gamma \sum_{k^{t+1} \in H^{t+1}} P(k^{t+1}|k^t) V_{\pi_*^{t+1}}^{t+1}(k^{t+1}) \big) \big) \end{cases} \tag{1.9}$$

where

$$V_\pi^t(h^t) = \sum_{s^t} P(s^t|h^t) V_\pi^t(s^t, h^t). \tag{1.10}$$

There are several things to note about Equations 1.9 and 1.10: $V_\pi^t(s, h^t)$ is the generalized n-robot version of Equation 1.7; $V_{\pi_*^{t+1}}^{t+1}(k^{t+1})$ is the value of following policy π_* for steps $t+1$ through T, given a joint history k^{t+1}; $P(s^t|h^t) = b_{h^t}(s)$ where b_{h^t} is the joint belief state associated with h^t; and $P(k^t|h_i^t)$ is the probability of a joint history k^t occurring given that h_i^t occurs. Obviously, if h_i^t is not the i-th part of k^t, this probability will be zero, otherwise it will be dependent on problem dynamics. The iteration over every possible joint history $k^t \in H^t$ is a necessary part of Equation 1.9: while the goal is to find the optimal policy for a specific joint history h^t, each robot's component of that history, h_i^t, can be part of other joint histories and the policy π_i^t can only define one action to take for each h_i^t.

Finally, because π^t and not just π_i^t appears in the optimizations, the $\arg\max$ operator must be performed simultaneously.[8] In other words, while its form might be more analogous to Equation 1.3, this equation is merely 'hiding' the exhaustive search required by Equation 1.7. At each timestep the number of items in the set to be maximized over for each robot is equal to the number of different policies it can have at timestep t which is $O(|A_i|^{|Z_i|^t})$. As the maximization must be done simultaneously, the total number of calls to find π_* is $O\big(|A_*|^{n|Z_*|^T}\big)$, where $|Z_*|$ and $|A_*|$ represent the largest observation and action space of single robot. This complexity renders finding π_* through Equation 1.9 computationally equivalent to exhaustive search.[9]

[8] Again, for general-sum games this operator would not be an $\arg\max$, but would make use of game-theoretic concepts.

[9] For the common-payoff case, this bound can be somewhat reduced by more intelligent dynamic programming as seen in Section 7.2.

Recasting the problem of solving POSGs into Equation 1.9, therefore, is only really useful for drawing comparisons between MDPs, POMDPs and POSGs. Figure 1.4, the POSG analogy to Figures 1.1(a) and 1.1(b), gives this comparison in a graphical fashion. Each robot can use problem dynamics to create a probability distribution d over all possible joint histories of length T that could be held by the team. This probability distribution defines a single point d in the space of all possible distributions over joint histories, and this point will define a corresponding set of plans for each robot. These plans do not dictate a single action for each robot, but are conditional and allow a robot to perform a different action for each of its possible observation histories h_i^t. No belief hierarchies are necessary in such a model; however, all information relevant to action selection that would be modelled explicitly in a belief hierarchy is still present in a set of policies π. In other words, a robot can use those policies and its own observation history to determine a conditional probability over the set of teammates' observation histories (what it believes its teammates saw) and what actions they will select for any timestep of the problem (what it believes they will do) and vice versa.

1.3.1 Computational Tractability

Finite-horizon POSGs with common payoffs are a very powerful way to coordinate a team of decentralized robots. Robots are able to properly integrate both their own observations and those of their teammates into the decision-making process without having to reason about recursive beliefs or trying to deduce what actions their teammates will take. However, exhaustive search methods for finding such policies are computationally intractable as they requires iterating over $O\big(|A_*|^{n|Z_*|^T}\big)$ possible joint policies. Do more efficient methods for solving POSGs exist?

Unfortunately, it has been shown by Bernstein et al. (2002) that the problem of solving finite-horizon POSGs with common payoffs is non-deterministic exponential time complete (NEXP-complete). The authors also prove that this class of problems, even for instances that are collectively fully observable, is NEXP-hard for two or more robots. This means that there are exponential many sets of policies to search through for the optimal solution, and that constructing and evaluating each set takes exponential time. Therefore, in the worst case, a computer would take time doubly exponential in the planning horizon T to solve a POSG.

Unlike the case of finite-horizon POMDPs which are PSPACE-complete (Papadimitriou & Tsitsiklis, 1987), it can be proved that there are no polynomial-time algorithms for

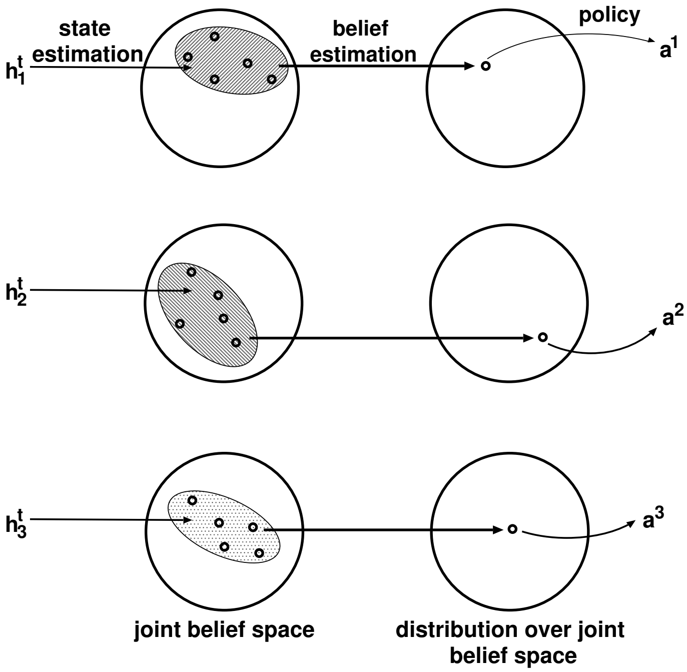


Figure 1.3: In this naive approach to decentralized planning, each robot considers all possible joint beliefs consistent with its own history of observations h_i^t and then plans in the resulting distribution over belief space d. That is, it finds the point d_i such that $d_i(b) = P(b|h_i^t)\,\forall b$. However, because each robot's observation history leads to a different d_i, its selected joint action a^i will also be different from the ones selected by its teammates even if they are in agreement about the mapping between each d and a. This state of affairs leads to miscoordination between robots.

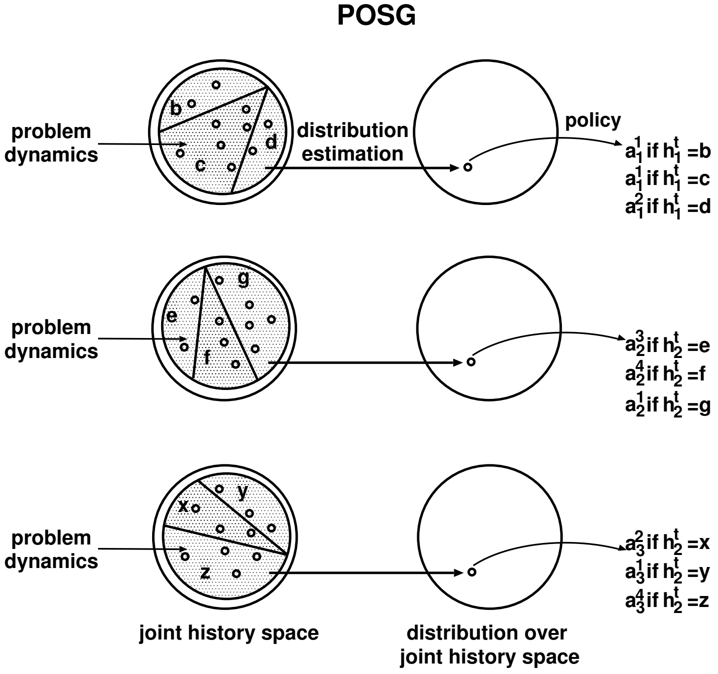


Figure 1.4: A graphical representation relating the concepts of planning with POSGs to those of MDPs and POMDPs. Each robot constructs the distribution over all possible joint histories using only information available to the entire team. It then constructs a conditional plan for every robot in the team that allows each robot to take a different action for each of its possible observation histories. The plans specify individual rather than joint actions for each robot, indicated here by the subscript. However, because each robot is planning for the same point, d, in the space of all possible distributions over joint histories, all robots will arrive at the same set of conditional plans and therefore be coordinated.

finding solutions to finite-horizon POSGs (i.e., it is known that P$\neq$NEXP and the NEXP-completeness of the POSG holds even if the horizon is smaller than the number of states). Indeed, assuming EXP$\neq$NEXP, in the worst case any algorithm for solving a POSG with common payoffs will be doubly exponential in the time horizon of the problem. This mathematical evidence backs up the intuition that solving decentralized problems is fundamentally harder than solving centralized problems or problems that can be treated as centralized due to communication. As Bernstein et al. (2002) point out, this also means that the key to solving POSGs exactly does not lie in discovering some way to convert them into POMDPs and then applying known solution techniques. Instead, a different approach is required. In the mean time, POSGs remain a powerful way to model tightly-coupled problems in Markovian environments and so the pursuit of approximate algorithms is important.

Furthermore, even if it is computationally intractable to solve a POSG, knowing how a framework for decentralized robot teams relates to this model can help provide insight into algorithms for problem solving. For example, the solutions generated by modelling a small version of the problem as a POSG could help create heuristics for larger versions of the problem. Alternatively, identifying what assumptions or approximations are made by an algorithm for action selection lets one understand what its limitations are with respect to performance. Finally, just because the general class of problems is NEXP-complete, it does not mean that all POSGs are hard to solve. Specific domain constraints can lead to solution techniques that are not applicable in general. These issues will all be explored in more detail in Chapter 7.

1.4 Bayesian Game Approximation Algorithm

As discussed in the previous section, the NEXP-complexity result for POSGs with common payoffs means that exact solutions for these problems will not be found through some conversion to POMDPs. While it is possible to approximate these solutions using POMDPs, it requires some modelling of teammates in order to properly deduce their effects on action selection. An alternative approach to explicit teammate modelling is one that retains the use of game theory to perform action selection for immediate timesteps but approximates the future value of actions using a heuristic. Furthermore, by interleaving planning and execution, robots can use past decisions to reduce the number of possible observation histories that must be taken into account during action selection at later timesteps.

The resulting algorithm for finding approximate solutions to POSGs with common payoffs is called the Bayesian game approximation, or BaGA. This algorithm uses a less well known game-theory construct, the Bayesian game, to model each timestep of a POSG. In each of the linked Bayesian games, robots reason about their current action selection by considering both immediate rewards and estimates of future rewards. These estimates come from heuristics that approximate how, for a given set of observation histories, robots will be able to perform in the future. Thus, robots only reason in the short term about true uncertainty in the observations and action selection of their teammates: a tradeoff has been made between reasoning about future uncertainty and computational tractability. However, if the heuristic function accurately evaluates the expected future value of actions, then BaGA becomes exact.

BaGA does not result in a set of full universal plans for the POSG but rather finds conditional plans for those parts of the POSG that can possibly occur (given earlier decisions) by concatenating together solutions from each of the smaller Bayesian games. By generating policies in a forward fashion and using heuristics in combination with game-theoretic solution concepts, much larger problems can be solved than previously possible. Indeed, it will be shown that it is even possible to generate real-time robot controllers using BaGA.

1.5 Thesis Statement

In this dissertation, I will advance the following thesis:

> Partially observable stochastic games (POSGs) are a powerful way to model decision making in a team of robots with limited communication. Approximate solutions to POSGs with common payoffs can be found by interleaving planning and execution, thereby transforming the problem into a series of smaller, linked Bayesian games. Furthermore, robots can generate good policies by reasoning only a limited time ahead about uncertainty in both world state and the experiences of their teammates.

1.6 Thesis Contributions and Outline

This dissertation presents an overview of the BaGA algorithm. Quantitative experiments compare the performance of BaGA to the solutions of full POSGs and basic heuristics on a variety of abstract and real-world problems. The core contributions of this dissertation can be summarized as follows:

- Familiarizing the robotics community with the POSG framework and its use in modeling the problem of decentralized decision making for teams of robots without full communication.
- The observation that, if a POSG is to be approximated in a forward fashion, then each timestep must be represented as a Bayesian game in order to keep robots coordinated. The basic BaGA algorithm for finding approximate solutions to POSGs is the result of this observation.
- An extension to BaGA which reasons over clusters of observation histories rather than every possible history, leading to computational savings without loss of performance.
- The use of BaGA to generate run-time communication policies that improve both computational efficiency and the expected performance of a robot team.
- An implementation of BaGA on a team of physical robots, which validates BaGA as a real-time robot controller.

Because several game-theoretic concepts are important to the development of the BaGA algorithm, the body of this dissertation starts with an overview of game theory rather than moving directly to the BaGA algorithm itself. Due to the similarity between POSGs and extensive-form games, the first part of Chapter 2 is devoted primarily to solution techniques for these game trees. Bayesian games are then introduced and discussed in detail in the latter half of the chapter. It is very important that the reader understands Bayesian games before proceeding because these games form the building blocks of the BaGA algorithm.

After all necessary game-theory concepts have been presented, the basic version of the BaGA algorithm is introduced in Chapter 3. BaGA is analyzed with respect to computational savings and the levels of approximation it makes in order to find tractable solutions to POSGs with common payoffs. Experimental results, validating BaGA, are presented for three domains of various size. In Chapters 4 and 5, two extensions to the basic algorithm that decrease computational costs are introduced: clustering together of similar histories and the addition of communication acts.

The basic algorithm and its two extensions are brought together in Chapter 6 to show how a relatively abstract notion of control, game theory, can be used to develop real-time robot controllers. In this chapter, a series of large, realistic robot tag problems are described and implemented in a high-fidelity simulator and on a physical robot team. BaGA is able to successfully control the robots in real time and shows performance improvements over common heuristics used for robot control. Chapter 6 closes with a discussion of how the BaGA algorithm meets the requirements of a software framework for robust, real-time multi-robot control.

While POSGs themselves are not a novel contribution of this thesis, the term POSG is still relatively unknown in the fields of multi-agent and multi-robot control (with the exceptions being economics and game theory). Over the past ten years, many different decision-theoretic frameworks have been proposed for controlling decentralized agents in stochastic environments. However, analysis of the parameters and constraints of these frameworks reveals that they are actually equivalent to various sub-classes of POSGs. In Chapter 7, this pantheon of frameworks and their relationship to POSGs with common payoffs is discussed, as are approximation algorithms for each of the framework and how they relate to BaGA. Chapter 7 also includes a discussion of several alternatives to decision-theoretic decentralized decision making: joint intentions, behaviour-based approaches and market-based approaches. The ways in which robots take their teammates into account during decision making in these frameworks are analyzed and relationships between these models and POSGs identified. Finally, in Chapter 8, some concluding remarks are made.

Chapter 2

Game Theory and Bayesian Games

This chapter provides a broad overview of key representations and solution techniques in non-cooperative game theory in order to provide the reader with enough game-theory background to understand BaGA, the POSG approximation algorithm presented in this dissertation. It may be safely skipped by those familiar with game theory. Readers familiar with basic game theory, but not with Bayesian games can find that information in Section 2.2.1.

Non-cooperative game theory is the study of strategic decision making: how self-interested agents choose actions in the presence of other agents who are also choosing actions. In a game with common payoffs, each agent has the same reward function and, despite selecting actions in a self-interested fashion, can still act cooperatively with others. Because this thesis deals with robot teams, this chapter will focus on games with common payoffs and, as such, is not intended to be an exhaustive overview of the entire field. There are many texts that provide a good introduction to game theory, including Owen (1968), Basar & Olsder (1982) and Fudenberg & Tirole (1991).

Information plays an integral role in choosing actions and this results in two different classes of games: games of imperfect information and games of incomplete information. Games of imperfect information are games in which all the aspects of the game are known to every player. While a player might not know what specific actions other players will take, it does have access to all the information upon which those decisions will be made. Extensive-form and normal-form games fall into this category as do POSGs. Games of incomplete information are games in which players do not have access to information crucial to the decision-making process of other players. This includes situations in which the number of players in the game, the reward functions of certain players or even the game itself is unknown to at least one player. A combination of game theory and decision theory leads to a solution technique for these problems known as Bayesian games.

Because Bayesian games form the basic building blocks of the BaGA algorithm, it is important to understand how they work. First, however, an overview of games of imperfect information and their solution techniques will be presented. This is done to provide the necessary foundation for understanding the more complex Bayesian game.

2.1 Games of Imperfect Information

This section describes the most common types of games: extensive and normal-form games. Extensive-form games and POSGs are actually quite similar; the major difference between these two frameworks is that while POSGs include the notion of world state in their definitions, extensive-form games do not. Because of this similarity, many of the issues about solving POSGs that were addressed in Chapter 1 also come up in the simpler extensive-form model. The sequence form, presented in Section 2.1.5, is a more compact representation of a multi-stage game that was developed to alleviate some of the difficulties of solving large extensive-form games (Koller et al., 1996; von Stengel, 1996). This technique will also be used in Section 3.1 to construct an algorithm for finding approximate solutions to an entire POSG at once. While this algorithm is not as efficient as the BaGA algorithm, it will be useful for evaluating the performance of BaGA.

This section ends with a discussion of correlated equilibria and sub-game perfect equilibria. These concepts are described and their relevance to common-payoff games and this thesis discussed.

2.1.1 The Extensive Form

Classic board and card games follow a distinct pattern. Players take turns selecting a move from a range of possibilities until the game is over and one player is declared the winner. In some games, chance moves can occur due to the roll of a die or the shuffling of a deck of cards. In board games, the entire state of the game is usually visible to all players, while in card games much information is hidden. All of these games, with their different variations in chance moves and visibility, can be intuitively represented with a tree structure called the *extensive form*. The root of the tree indicates the starting condition of the game and each leaf represents a terminal condition (with an associated payoff). Internal nodes are separated into chance nodes and player nodes, and represent decision points for the associated players. Vertices between nodes are moves amongst which a player (or chance) may decide. A vertex associated with a chance node generally has a probability assigned to it that indicates the probability that the associated move will occur. These chance nodes can also be thought of as nodes associated with the player Nature.

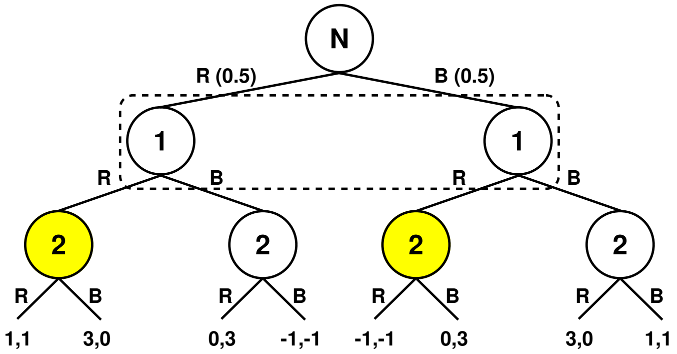
The actual play of the game is represented by a single path through the tree that reflects all of the actions taken by the players and any outcomes of the chance elements. The outcome of the game is given by the payoff associated with the terminal node reached.

If a player cannot distinguish between two or more nodes in the tree, those nodes are joined together into an *information set* and the player must make the same decision for all of the nodes in that set. For example, if it is a player's turn in Bridge, then all of the possible hands that are consistent with the cards she was dealt are represented by a single node in the tree. Because she cannot view the cards of the other players, she cannot distinguish between any of these hands and so she must make the same decision for all of them.

Figure 2.1 (a) shows the extensive-form representation for a very simple card game that captures all of these ideas. There are two players, each with a single turn. A card is drawn from a standard deck of cards at random and the first player makes a guess at its colour. After hearing Player 1's action (her guess), Player 2 then guesses the colour of the card. If both players correctly guess the same colour then they both get a payoff of 1. If only one is correct, then she or he gets a payoff of 3 and the other a payoff of 0, and if they both guess the same colour incorrectly then they both get a payoff of -1. This game is not a common-payoff game but rather a *general-sum* game because the players have different reward functions and those rewards do not add up to a constant sum (what would be a *zero-sum* game).

In this game, Player 1 has two nodes in the tree, one in which a red card is drawn and one in which a black card is drawn, but only one information set because she does not actually know which card was drawn. Player 2 has four nodes in the tree, one for each combination of Player 1's guess and the colour of the card. However, as Player 2 only knows what Player 1's action was, these nodes must be grouped into two information sets: one for each of Player 1's actions.

If this game were to be extended into a larger game, perhaps with players getting a chance to amend their guesses after hearing the other player's guess, then the tree would simply increase in size. Each player would have two decision-making points in the game: she would make a guess in the first round and then a second guess in the second round after



(a) Extensive-form representation.

		Player 2			
		RR	RB	BR	BB
Player 1	R	(0,0)	(0,0)	(1.5,1.5)	(1.5,1.5)
	B	(1.5,1.5)	(0,0)	(1.5,1.5)	(0,0)

(b) Normal-form representation.

Figure 2.1: Example of the extensive-form (a) and normal -form (b) games for a very simple 2-player problem. In the game tree the root node N represents the drawing of a card from the deck at random by the Nature or chance player. Player 1 then guesses whether or not the card is red (R) or black (B). She has one information set indicated by the two nodes within the dashed box. Player 2, after hearing Player 1's guess, then makes his own guess. He has two information sets, one for each guess of Player 1. His two information sets consist of his shaded and unshaded nodes respectively.The payoffs to the two players are shown at the terminal nodes with the payoff to Player 1 given first. The normal form game is represented as a matrix M of payoffs to each player. For example, the strategy pair (R,BR) has the outcome (3,0) if Nature selects a red card, but (0,3) if a black card is chosen. The expected value of this strategy pair is therefore (1.5,1.5) as the card can be red or black with equal probability.

hearing the other player's selection. The specific guesses, or actions, made by a player are known as the player's *strategy*. A strategy is a plan that tells a player what action it will select for each information set in the tree. In extensive-form games there are two types of strategies: a *pure strategy* and a *behaviour strategy*. A pure strategy is a strategy in which only one action is played deterministically at each information set. For example, if Player 1 decides to always guess red then she is playing a pure strategy. A behaviour strategy is used to described randomized strategies. In these strategies a probability distribution over the action set of each player is defined for each information set. If an information set is reached during game play, then a player will randomly select one of its actions using the given distribution. For example, Player 1 could have a behaviour strategy for her only information set that assigns a probability distribution of $\{0.5,0.5\}$ to the guesses of red and black respectively. While she will only play one of these two actions during an actual instance of the game, if the game were to be repeated multiple times then, on average, she would guess red and black roughly an equal number of times.

The criteria for decision making has not yet been addressed and so a discussion of what the optimal strategy for each player would be in such a game is left until later.

2.1.2 The Normal Form

In an extensive-form representation of a game, each player is thought of as not making a decision until it reaches a specific information set during the course of play. However, for many games a player does not have to wait until that point to make a decision. For example, in the card game in Figure 2.1, Player 2 does not need to wait until Player 1 has selected a move to choose his own move. Instead, he can decide ahead of time on a plan for how he will react to Player 1. This plan will be conditional as it will tell the player which action he will select in response to each possible action of Player 1. For example, his plan might be to guess red if Player 1 guesses black and black if Player 1 guesses red. Player 1 can also make such a plan before the game begins. As in extensive-form games, these plans are called strategies and each strategy assigns a move to each information set that can be encountered by a player during the entire course of the game.

In this simple card game, Player 1 has two pure strategies: play red or play black (R or B). She has only two possible strategies because she has only one information set at which to make a decision and only two possible action choices. Player 2, however, has two information sets, each of which has two possible actions (R or B), yielding four possible pure strategies for the game. If the first element represents what he will do if Player 1 selects

R and the second if Player 1 selects B, then these four strategies are: RR,RB,BR and BB. For example, the strategy in which Player 2 always makes the opposite guess to Player 1 is given by BR.

The *normal-form* representation of a game is a tabulation of the expected payoffs to each player for each possible combination of pure strategies. The payoff for a pair of strategies is determined by the expected value of the terminal node(s) reached by following the path(s) indicated by the strategies through the game tree. For example, as shown in Figure 2.1, the strategy pair (R, BR) has two possible paths through the tree: one in which the card selected is red, Player 1 guesses red and Player 2 guesses black leading to a payoff of (3,0) and a second path in which the card selected is black, Player 1 chooses red and Player 2 chooses black leading to a payoff of (0,3). As each path occurs with equal probability of 0.5, the overall payoff to the players is (1.5,1.5). The two players make the same action selections in each path because Player 1's strategy dictates that only one action, R, will be played and Player 2's conditional plan also yields the same action both times because Player 2's choice is conditioned only on Player 1's choice.

As in extensive-form games, in normal-form games a pure strategy is a strategy in which only one action is selected at each information set. In this case, Player 2 has four pure strategies while Player 1 has two pure strategies. Randomization over pure strategies, however, results in *mixed strategies* rather than behaviour strategies. A mixed strategy consists of a single probability distribution over all of the possible pure strategies for the game. Before game play starts, a player randomly selects one of her pure strategies according to this probability distribution. During game play, she then follows that pure strategy with no further randomization. For example, a mixed strategy for Player 2 might be $\{0.35, 0.2, 0.15, 0.3\}$ which means he will play RR 35% of the time, RB 20% of the time, etc.

As opposed to behavioural strategies, which only involve independent randomizations, mixed strategies permit correlations across information sets. For this reason, a behaviour strategy can be achieved by a mixed strategy (by more than one, in some cases) but not every mixed strategy can be achieved by a behaviour strategy. In games of perfect recall (i.e., no player forgets an action or observation made in a previous round), mixed and behaviour strategies are interchangeable as any probability distribution over outcomes which can be achieved by one can be achieved by the other.

2.1.3 Formal Definitions

A general-sum n-player normal-form game is defined as a tuple (I, A, R) where I is a set of players 1 through n, $A = \{A_1, \dots, A_n\}$ is a set of action choices for each player and $R = \{R_1, \dots, R_n\}$ is a set of payoff functions. The payoff function $R_i(a_1, \dots, a_n)$ gives the payoff to player i for each combination of actions $a_i, \dots, a_n$ taken by all players. A 2-player normal-form game is generally represented as a set of matrices $\{M_1, M_2\}$. Each element r^i_{jk} of matrix M_i corresponds to the payoff to player i when Player 1 selects strategy $a_j \in A_1$ and Player 2 strategy $a_k \in A_2$. That is, each row of the matrix represents the payoffs for a strategy for Player 1 and each column the payoffs for a strategy for Player 2.

A strategy is a function that assigns an action to each information set of a player. There are three types of strategies: pure, behaviour and mixed. A policy in a normal-form game for player i, π_i, is defined to be a probability distribution over its set of strategies. If the game involves only one instance of action selection per player (a *single-stage* game), then this set of strategies is the set of actions A_i. A pure strategy is represented by a policy π_i in which only one element is non-zero while a mixed strategy is represented by any probability distribution. Under a mixed strategy, the probability of playing a specific action a_j is given by $\pi_i(a_j)$.

The expected payoff to player i under policy π_i, assuming that the other players adopt policies $\pi_1, \dots, \pi_{i-1}, \pi_{i+1}, \dots, \pi_n$, is given by the expected value of R_i when weighted by the probabilities of each action combination given by $\pi = \{\pi_1, \dots, \pi_n\}$:

$$R_i(\pi) = \sum_{a \in A} \big(\prod_{j=1}^{n} \pi_j(a_j) \big) R_i(a_1, \dots, a_n) \tag{2.1}$$

where a_j is the individual action of player j in the joint action a.

A general-sum n-player game with multiple action selections for each player is defined similarly to a single-stage normal-form game, but it also includes information about the number of steps in the game, who makes decisions at which points, and the action set available to each player at each of their decision points. In a mixed strategy for such a game, the probability of playing the specific sequence of actions s_j (i.e., a specific strategy) is given by $\pi_i(s_j)$. The reward for a set of strategies, $R_i(s)$, is simply the sum of rewards over the sequence of action choices. The expected payoff to player i, given a sequence action choices, can therefore be defined by replacing the action a_j with the strategy s_j in Equation 2.1.

A behaviour strategy for an extensive-form game, as stated earlier, is a probability distribution over the action set for each possible information set. That is, it is a set of probability distributions $\pi_i = \{\sigma_1, \ldots, \sigma_k\}$ where σ_k defines a probability distribution over the possible actions at information set k . Similarly to a normal-form game, for a pure strategy each of these probability distributions will have only one non-zero element.

2.1.4 Solution Techniques

While the definitions of a strategy and policy for extensive- and normal-form games have been given, what is meant by a solution to the game has not yet been defined. A solution to a game is a set of policies, π, which dictate how each player should play the game in order to optimize some criterion.

First, the notion of a *best-response* policy must be defined. A best-response policy π_i^* is the policy for player i that maximizes its expected reward given a set of known policies π_{-i} for all other players:[1]

$$\pi_i^* = \arg\max_{\pi_i} R_i(\pi_i, \pi_{-i}).$$

The benefit of playing mixed strategies now starts to make sense: if any other player knows exactly what strategy player i will select, then she can guarantee playing her best-response to that specific strategy. If player i instead plays according to a mixed strategy, the other players can only select policies that are best responses in expectation. That is, player i can now hide its specific strategy selection from the other players in order to achieve a higher payoff for itself (randomization also has other benefits in a general-sum game). A basic requirement for a stable solution to a game is that the set of policies π be a set of best-response strategies.

There are various solution concepts for games, the most well known of which is the Nash equilibrium (Nash, 1950). Every normal-form game has at least one set of best-response policies π^* such that no player i can improve its expected payoff by unilaterally changing its policy π_i^*. That is:

$$R_i(\pi_i^*, \pi_{-i}^*) \geq R_i(\pi_i, \pi_{-i}^*) \qquad \forall \pi_i, i \in I.$$

[1] The subscript $-i$ refers to all players other than i. For example, $\pi_{-i} = \pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n$.

These equilibria points are called Nash equilibria. While some games can have Nash equilibria in pure strategies, every game of this form is guaranteed to have at least one mixed-strategy Nash equilibria (Nash, 1950). A game may have multiple Nash equilibria and selecting an appropriate one to play can become part of the decision-making process for certain games. Unfortunately, it is very difficult to find Nash equilibria for general-sum n-player games (see McKelvey & McLennan (1996) for an overview of solution techniques including approximation techniques). Games with common payoffs, however, have nice properties that make them easier to solve.

First, in games with common payoffs, there is no benefit to hiding one's strategy from teammates or for randomizing action choices and so the Nash equilibrium solutions can be found in pure strategies. Furthermore, not only can a Nash equilibrium be found, but the set of Pareto-optimal Nash equilibria can also be found. In general, a Pareto-optimal Nash equilibrium point is one in which it is not possible for one player to be better off without harming the performance of any other players.[2] For common-payoff games, in the Pareto-optimal Nash equilibrium each player is receiving its highest possible payoff out of all possible Nash equilibria of the game. If it were not the case, and there is another equilibrium in which one player could receive a higher payoff, then there must be an equilibrium in which all players receive a higher payoff because they have the same reward function. Because all players receive this higher payoff, no players are harmed by moving to the new equilibrium point. But, by definition, the original Nash equilibrium could not have been Pareto-optimal. Therefore, the set of Pareto-optimal Nash equilibria for common-payoff games correspond to the set of Nash equilibria with highest possible payoff to each player.

Furthermore, these Pareto-optimal Nash equilibria can be defined to be the action set(s) that correspond to the maximum element(s) in R.[3] If multiple Pareto-optimal Nash equilibria exist, then players will require some form of coordination strategy (e.g., Boutilier (1996); von Stengel & Koller (1997); Wang & Sandholm (2002)) to ensure that they select the same equilibrium. Without this coordination, players might select their actions from different equilibrium points and the resulting play is generally sub-optimal (i.e., not an equilibrium point itself).

Solutions to extensive-form games are much harder to find. If each information set col-

[2] Not every Nash equilibrium is Pareto-optimal. Consider a case with two Nash equilibria with payoffs 1.0 and 2.0 to all players, respectively. While considering the first Nash equilibrium, Player 1 cannot receive more by unilaterally changing its policy; however, she could do better if everyone were to switch to the second equilibrium point, making this second one Pareto-optimal for the problem.

[3] For a fully observable, multi-step common payoff game, these Pareto-optimal Nash equilibria correspond to the optimal solution that would be found for the MDP formulation of the problem in joint-action space.

lapses to a single node in the tree then a solution can be found through backward induction, starting at the terminal nodes of the tree and working upward (Fudenberg & Tirole, 1991). Backward induction fails if information sets contain multiple nodes as the same action must be selected for each node in the set. Instead, a generalization of backward induction, sub-game perfection, can be used to find policies (Fudenberg & Tirole, 1991).

Two alternatives exist for finding solutions to extensive-form games. The first is to convert the game into its normal-form equivalent and to then find optimal policies. In general, the resulting policies will be mixed strategies rather than behaviour or pure strategies; however, in common-payoff games, the solutions will correspond to pure strategies. The problem with this method is that the number of possible pure strategies that must be represented is exponential in the size of the game tree and, as the size of the game tree increases, it rapidly becomes computationally infeasible to solve the corresponding normal-form game. This exponential scaling affects not just general-sum games but also common-payoff games; while it is easier to solve the common-payoff case, conversion to normal form still requires an iteration over every combination of possible pure strategies.[4]

For example, consider the game in Figure 2.2. In this common-payoff game each player has two turns in which to select between the actions A and B. In the first round, each player selects an action simultaneously, then in the second round they are informed of the other player's previous selection and, using that information, once again select an action simultaneously. The players both receive a payoff of 1 for each round in which they select the same action and a payoff of -1 for each round in which they do not. Each player has five unique information sets: an initial information set in which she has no information and then four information sets, each one corresponding to a combination of her first move and the other player's first move, to represent the second round. For each of these five information sets, the player has two action choices and therefore 2^5 different possible pure strategies. The normal-form matrix representing this game would be 32×32. If the game were expanded to include a third round then each player would have an additional 16 information sets leading to a total of 2^{21} pure strategies each.

In general, if each player has the same action set with size $|A|$ and has k possible information sets during the game, then each player has $|A|^k$ different possible pure strategies and determining the Pareto-optimal Nash equilibrium will require representing $|A|^{kn}$ tabulated

[4] This statement does not contradict Chu & Halpern's (2001) NP-completeness proof for finding the optimal solution to a common-payoff extensive-form game . While straightforward, conversion to the normal form is not the most efficient method for solving extensive form games. In Section 2.1.5, the sequence form is presented and an algorithm for finding solutions to common-payoff problems using this more efficient representation discussed. This latter algorithm is more consistent with Chu & Halpern's result.

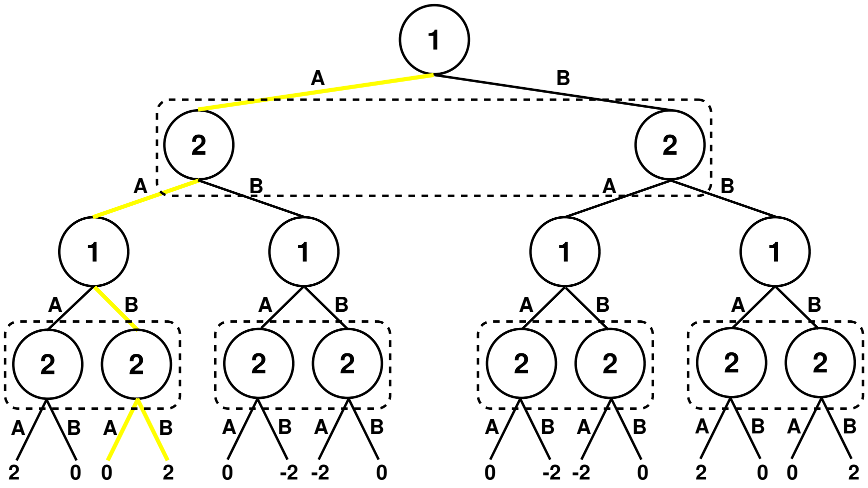


Figure 2.2: A more complicated game tree showing a game with two turns per player. In this common-payoff game, the two players select between two actions A and B simultaneously, are informed of each other's action choice and then select between the two actions again simultaneously. Each player has five unique information sets. Nodes within a dashed box indicate one information set. The players receive a reward of 1 for each round in which they select the same action, and a reward of -1 for each round in which they do not. The highlighted path through the tree indicates the path realized by the sequence AB_A for Player 1.

payoffs. While k may be low for games with only one decision point, as the number of times a player makes a decision grows, so does k. For example, if the players each select an action at timestep t and know the actions taken by all players at times $1, \ldots, t-1$ then by timestep t each player will have:

$$k = \sum_{j=1}^{t} \left(|A|^n\right)^{j-1},$$

which makes the number of possible pure strategies per player for this type of game doubly exponential in the number of players.

The alternative is to convert the game into its corresponding *sequence form* and then find a solution to the new game.

2.1.5 The Sequence Form

The sequence form is designed to find equilibrium policies in extensive-form games that are too large to solve by conversion to normal form (Koller et al., 1996; von Stengel, 1996). Specifically, for 2-player games, once the game has been converted into its sequence form, it can be solved as a linear complementarity problem. For n-player common-payoff games in sequence form, an alternating-maximization algorithm can be used to find locally optimal solutions.

The sequence form of a game is similar to the normal form of a game except that, instead of using pure strategies as the action set of each player, sequences are used. A sequence of actions σ_j^i is the series of choices made by player i at each information set it encounters as it traverses a single path j through the game tree. Naturally, this sequence can only occur, or be realized, if all other players and Nature also make selections that lie on the path j. For each player there are at most as many sequences as there are nodes in the tree, and so the sequence form is linear in the size of the game tree (instead of exponential).

A sequence of moves is highlighted in Figure 2.2. It corresponds to Player 1's sequence of actions: AB_A (play action A and then play action B if Player 2 plays action A in the first round). This specific sequence, however, is only realized if Player 2 has a series of actions that starts with action A and then makes an action choice conditioned on Player 1's first action of A; otherwise, it cannot occur. Player 2 actually has two sequences that permit Player 1's sequences to be realized: AA_A and AB_A, both of which are part of the highlighted path.

In general, for the sequence form, if a player has k information sets, then she has $k|A|+1$ sequences (the additional sequence is a empty sequence, $\emptyset$, used for the purpose of building constraints and the set of sequences includes all sub-sequences). For the example game, the total number of sequences is exponential in the number of players, because k is exponential in the number of players. Unlike the total number of pure strategies, however, it is not doubly exponential. For the game in Figure 2.2, the sequence form has 11 possible sequences per player: $\emptyset$, A, B, AA_A, AA_B, AB_A, AB_B, BA_A, BA_B, BB_A and BB_B. In contrast, there are 32 different pure strategies. If the game is extended to a third round, then the number of sequences is 41 compared to the 2^{21} pure strategies of the normal form.

If σ_u represents a sequence of actions that lead up to information set u, then $\sigma_u a$ is the sequence of actions that includes the action choice of a at information set u. A realization plan x corresponds to a behaviour strategy for a player in which action a is played at information set u with probability $x(\sigma_u a)/x(\sigma_u)$ (Koller et al., 1996). The goal of solving a game in sequence form is to find a realization plan x for Player 1 and a realization plan y for Player 2 that correspond to a Nash equilibrium.

An equilibrium is formed by the realization plans x and y if each of the plans is a best response to the other. In order to solve for the optimal x and y simultaneously, the linear complementarity problem that simultaneously maximizes the payoffs to both players subject to constraints on x and y must be created and solved, for example, using Lemke's algorithm. The payoffs to Players 1 and 2 are given by the matrices B and C respectively. Each element of B, b_{jk}, is the expected payoff to Player 1 if she takes the path given by her sequence j while Player 2 takes the path given by his sequence k. There are constraints $Ex = e$ and $Fy = f$ that are designed to ensure that an action is selected for each information set and that the sum of the probabilities of reaching the children of an information set is equal to the probability of reaching that information set. Under the realization plans x and y, the expected payoff to Player 1 is $x^T By$ and to Player 2 is $x^T Cy$. The optimal solution maximizes the following two equations, subject to the given constraints:

$$\begin{aligned} \max_{x} \quad & x^T(By) \\ \text{subject to} \quad & x^T E^T = e^T \\ & x^T \geq 0 \end{aligned} \tag{2.2}$$

and:

$$\begin{aligned} \max_{y} \quad & (x^T C)y \\ \text{subject to} \quad & Fy = f \\ & y \geq 0. \end{aligned} \tag{2.3}$$

See Koller et al. (1996) and von Stengel (1996) for more details on this algorithm.

Solving the sequence form is non-trivial for arbitrary n-player games; however, a set of locally optimal best-response strategies for common-payoff games can be found using an *alternating-maximization* algorithm. Holding the realization plans, x_{-i}, of all but one player fixed, linear or dynamic programming can be used to find a best response plan x_i for the remaining player. Optimization is done for each player, in turn, until no player wishes to change its realization plan (this process will eventually converge for common-payoff games because the policies are deterministic). The resulting equilibrium point is a local optimum, but random restarts can be used to further explore the strategy space. During this optimization process, it can be ensured that players have realization plans equivalent to pure strategies rather than behaviour strategies by using some rule for breaking ties between actions that are equivalent in reward.

For a 2-player game this algorithm is equivalent to repeatedly performing the maximizations given by Equations 2.2 and 2.3 but instead of doing so simultaneously, first y is held fixed while $x^T(By)$ is maximized and then x is held fixed while maximize $(x^T B)y$ is maximized.

2.1.6 Alternate Solution Concepts

In general-sum games, correlated equilibria (Aumann, 1974) are gaining popularity over Nash equilibria. In a Nash equilibrium, each player defines an independent probability distribution over what actions to play. In correlated equilibria, however, players can condition their action selection on an external signal in order to achieve higher payoffs. The benefits of a correlated equilibria can be easily seen in the classic Battle of the Sexes problem. In this 2-player game, each player selects from actions A and B. Player 1 has a stronger preference for action A and Player 2 for action B, but both prefer to play the same action as each other leading to the payoff function in Figure 2.3. This game has two pure strategy Nash equilibria: ($\pi_1 = \{1, 0\}, \pi_2 = \{1, 0\}$) and ($\pi_1 = \{0, 1\}, \pi_2 = \{0, 1\}$). These equilib-

		Player 2	
		A	**B**
Player 1	**A**	(2,1)	(0,0)
	B	(0,0)	(1,2)

Figure 2.3: Example of a general-sum game with a payoff structure equivalent to that of Battle of the Sexes. Each player has a preference for a different action choice but overall prefers to choose the same action as its opponent.

ria lead to unequal payments to the two players: each player gets 2 or 1 depending on which equilibrium is selected. A mixed Nash equilibrium solution has each player independently define a probability distribution over their individual action choices in proportion to how much he prefers one action to the other. The mixed strategy equilibrium of $\pi_1 = \{2/3, 1/3\}$ and $\pi_2 = \{1/3, 2/3\}$ generates an expected reward of 2/3 to each player. Even though this equilibrium generates a lower outcome to both players, it could be considered more fair because it does not favour one player over the other. Additionally, if each player opts to play according to the pure strategy Nash equilibrium that favours her, then the resulting actions do not form a Nash equilibrium at all (i.e., AB or BA) and each player would receive only 0.

During game play, even if players select their actions according to their mixed strategies, due to the independence of the probability distributions, the sub-optimal joint actions of AB or BA can occur. If, instead, action choices are allowed to be conditioned on something like a coin-flip (the outcome of which is observed by each player), then players can define equilibrium policies that only result in situations in which AA or BB are played. For example, if the coin is Heads, both players play A, otherwise B. In this correlated equilibrium, their expected payoff is now 3/2 which is clearly better than that given by the mixed Nash equilibrium solution (and is equivalent to the players alternating between the two pure strategy Nash equilibria). Players will follow the policies induced by the correlating device because they do not have any incentive to deviate: if Player 2 knows that Player 1 will play A if the coin is Heads, his best response is to also play A.

Correlated equilibria are equivalent to Nash equilibria in a larger version of the game that includes information about the correlating device. So the Nash equilibria of the original game will also appear in the set of correlating equilibria. It is possible to define a universal correlating device, rather than rely on an external device, in which a trusted third party gives advice on what actions to play. As with a randomized correlating device, players will have no incentive to deviate from the given advice. The correlated equilibrium in which the payoff to each player is maximized can be found in polynomial time by constructing a linear program with the appropriate constraints (e.g., that the sum of each player's payoff is optimized). However, constructing constraints that express the incentives that each player has to play according to the correlated equilibrium still requires enumeration over all the possible joint actions or joint strategies.

In common-payoff games, the algorithm used to find Nash equilibria and to resolve conflicts between multiple possible Nash equilibria can be thought of as a correlating device. Additionally, as both Nash and correlated equilibria will exist in pure strategies, there would be no gain in reward to be achieved by switching from one solution concept to the other (i.e., as each probability distribution is a point distribution, no farther correlations are necessary).[5] Therefore, it is not necessary to use correlated equilibria as a solution concept for these types of problems unless it simplifies the search for a Pareto-optimal Nash equilibrium. For this reason, this thesis will focus solely on Nash equilibria.

2.1.7 Sub-Game Perfect Equilibria

For games with multiple turns, the issue of sub-game perfect equilibria must be addressed. For extensive-form games solved using backward induction or sub-game perfection then this issue is moot; however, if the game is solved by conversion to normal or sequence form then it is important. Basically, while a strategy for a player defines what it would do for every information set in the game, it is possible that decisions made early during game play will render parts of the game tree (i.e., sub-games) unreachable.[6] Because any decisions made in these unreachable portions will have no effect on the expected value of the game (if calculated using the normal or sequence form), their effect on Nash equilibria strategies is unusual; there can now be multiple Nash equilibria that make the same decisions (and therefore have the same expected value) for all the reachable parts of the game tree but make different action choices in the unreachable portions. Which Nash equilibrium is more desirable? For general-sum games, the notion of credible threats makes it important to play sub-game perfect equilibria. However, for a common-payoff game, if one truly believes that

[5] If the policies of the agents are stochastic, then correlated equilibria may have higher payoffs. See Bernstein et al. (2005) for an example of using finite-memory stochastic controllers that include a correlating device to approximate the solutions to infinite-horizon common payoff POSGs.

[6] By unreachable, it is meant only those parts of the game tree that are impossible given any common information rather than those information sets off of the current path of play. For example, in Figure 2.2, if Player 1 policy selects action A in the first round then, for the second round, it is known that any of the sub-games that would be reached by Player 1 selecting action B in the first round are now unreachable.

those portions of the game tree will never be visited (because they represent sub-optimal play), then it is perhaps not important. But if a mistake could occur, either during action selection or action execution, then the strategies taken by players in these now reachable parts of the game tree are important.

A sub-game perfect equilibrium (sometimes referred to as a perfect equilibrium) defines strategies for each player that are in equilibrium even for those parts of the game tree that can never be reached due to earlier decisions. That is, while any strategy defines what a player would do for each information set, only a sub-game perfect equilibria is guaranteed to be made up of strategies that are best responses to each other at every point in the tree. It is an extremely stable solution concept because it ensures that players will recover and play optimally in the future, even if some mistake is made by a player at an earlier timestep. Most of the solution methods looked at in this thesis will not, in general, generate sub-game perfect equilibria. This is because, from the point of view of the normal or sequence form, a sub-game perfect equilibrium and an equilibrium that defines identical action choices for all reachable portions of the tree (but different actions for unreachable sub-games) will generate the same expected value and so the more 'optimal' of the two can not be identified. Therefore, if the domain is such that noisy actions can occur, this will be reflected in the problem dynamics.

2.2 Games of Incomplete Information

This section describes an example of an incomplete information game: the Bayesian game. In games of incomplete information, players do not have access to information crucial to the decision-making process of other players. While POSGs are games of imperfect information, if one approximates a POSG by interleaving planning and execution, then each timestep must be represented as a Bayesian game in order to guarantee that the agents remain coordinated. This observation is the cornerstone of the BaGA algorithm which is presented in Section 3.2. Because understanding Bayesian games is crucial to the rest of the thesis, this section includes a high-level overview of this kind of game, a formal definition and a simple 2-robot example.

2.2.1 Bayesian Games

Bayesian games model single state problems in which each player has private information about something relevant to the decision-making process (Harsanyi, 1968). While this private information can range from knowledge about the number of players in the game to the action set of the other players, 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.[7]

The games shown in Figures 2.1(a) and 2.2 do not have this type of private information; while the players may not know the actions taken by the other player, they do have access to all information relevant to the decision-making process. Each player knows the payoff functions for all players, the actions available to all players and the dynamics of the game. Using this information, each one can solve the game in parallel to find a policy for all players and, provided that there is a way to break ties between different Nash equilibria, they are guaranteed to find the same policy π.

An example of a game in which players do have private information would be an art auction in which each player has received an appraisal for a painting from a different appraiser. The players must now decide upon a bid using only their own evaluation of the painting's worth and not those of other players. If Player 1 makes assumptions about the values held by other players, perhaps naively assuming they are the same as her own, then she will create and solve a game G_1 to find a set of policies π^1. Player 2, however, following the same reasoning but having a different evaluation will create and solve a different game G_2 to find π^2. The players are no longer playing the same game and so will no longer play according to an equilibrium solution. This situation leads to sub-optimal performance on the part of one or both players. The Bayesian game framework, however, is able to properly represent the information known by each player and guarantee that each player is once again playing the same game. Generally, it does so by making the assumption that a common prior over all possible instantiations of joint private information is available to all players.

Games with incomplete information such as this auction might lead one to think about solutions in terms of infinite hierarchies of beliefs: Player 1 has beliefs about the evaluation of the painting, she has beliefs about the other player's evaluation of the painting, she has beliefs about the other player's beliefs about herself and so on. Using this infinite belief hierarchy, one could imagine each player being able to construct the same, albeit large,

[7] In general, utility will be used in this thesis to refer to the payoff or reward function for a problem modelled as a Bayesian game. This choice was made to keep the discussion in line with the game-theory literature.

game and find a solution. Luckily, this infinite belief hierarchy can be replaced with a probability distribution over the space of all possible joint sets of private information.

While each player has received a different evaluation of the painting, those evaluations are not completely independent as they are based upon the same painting. This allows the assumption to be made that, based on some publicly known facts (e.g., sums for which paintings by the same artist were previously auctioned), the players can determine a probability distribution over the set of all possible combinations of painting evaluations for Player 1 and Player 2. For example, the combination of evaluations $\{\$100, \$90\}$ has a higher probability of occurring than $\{\$100, \$5\}$ if previous paintings were valued in the range of $95. Furthermore the assumption is made that each player calculates the same probability distribution over the joint set of evaluations (this assumption will be explained in more detail later). Each player can now create a larger game in which pure strategies define an action choice for each possible evaluation she can hold for the painting. To keep things simple, assume that players can either bid an amount equal to the value they hold for the painting (B) or not bid (N), and that they either have a high evaluation or low evaluation for the painting. An example pure strategy for Player 1 would be BN, in which she bids if she has a high evaluation for the painting and otherwise does not bid. The reward received for this strategy depends not only on the payoffs of the game but also on the distribution over possible evaluations.

In the game-theory literature, the private information held by a player is called its *type*, and it encapsulates all non-commonly-known information to which the player has access. The set of possible types for a player can be infinite, but this thesis will be limited to games with finite sets. Each player knows its own type with certainty but not those of other players. Beliefs about the types of others are given by a commonly held probability distribution over joint types. So long as the type space meets certain consistency requirements, infinite recursion about beliefs about the beliefs of others can be modelled as a prior over joint types rather than explicitly as a hierarchy. That means that, for each player, the subjective probability distribution over other players that would arise from reasoning about an infinite hierarchy of beliefs is the same as the conditional probability distribution generated by the common prior over joint types (Harsanyi, 1968).

The assumption of a common probability distribution over joint types is actually not a requirement for Bayesian games (see Sakovics (2001) for an example of work that does not make this assumption); however, it makes the games much easier to solve. Furthermore, for the common-payoff games for which this framework will be used, it is not an unreasonable assumption to make as all players have access to the same problem dynamics and are functioning as a team.

If $I = \{1, \ldots, n\}$ is the set of players, θ_i is a specific type of player i and Θ_i is the set of all possible types for player i such that $\theta_i \in \Theta_i$, then a joint type θ is $\theta_1, \ldots, \theta_n$ and the joint-type space is $\Theta = \Theta_1 \times \cdots \times \Theta_n$.[8] θ_{-i} is the type of all players but i, and Θ_{-i} is defined similarly. A probability distribution, $P \in \Delta(\Theta)$, over the joint-type space Θ is used to assign types to players, and this probability distribution is assumed to be commonly known. From this probability distribution, the marginal distribution over each player's type, $P_i \in \Delta(\Theta_i)$ can be recovered as can the conditional probability $P_i(\theta_{-i}|\theta_i)$.

The utility (or payoff) u of action a_i to a player is dependent on the actions selected by all other players as well as on their joint type, and is defined as $u_i(a_i, a_{-i}, (\theta_i, \theta_{-i}))$. By definition, a player's strategy π_i must define an action for every one of its possible types even though at run-time it will only be assigned one type. This constraint follows because the solution to the game is a set of best-response policies. In order for player i to know what its best response is to the other players, it must simultaneously solve for policies π_i and π_{-i}. Therefore, π_{-i} must state what players $-i$ will do for any one of their types because i does not know their specific types. In turn, players $-i$ are also simultaneously finding policies π_{-i} and π_i. Because they do not know what type player i is, π_i must define what i would do for each of its possible types. As the players are trying to find the same π_i and π_{-i}, each player's policy must therefore be conditional over her entire possible type space Θ_i. If a player's strategy is given by π_i, then the probability distribution over actions it assigns for its type θ_i is given by $P(a_i|\theta_i) = \pi_i(\theta_i)$.

Formally, a Bayesian game is a tuple (I, Θ, A, P, u) where $A = A_1 \times \cdots \times A_n$, $u = \{u_1, \ldots, u_n\}$ and I, Θ and P are as defined above. Given the commonly held prior $P(\Theta)$ over the distribution over players' types, each player can calculate a Bayesian-Nash equilibrium policy. A Bayesian-Nash equilibrium is a set of best response strategies, π, in which each player maximizes its expected utility conditioned on its probability distribution over the other players' types, and in which no player can do better by unilaterally changing its strategy. In this equilibrium, the policy π_i maximizes the expected value:

$$u_i(\pi_i, \theta_i, \pi_{-i}) = \sum_{\theta_{-i} \in \Theta_{-i}} P(\theta_{-i}|\theta_i) u_i(\pi_i(\theta_i), \pi_{-i}(\theta_{-i}), (\theta_i, \theta_{-i}))$$

for each $\theta_i \in \Theta_i$, given π_{-i}. Note that the conditional probability $P(\theta_{-i}|\theta_i)$ comes from

[8] In game-theory literature, a joint type is also referred to as a type profile. In this thesis, joint type is used because it is a more consistent with terms from multi-agent control (e.g., joint action, joint observation).

the marginals of the prior $P(\Theta)$. By using the marginals in this way, each player can determine, in expectation, the best action to take for each of its types without having to recurse infinitely on what type the other players are. Used in this way, the prior guarantees that each player will solve the same game while still being able to come up with conditional policies.

In general, one finds the set of Bayesian-Nash equilibria in the same way one finds the set of Nash equilibria for a normal-form game. It follows because the Bayesian game is represented as a normal-form game with a set of pure strategies that define what action to take for each possible type. However, like the exponential explosion in the conversion of an extensive-form game to a normal-form game, as the number of possible types increases, the number of pure strategies goes up exponentially.

In this thesis, Bayesian games are solved by transforming them into their sequence form. If player i has a set of types Θ_i and an action set A_i, then each type θ_i can be thought of as an information set. This transformation results in $|A_i||\Theta_i|$ possible sequences for each player. For common-payoff Bayesian games in which $u_i = u_j \; \forall i, j$, the alternating-maximization algorithm described in Section 2.1.5 can be applied to find the set of realization plans x that correspond to a locally optimal set of strategies π.

For common-payoff games, alternating maximization will find a set of locally optimal best-response policies that correspond to a Bayesian-Nash equilibrium. Multiple restarts must therefore be used to determine if there are other Bayesian-Nash equilibrium with higher payoffs to the team. The global optimum of this search problem is equivalent to the set of Pareto-optimal Bayesian-Nash equilibria. Alternating maximization as applied to this Bayesian game requires that the expected payoff of any combination of joint sequences be evaluated. If each agent has $|A_i||\Theta_i|$ sequences, then this algorithm has computational complexity with an upper bound of $O\big(|A_*|^n|\Theta_*|^n\big)$ where A_* and Θ_* are the largest action and type spaces of any player.

2.2.2 Example Bayesian Game

Consider a very simple 2-robot task-allocation problem. In this problem, there are two different types of tasks: simple and complex. A simple task can be completed by one robot while a complex task requires both robots. A task is generated with probability p, and with probability q it will be a simple task and with probability $1 - q$ a complex task. There are three possible tasks or world states: *no-task*, *simple-task* and *complex-task* with the prior

Table 2.1: Reward function for the 2-robot task allocation problem.

State	**Action**	**Reward**
*	*<do-nothing,do-nothing>*	0
no-task	*<do-nothing,do-task>*	$-f$
no-task	*<do-task,do-nothing>*	$-f$
no-task	*<do-task,do-task>*	$-2f$
simple-task	*<do-nothing,do-task>*	$s-f$
simple-task	*<do-task,do-nothing>*	$s-f$
simple-task	*<do-task,do-task>*	$s-2f$
complex-task	*<do-nothing,do-task>*	$-f$
complex-task	*<do-task,do-nothing>*	$-f$
complex-task	*<do-task,do-task>*	$c-2f$

Table 2.2: Emission probabilities for task signals.

State	**No-Signal**	**Simple-Signal**	**Complex-Signal**
no-task	1.0	0.0	0.0
simple-task	0.2	0.5	0.3
complex-task	0.2	0.3	0.5

probability of each occurring being $1-p$, pq and $p(1-q)$, respectively. Each robot must select between one of two actions: *do-nothing* and *do-task*. The reward function is given in Table 2.1. A robot incurs no costs if it does nothing, but has a fuel cost of f for doing a task. A simple task has a reward of s while a complex task has a reward of c. It is assumed that $c \gg 2f > s > f$. Finally, the robots do not know the actual task or world state, but they do each receive an independent (but correlated) observation of that state. There are three possible signals: *no-signal*, *simple-signal* and *complex-signal* with emission probabilities as shown in Table 2.2.

In this example, the private information received by each robot is its observation: $\Theta_i = \{$*no-signal*, *simple-signal*, *complex-signal*$\}$. Using the prior over the three world states and the emission probabilities of the observations, the set of conditional probabilities such as $P_i(\textit{no-signal}_{-i} | \textit{no-signal}_i)$ can be calculated, as can the probability distribution over Θ, which consists of nine possible joint types. These values are shown in Tables 2.3 (a) and (b), assuming a value of 0.3 for p and 0.5 for q.

Table 2.3: Common prior over joint types (a), the resulting conditional probabilities (b), and the utility function (c) for a 2-robot task allocation problem. These values were found using the emission probabilities given in Table 2.2 and the prior probability over world states with $p = 0.3$ and $q = 0.5$. In (b) each element is interpreted as $P_i(\text{row-entry}_{-i}|\text{column-entry}_i)$.

	no-signal$_2$	***simple-signal***$_2$	***complex-signal***$_2$
no-signal$_1$	0.328	0.056	0.056
simple-signal$_1$	0.056	0.119	0.105
complex-signal$_1$	0.056	0.105	0.119

(a) Common prior.

	no-signal$_i$	***simple-signal***$_i$	***complex-signal***$_i$
no-signal$_{-i}$	0.745	0.127	0.127
simple-signal$_{-i}$	0.2	0.425	0.375
complex-signal$_{-i}$	0.2	0.375	0.425

(b) Conditional probabilities.

Joint Type	**Joint Action**	
	<***do-nothing,do-task***> **or** <***do-task,do-nothing***>	<***do-task,do-task***>
(*no-signal,no-signal*)	$0.043s - f$	$0.043s + 0.043c - 2f$
(*no-signal,simple-signal*)	$0.625s - f$	$0.625s + 0.375c - 2f$
(*no-signal,complex-signal*)	$0.375s - f$	$0.375s + 0.625c - 2f$
(*simple-signal,no-signal*)	$0.625s - f$	$0.625s + 0.375c - 2f$
(*simple-signal,simple-signal*)	$0.735s - f$	$0.735s + 0.265c - 2f$
(*simple-signal,complex-signal*)	$0.5s - f$	$0.5s + 0.5c - 2f$
(*complex-signal,no-signal*)	$0.375s - f$	$0.375s + 0.625c - 2f$
(*complex-signal,simple-signal*)	$0.5s - f$	$0.5s + 0.5c - 2f$
(*complex-signal,complex-signal*)	$0.265s - f$	$0.265s + 0.735c - 2f$

(c) Utility function. The utility for performing <*do-nothing,do-nothing* >is always 0.0.

In this problem, the utility of taking a set of actions given a set of types is the expected reward of taking those actions given the belief over world states induced by those types. For example, if the joint type {*simple-signal*, *complex-signal*} leads to a probability distribution over the three world states of {0.0,0.5,0.5} then the utility of taking actions {*do-nothing*, *do-task*} is $0.5(s - f) + 0.5(-f) = 0.5s - f$. This is a common-payoff game and so $u_1 = u_2 = u$ with the complete payoff function tabulated in Table 2.3 (c).

The normal-form representation of this game has eight pure strategies per player. In order to simplify notation, DN and DT represent the actions *do-nothing* and *do-task* respectively and n,s and c represent the observations *no-signal*, *simple-signal* and *complex-signal*. An example pure strategy is then $DN_nDN_sDT_c$ which corresponds to a strategy of only performing the task if the player hears the *complex-signal*. As each entry r_{jk} in the normal form represents the expected reward of taking the pure strategies j and k, the calculation of each entry involves iterating over the eight possible combinations of observations and the resulting actions taken by the two players.

The sequence-form representation, however, has seven different sequences per player (six plus the null sequence $\emptyset$ as a placeholder used for the purpose of building constraints). An example sequence is $\emptyset DN_n$ which means the player will *do-nothing* if it hears the observation *no-signal*. If the payoff matrix B is tabulated for the sequence form, each entry b_{jk} requires the evaluation of taking sequences j and k: if the two sequences cannot occur together then the entry has value 0.0, otherwise it is the expected outcome of taking those actions given the associated signals (given by Table 2.3 (c)), weighted by the probability of those signals occurring (given by Table 2.3 (a)). The full payoff matrix can be seen in Table 2.4.

In order to solve for the locally optimal set of realization plans, the alternating-maximization algorithm is applied to matrix B using a set of random initial plans x and y. If values for f, s and c of 2.0,3.0 and 10.0 are substituted then, using random-restarts, the best plans are $x = \{\emptyset DN_n, \emptyset DT_s, \emptyset DT_c\}$ and $y = \{\emptyset DN_n, \emptyset DT_s, \emptyset DT_c\}$ with an expected value of 1.02 to the team (notation has been abused somewhat to only show the sequences with weights of 1.0). This result corresponds to a policy in which each robot will do the task so long as it receives the *simple-signal* or *complex-signal*.

Table 2.4: The payoff matrix B for the sequence form of the 2-robot task allocation problem. Player 1's sequences are shown as the rows and Player 2's as columns. Sequences for the same information set are grouped together. Because it is a common payoff game, the solution for each player requires that she selects one sequence per group.

	$\emptyset$	$\emptyset DN_n$	$\emptyset DT_n$	$\emptyset DN_s$	$\emptyset DT_s$	$\emptyset DN_c$	$\emptyset DT_c$
$\emptyset$	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\emptyset DN_n$	0.0	0.0	$0.014s - 0.328f$	0.0	$0.035s - 0.056f$	0.0	$0.021s - 0.056f$
$\emptyset DT_n$	0.0	$0.014s - 0.328f$	$0.014s + 0.014c - 0.656f$	$0.035s - 0.056f$	$0.035s + 0.021c - 0.112f$	$0.021s - 0.056f$	$0.021s + 0.035c - 0.112f$
$\emptyset DN_s$	0.0	0.0	$0.035s - 0.056f$	0.0	$0.087s - 0.119f$	0.0	$0.053s - 0.105f$
$\emptyset DT_s$	0.0	$0.035s - 0.056f$	$0.035s + 0.021c - 0.112f$	$0.087s - 0.119f$	$0.087s + 0.032c - 0.238f$	$0.053s - 0.105f$	$0.053s + 0.053c - 0.21f$
$\emptyset DN_c$	0.0	0.0	$0.021s - 0.056f$	0.0	$0.053s - 0.105f$	0.0	$0.032s - 0.119f$
$\emptyset DT_c$	0.0	$0.021s - 0.056f$	$0.021s + 0.035c - 0.112f$	$0.053s - 0.105f$	$0.053s + 0.053c - 0.21f$	$0.032s - 0.119f$	$0.032s + 0.087c - 0.238f$

2.3 Summary

This chapter has provided a summary of different representations and solution techniques used in game theory both for games of imperfect information and games of incomplete information. Because this information is intended to provide the reader with the necessary background for understanding BaGA, its focus has been on games with common payoffs.

There are two key concepts addressed in this chapter that are of particular importance to the rest of this dissertation and, therefore, should be understood before continuing. First, POSGs and extensive-form games are very similar, with the primary difference being that a POSG includes state in its model parameters while an extensive-form game has no notion of state. As a result, many of the issues that arise when finding policies for extensive-form games also occur in POSGs. In Section 2.1.5, the sequence form, an alternative representation of an extensive-form game, was presented and solution techniques discussed. Unlike the normal form, the sequence form is linear in the size of the game tree and not exponential. The space and time savings afforded by using the sequence form allows larger extensive-form games to be solved. In the following chapter, this sequence-form approach to solving extensive-form games will also be applied to small POSGs in order to find locally optimal policies.

The second key concept is that of Bayesian games. In Bayesian games, players have private information that is crucial to the decision-making process of other players. These games use a common prior to implicitly represent the infinite belief/knowledge hierarchy about private information and to find a set of best-response policies for all players. Because Bayesian games will be used as the building blocks of the BaGA algorithm presented in the following chapter, Section 2.2.1, which described Bayesian games, was very detailed and an example Bayesian game is presented and solved in Section 2.2.2.

Chapter 3

Approximate Solutions for POSGs with Common Payoffs

This chapter presents an algorithm for finding approximate solutions to a POSG with common payoffs that transforms the original problem into a sequence of smaller Bayesian games that are computationally tractable (Emery-Montemerlo et al., 2004). So long as each robot is able to build and solve the same sequence of games, the team can coordinate on action selection. This Bayesian game approximation algorithm (BaGA) allows one to handle finite horizon problems of indefinite length by interleaving planning and execution.

Before presenting the BaGA algorithm, it will be shown how the alternating-maximization algorithm of the previous chapter can be used to find locally optimal solutions to the full POSG. It is, however, only appropriate for very small problems due to its computational overhead.

3.1 Alternating-Maximization Approximation

In Section 1.3, a POSG was formally defined as a tuple (I, S, A, Z, T, R, O). Accompanying this definition was a decision-theoretic analysis for finding optimal solutions for POSGs with common payoffs; however, a POSG can also be analyzed from a game-theoretic view. While the POSG tuple parameters are a very compact representation, a POSG can be more intuitively thought of as an extensive-form game.[1] In this representation, a game tree is built

[1] As mentioned in Chapter 2, the major difference between extensive-form games and POSGs is that the latter includes state in its definition. This state information, however, can be folded into the game tree.

up in which robots sequentially make decisions about what actions to take and a player representing Nature is used to generate the different possible observation histories for each robot.

In this extensive-form representation of a POSG, identification of the different information sets is very important to properly represent what information is known to each robot at each of its decision nodes. If a robot has only access to its own observations of the global state and its own actions (i.e., no communication and no knowledge of the actions taken by other robots through observation of the reward function etc.), then at timestep t it has $|A_i|^{t-1}|Z_i|^t$ different possible information sets. Over the entire tree, it has $\sum_{t=1}^{T}|A_i|^{t-1}|Z_i|^t$ information sets, each of which has well defined relationships between itself and the other sets (such as which information sets are children of other information sets). Finding a solution to such an extensive-form game is not trivial: converting it to a normal-form game and then solving the resulting game would require as much computation as the exhaustive search over all possible joint policies described in Section 1.3. However, as discussed in Section 2.1.5, one can find a locally optimal solution to a common-payoff extensive-form game by converting it to the sequence form and applying alternating maximization.

Given the information sets of a POSG described above, each robot has $\sum_{t=1}^{T}|A_i|^t|Z_i|^t$ or $O\big((|A_i||Z_i|)^T\big)$ possible sequences. Defining a starting policy (i.e., realization plan x_i) for each robot takes exponential time as it requires a weight, $x_i(\sigma_u a)$, to be assigned to each possible sequence $\sigma_u a$ (i.e., $O\big(n(|A_i||Z_i|)^T\big)$). Note that this set of sequences includes all sub-sequences, and not just those of length T. Once this is done, alternating maximization requires that every joint sequence of length T be evaluated in order to find the best actions for robot i to take at the final timestep. Those values are then backed up using dynamic programming.

The expected value of a joint sequence is the sum of expected rewards for its associated joint actions at each timestep, multiplied by the probability of the joint sequence itself occurring (note that this definition does not take into account any future reward if the sequence is smaller than length T). Provided that the set of possible sequences of length t are constructed incrementally from the set of sequences of length $t-1$, then these expected rewards and probabilities can also be calculated incrementally. Given that it takes $O(|S|^2)$ to calculate these values for each joint sequence of length t, finding the expected value of all joint sequences of length T has an overall cost of $O(\big(|S|^2\big(|A_*||Z_*|\big)^{nT}\big)$. During the second, dynamic-programming, part of the alternating-maximization algorithm, constraints on information sets are used to evaluate the expected future value of sequences of length $t < T$. As a result the computation time required by one iteration of alternating maximization has two parts: the first is $O\big(|S|^2\big(|A_*||Z_*|\big)^{nT}\big)$, which allows the best sequences of length T to be found, while the second is $O\big((|A_*||Z_*|)^{t+1}\big)$, which backs the values at timestep $t+1$ up to timestep t.[2]

Assuming that a bounded number of iterations of policy evaluation is required to find a fixed-point set of best-response policies, then overall this algorithm is $O\left(|S|^2\big(|A_*||Z_*|\big)^{nT}\right)$ and, therefore, requires time exponential in the horizon and number of robots to find a locally optimal solution. This procedure is repeated a constant number of times with random starting policies in order to search the space of locally optimal solutions.

In contrast, the exhaustive search approach to solving POSGs presented in Section 1.3 is $O(|S|^2|Z_*|^{nT}|A_*|^{n|Z_*|^T})$ and therefore requires time doubly exponential in the planning horizon. This exhaustive search, however, is guaranteed to find the globally optimal solution rather than just a locally optimal one. It is possible to guarantee that the alternating-maximization approximation also finds the globally optimal solution by restarting it sufficiently many times. However, the resulting algorithm would become doubly exponential in the planning horizon because all combinations of possible starting policies would need to be considered.

3.2 Bayesian Game Approximation

In the previous section, locally optimal policies were found by using the sequence form and applying alternating maximization to the entire POSG at once. This use of the sequence form results in a large computational savings over exhaustive search; however, it is possible to define an approximation algorithm for POSGs that requires even less computational time by interleaving planning and execution.

When the POSG is thought of as a large tree, then an optimal path through the tree is found by finding the best action to take at the leaves and then backing up those values through the tree, taking care to obey the constraints of information: robots must make the same decision at all points in the tree between which they cannot distinguish. Once the values have been backed up through the tree, the optimal policy can then be found in a forward fashion by

[2] To find out which sub-sequence is best for each of the information sets at timestep t, the algorithm must evaluate each of the $(|A_*||Z_*|)^t$ sequences. The value of each sequence is its immediate value plus the expected value of each possible next sequence. In general, this means looking at the payoff generated by each of its $|A_*||Z_*|$ child sequences. Bookkeeping, however, can be used to look only at those child sequences that have positive probability of being played. This approach reduces the calculation of expected future payoff to $O\big(|Z_*|\big)$, and reduces the overall complexity of this calculation by a factor of $|A_*|$ to $O\big(|A_*|^t|Z_*|^{t+1}\big)$.

myopically selecting the best action for each information set. This action selection involves game-theoretic reasoning about the actions taken by the other members of the team.

This approach, however, whether done approximately through alternating maximization or exactly, requires all actions to be evaluated at the leaves of the tree and their values to be backed up even if an action choice at an early timestep means that huge parts of the tree will not be visited. That is, even for finding locally optimal solutions, a lot of computation time is spent evaluating the effects of actions for information sets that will ultimately never be reached.[3] If one could instead solve the game in a forward fashion, then computation time could be focused on only the relevant parts of the game tree. Of course, without loss of optimality, this would require a utility function that returns the exact value of the internal nodes of the tree which is impossible unless one has already solved the entire game tree. However, solutions that are locally optimal with respect to a given utility function can still be found.

This thesis proposes a Bayesian game approximation (BaGA) algorithm that generates a solution to a POSG by myopically selecting actions at each timestep using heuristic values for the internal nodes rather than their true values. Interleaving planning and execution, a Bayesian game is constructed for each timestep in the problem and solved, with the resulting policies, π^t, concatenated together to form an overall policy π for the problem. This transformation is similar to the classic one-step lookahead strategy for fully observable games (Russell & Norvig, 2002): the robots 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. A very similar approach, proposed by Shi & Littman (2001), is used to find near-optimal solutions for a scaled-down version of Texas Hold'Em.

In Figure 3.1 a high-level diagram of the BaGA algorithm is shown. If one considers the full POSG game tree to be represented by the large triangle, then the BaGA algorithm works by constructing a smaller game at each timestep that approximates the optimal policy. Using information common to the team, such as problem dynamics (e.g., T and O) and the policies found at previous timesteps, the robots are able to ignore parts of the tree that

[3] The degree to which a child (or grandchild etc.) information set affects the action taken at a parent information set depends on the ratio of the immediate expected reward that the action yields the parent, to any resulting future reward from the appropriate children. For many problems, there are parts of the game tree that can be safely ignored without worrying about their effect on parent information sets. For example, if a team of robots is operating in an environment with a clearly identifiable flight of stairs, then no policy should direct a member of the team to take an action that would lead it to fall down the stairs. All child information sets that would follow from such a decision can clearly be ignored during policy construction because they have essentially no effect on immediate expected reward: once a robot is dead, it is dead.

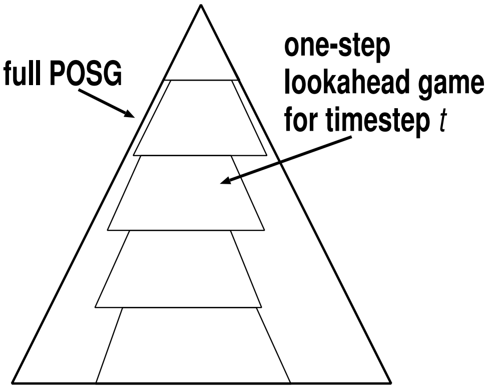


Figure 3.1: A high level representation of the BaGA algorithm for approximating POSGs. If all agents know every other agent's history of observations and actions at timestep t, then each one-step game would be represented as a triangle. As this is not the case, each one-step game is depicted as a trapezoid. Some pruning of the game tree, however, is possible because of common knowledge about the policies found at earlier timesteps.

would never be visited due to previous decisions. In this way, each game models a subset of the possible experiences occurring up to that point and then finds a one-step lookahead policy for each robot contingent on that subset.

The solution to a POSG is a set of policies that guarantee that the robots are fully coordinated during task execution. Each robot individually constructs the same game and solves it using the same algorithm resulting in the same set of policies for the entire team. In order for the BaGA algorithm to guarantee that robots will remain coordinated, it must ensure that each robot builds and solves the exact same sequence of Bayesian games. This constraint is especially important as information about previous policies is used to keep future games small: should robots not construct the same game at timestep t, then each will use a different set of policies $\pi^{i,t}$ to build a new game at timestep $t+1$, leading to a different game being constructed and further compounding future differences.

3.2.1 Conversion from a POSG to a Series of Bayesian Games

In order to model each timestep of the full POSG as a Bayesian game, the conversion between the two frameworks must be defined. In order to model a single timestep of the game it is necessary to be able to represent each sub-path through the tree up to that timestep as a single entity. A single sub-path through the tree corresponds to a specific set of observations and actions up to timestep t for all robots in the team. If all robots know that a specific path has occurred, then the problem becomes fully observable and the payoffs of taking each joint action at timestep t are known with certainty. This is analogous to how utility in Bayesian games is conditioned on specific joint types, which is why a Bayesian game, rather than a game of imperfect information, has been selected as the model for each smaller game.

Each path in the POSG of length t is represented as a specific joint type, θ^t, with the type of each robot, corresponding to its own observations and actions, as the type θ_i^t. A single θ_i^t may appear in multiple joint types. With this definition of the type space in the Bayesian game, robots are able to condition their policies on their individual observation and action histories as they do in the original POSG. There are, however, still two pieces of the Bayesian game model left to define before these policies can be found.

First, the Bayesian game model requires robots to have a common prior over the joint-type space Θ. If it is assumed that all robots have common knowledge of the starting conditions of the original POSG, i.e., a probability distribution $P(s^o)$ over possible starting states, then the algorithm can iteratively find both 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, π, is a set of policies for all robots, each robot not only knows its own next-step policy but also those of its teammates, and this information can be used to update the joint-type space. That is, any type θ_i^t that has robot i perform a different action at timestep $t-1$ than the action given by the policy π_i^{t-1} can be removed from Θ_i^t because it has zero probability of occurring. Because the set of all possible histories at timestep t can be quite large, the initial BaGA algorithm discards all joint types (or joint histories) with *prior probability* less than some threshold and then renormalizes $P(\Theta^t)$. By using the prior probability, it can be guaranteed that all robots discard the same set of joint histories.

The final piece of the Bayesian game model that must be defined is that of the utility function u that represents the payoff of actions conditioned on the joint-type space. Ideally, a utility function should represent not only the immediate value of a joint action but also its expected future value because, in general, POSGs are used for multi-step problems. In finite horizon MDPs and POMDPs, these future values are found by backing up the value of actions at the final timestep through time. In BaGA, finding these values corresponds 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 timestep $T-1$, solving that Bayesian game and so-on until the first timestep is reached. Unfortunately, this approach is intractable: it requires BaGA to do as much work as solving the original POSG because a specific probability distribution over Θ^T is not known until the game is solved for timesteps 0 through

$T-1$. In addition, unlike the proposed algorithm that interleaves planning and execution, no advantage could be taken of any reduction in the size of Θ^t that would arise through applying common knowledge from π^{t-1}. Instead, BaGA performs a one-step lookahead using a heuristic function for *u*, resulting in policies that are locally optimal with respect to that heuristic. BaGA uses heuristic functions that try to capture some notion of future value while remaining tractable to calculate. Those heuristics are domain-dependent.

Now that all parameters of the Bayesian game have been defined, the robots 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 Bayesian-Nash equilibrium is found using alternating maximization.[4] Once a set of best-response strategies is found at timestep t, each robot i matches its true history of observations and actions, h_i^t, to one of the types in its type space Θ_i^t and then selects an action to execute based on its policy π_i^t and type θ_i^t.[5]

BaGA runs in parallel on all robots in the team as shown in Figure 3.2. So long as a mechanism exists to ensure that each robot finds the same set of best-response policies π^t, then each robot will maintain the same joint-type space for the team as well as the same probability distribution over that joint-type space without having to communicate. This condition ensures that the common prior assumption of the Bayesian game model is always met. Shading is used in Figure 3.2 to indicate which steps of the BaGA algorithm should result in the same output for each robot. Robots run the algorithm in lock-step and a synchronized random-number generator is used to ensure that all agents come up with the same set of best-response policies using the alternating-maximization algorithm. It is assumed that the robots have some sort of locker-room agreement that provides this synchronization. Locker-room agreements can be used to eliminate or reduce the need for communication by specifying how agents will act in a certain scenario (Stone & Veloso, 1999). For example, if two joint actions in a decentralized MDP have the exact same value, a locker-room agreement would specify which of the two actions should be taken (e.g., select the action that occurs first in an agreed ordering). In this case, the locker-room agreement specifies either the pseudo-random number generator seed or sequence of random numbers to be used for any randomization calls in BaGA.

Only common knowledge arising from the policies for previous timesteps or domain specific information (e.g., robots might always know the position of their teammates from a

[4] Note that this instance of alternating maximization is not the same as the alternating-maximization approximation of Section 3.1, which was applied to the entire POSG at once. This instance of alternating maximization is applied only to the Bayesian game representation of timestep t.

[5] If the robot's true history has been pruned, then the actual history is mapped to the nearest non-pruned history using Hamming distance. A better method of mapping will be defined in Chapter 4.

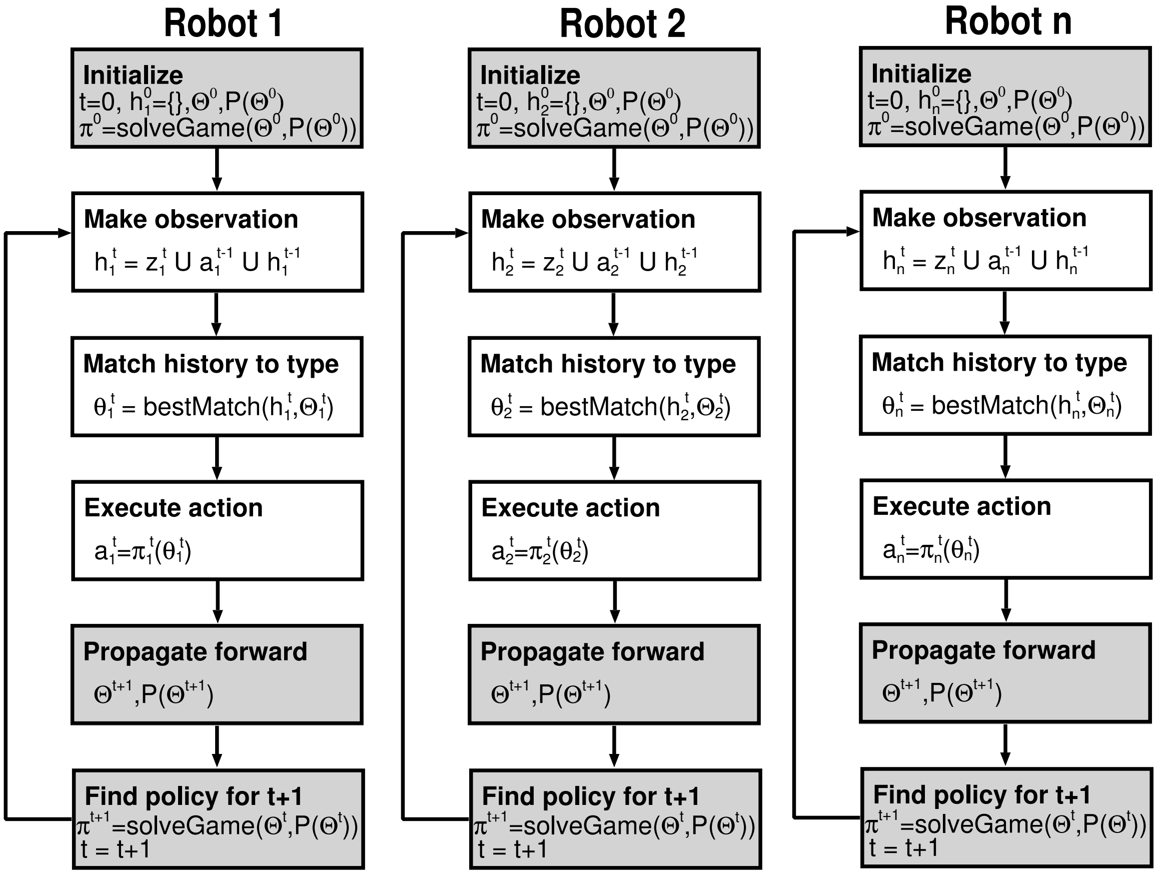

Robot 1
Initialize
t=0, h_1^0={},Θ^0,P(Θ^0)
π^0=solveGame(Θ^0,P(Θ^0))
Make observation
$h_1^t = z_1^t \cup a_1^{t-1} \cup h_1^{t-1}$
Match history to type
θ_1^t = bestMatch(h_1^t,Θ_1^t)
Execute action
$a_1^t=\pi_1^t(\theta_1^t)$
Propagate forward
Θ^{t+1},P(Θ^{t+1})
Find policy for t+1
π^{t+1}=solveGame(Θ^t,P(Θ^t))
t = t+1
Robot 2
Initialize
t=0, h_2^0={},Θ^0,P(Θ^0)
π^0=solveGame(Θ^0,P(Θ^0))
Make observation
$h_2^t = z_2^t \cup a_2^{t-1} \cup h_2^{t-1}$
Match history to type
θ_2^t = bestMatch(h_2^t,Θ_2^t)
Execute action
$a_2^t=\pi_2^t(\theta_2^t)$
Propagate forward
Θ^{t+1},P(Θ^{t+1})
Find policy for t+1
π^{t+1}=solveGame(Θ^t,P(Θ^t))
t = t+1
Robot n
Initialize
t=0, h_n^0={},Θ^0,P(Θ^0)
π^0=solveGame(Θ^0,P(Θ^0))
Make observation
$h_n^t = z_n^t \cup a_n^{t-1} \cup h_n^{t-1}$
Match history to type
θ_n^t = bestMatch(h_n^t,Θ_n^t)
Execute action
$a_n^t=\pi_n^t(\theta_n^t)$
Propagate forward
Θ^{t+1},P(Θ^{t+1})
Find policy for t+1
π^{t+1}=solveGame(Θ^t,P(Θ^t))
t = t+1

Figure 3.2: A diagram showing the parallel nature of BaGA. Shaded boxes indicate steps that have the same outcome for each robot.

common observation signal) is used to update individual- and joint-type spaces. This includes pruning of types due to low probability because pruning thresholds are common to the team. The shaded boxes of Figure 3.2 indicate steps in which robots will have the same outcome: they all solve the same Bayesian game and find the same policy π^{t+1}. The non-shaded boxes of Figure 3.2 indicate steps in which robots will have different outcomes: each robot will match its true observation and action history, h_i^t, to a type modeled in the game and then execute the action given by that type, $a_i^t = \pi_i^t(\theta_i^t)$.

3.2.2 Implementation Issues

In practice there are two possible difficulties with using the BaGA algorithm for approximating the full POSG. The first is that the joint-type space Θ^t can be extremely large, and the second is that it can be difficult to find a good heuristic for *u*. A system designer can exert some control over the size of the joint-type space through the selection of the probability threshold for pruning unlikely histories. The benefit of using such a large type space is that it can be guaranteed that each robot has sufficient information to independently construct the same set of joint histories given by Θ^t and the same probability distribution over Θ^t at each timestep t. In the following chapters, two additional approaches are described for reducing the size of Θ^t while still doing a good job of representing the true distribution over joint histories.

Finding a good utility function u that allows to Bayesian game to find high-quality actions is more difficult because it depends on the system designer's domain knowledge. For real robot problems, good results have been achieved with a Q_{MDP} heuristic (Littman et al., 1995). Q_{MDP} is a very simple to calculate POMDP approximation. In the Q_{MDP} utility function, $u(a, \theta)$ 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 sufficient to find Q-values for a fully observable version of the problem. These Q-values can be found by dynamic programming or Q-learning with either exact or approximate Q-values, depending upon the nature and size of the problem.

Perhaps a better utility function is one that is based on the POMDP solution of a version of the problem in which the robots can share all observations. That is, instead of assuming that the problem becomes fully observable in the next timestep, it is assumed that the robots have full communication on the next step while retaining the partially observable quality of the world. However, even the smallest of the robot problems examined in this thesis are too large to solve using approximate POMDP techniques such as those of Pineau et al. (2003a). Section 3.4.1.2, however, presents experimental results comparing a variety of different heuristics for a small problem in which the POMDP solution could be found for the joint-action and -state spaces. Hand-coded heuristics are compared with those that make use of a POMDP solution that assumes the robots will be able to share all observations after the current timestep.

3.3 Algorithms for the Bayesian Game Approximation

Algorithms showing how to implement BaGA are given in Algorithms 3.1 and 3.2. Algorithm 3.1 takes the parameters of the original POSG (the initial joint-type space is generated using the initial distribution over S) and builds up a series of one-step policies, π^t. In parallel, each agent will: solve the Bayesian game for timestep t to find π^t; match its current history of experiences to a specific type θ_i^t in its type space; match this type to an action as given by its policy π_i^t; and then execute that action. Algorithm 3.2 provides an implementation of how to solve a Bayesian game given the dynamics of the overall POSG and a specific joint-type space and distribution over that space. It also propagates the joint-type space and its prior probability forward one timestep based on the the solution to that game. Note that the quantity $P(z, a|\theta)$ in Algorithm 3.2 is the probability of an observation z and an action a occurring as the one-step extension to the type θ and will therefore depend both on problem dynamics as well as the policy just found for the team.

Additional variables used in these algorithms are: r is the reward to the team, rs is a random seed, σ_i is a generic strategy, ϕ a generic type, and superscripts refer to the timestep under consideration. The function *beliefState*(θ) returns the belief state over S given by the joint history of observations and actions defined by the joint type θ. In Algorithm 3.2, a Q_{MDP} heuristic for calculating utility is used, but any other heuristic could be substituted.

For alternating maximization, shown in its dynamic-programming form in Algorithm 3.3, 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, BaGA synchronizes the random number generation of each agent. For each Bayesian game, alternating maximization is applied for only a single timestep and so, unlike Section 3.1, it is not necessary to back up values from timestep $t + 1$ to t.

Algorithm 3.1: $PolicyConstructionAndExecution$

```
$I, \Theta^0, A, P(\Theta^0), Z, S, T, R, O$
Output: $r, s^t, \pi^t, \forall t$
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)
            $setRandSeed(rs^t)$
            $\pi^t, \Theta^{t+1}, P(\Theta^{t+1}) \leftarrow BayesianGame(I, \Theta^t, A, P(\Theta^t), Z, S, T, R, O, rs^t)$
            $h_i \leftarrow h_i \cup z_i^t \cup a_i^{t-1}$
            $\theta_i^t \leftarrow matchToType(h_i, \Theta_i^t)$
            $a_i^t \leftarrow \pi_i^t(\theta_i^t)$
        /*all agents execute their action $a_i^t$ */
        $s^{t+1} \leftarrow T(s^t, a_1^t, ..., a_n^t)$
        $r \leftarrow r + R(s^t, a_1^t, ..., a_n^t)$
end
```

3.3.1 Computational Complexity

For a problem in which the type space of a robot consists of its history of observations of the global state and actions up to timestep t (i.e., robots cannot observe the actions of their teammates), then the size of that type space, $|\Theta_i^t|$, would be $|A_i|^{t-1}|Z_i|^t$ if no pruning of the game tree was done. BaGA, however, does prune away parts of the game tree that all agents know will never be visited in the future due to previous policy decisions. Basically, because BaGA ensures that policies are deterministic, for all types $\theta_i^{t-1} \in \Theta_i^{t-1}$, only those one-step extensions of type θ_i^{t-1} that include the action given by $\pi_i^{t-1}(\theta_i^{t-1})$ will be included in the type space at timestep t. This constraint makes the size of its type space have an upper bound of $|\Theta_i^t| \leq |Z_i|^t$ as further pruning may be possible due to problem constraints or other common knowledge such as a pruning threshold.

Each timestep of the BaGA algorithm requires a Bayesian game to be solved, which, using the sequence form and alternating maximization, has a computational complexity of $O\big((|A_*||\Theta_*^t|)^n\big)$. Due to pruning, this complexity has an upper bound of $O\big(|A_*|^n|Z_*|^{nt}\big)$. Solving this game, however, also requires that the utility function for each $\{\theta^t, a\}$ pair to be defined. With a heuristic like Q_{MDP}, this procedure requires $|S|$ operations per joint sequence, leading to an overall complexity of $O\big(|S||A_*|^n|Z_*|^{nt}\big)$. Finally, for each timestep of the algorithm $t \neq T$, the joint-type space and the probability distribution over it must be generated for $t+1$. Generating the new joint-type space requires iterating over all possible $(|A_*||Z_*|)^n$ one-step extensions of each Θ^{t-1} and calculating the probability of such an

Algorithm 3.2: $BayesianGame$

```
Input: $I, \Theta, A, P(\Theta), Z, S, T, R, O, randSeed$
Output: $\pi, \Theta', P(\Theta')$
begin
    $setSeed(randSeed)$
    for $a \in A, \theta \in \Theta$ do
        $u(a, \theta) \leftarrow qmdpValue(a, beliefState(\theta))$
    $\pi \leftarrow findPolicies(I, \Theta, A, P(\Theta), u)$
    $\Theta' \longleftarrow \emptyset$
    $\Theta'_i \longleftarrow \emptyset, \forall i \in I$
    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 3.3: $findPolicies$

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

extension requires $O(|S|^2)$ operations.

If the horizon of the problem is T, then altogether the BaGA algorithm has an upper bound on complexity of $O\big(|S|^2|A_*|^n|Z_*|^{nT}\big)$, which is at least a factor of $A^{n(T-1)}$ less than that of performing alternating maximization on the entire POSG at once. In practice, this factor can be even larger due to any further reductions in the size of the type space.

It is important to note that for many problems, the size of the global state space, $|S|$, actually serves as a fairly loose upper bound in these calculations due to problem constraints. For example, consider a robotic tag problem in which the type of each robot specifies its actions, position, and any observations of its teammates and the opponent. I.e., Z_i includes the position of robot i. Rather than iterate over all states in S, for each joint type θ^t, only those elements of S that are consistent with the associated robot positions at timestep t need to be considered when determining the probability and utility of θ^t. If S is the cross-product of the robots' positions (*RT*) and the opponent's position (*Opp*), i.e., $|S| = |RT|^n|Opp|$, then the upper bound on the complexity of the problem would actually be $O\big(|Opp|^2|A_*|^n|Z_*|^{nT}\big)$, which is much smaller than $O\big(|S|^2|A_*|^n|Z_*|^{nT}\big)$.

3.3.2 BaGA for POSGs with Limited Communication

While POSGs can model problems with any degree of communication, in this chapter, and the one following it, the BaGA algorithm is developed for solving POSGs without communication. The addition of periodic communication can only improve the performance of the team as it would allow robots to share their specific information histories and therefore make decisions conditioned on the true joint history of the team rather than a probability distribution over the set of possible joint histories. In this way, POSGs with no communication provide a lower bound on the performance of a robot team while those with full communication, modeled as POMDPs for a larger meta-agent, provide an upper bound. A POSG with arbitrary, but limited communication, will have performance somewhere between the two.

If a POSG has a fixed communication policy in which robots share their observations at periodic intervals then the BaGA algorithm can be modified to incorporate this communication by treating the communicated information as common knowledge that is used to affect the set of joint types Θ^t and the probability distribution over them $P(\Theta^t)$. For example, if the team communicates every x timesteps, then for $t = cx$, the set of joint types would contain just one element, the true joint history of the team θ^t, and this element would have probability of 1.0 of occurring.[6]

The generation of dynamic communication policies, in which robots make decisions to communicate based on specific observation histories, is a much harder problem and will be examined in more detail in Chapter 5.

3.3.3 Levels of Approximation

In BaGA, there are two main levels of approximation used to find the solution to the original POSG. First, alternating maximization is used to find a locally optimal Bayesian-Nash equilibrium of the one-step game representing the current timestep. This equilibrium is not guaranteed to be the Pareto-optimal Bayesian-Nash equilibrium because only a limited number of random restarts are used to move out of local maxima. Unless one were to exhaustively search the space of all policies, one cannot guarantee that the Pareto-optimal equilibrium has been found, nor place useful bounds on how much worse the expected payoff of a locally optimal policy is than that of the Pareto-optimal policy. Worst-case bounds could be found by comparing the optimal joint action given full observability to the expected value of the found equilibrium point; however, in practice, the Pareto-optimal Nash equilibrium would not be able to achieve this fully observable value due to partial observability.

If the expected future value of actions, given the current timestep, were known with certainty, then using alternating maximization to find the one-step policy would be the only place in which approximation was introduced into the algorithm; however, as this function is not known (it would involve solving the original problem), further approximation is introduced through the use of heuristics to estimate this function. Again, error bounds on performance introduced through this second level of approximation cannot, in general, be found. Assumptions can be made on the problem domain that render the heuristics correct, such as robots being able to share all observations on the following timestep. An example would be problems with some sort of delayed communication; robots have full communication but the communicated information takes one timestep to reach the rest of the team. In practice, however, these assumptions are not valid in many interesting problems.

The effects of these two levels of approximation can be mitigated at the expense of greater computation. First, alternating maximization can be replaced with an algorithm that has

[6] If robots can only share partial information then this information is simply treated as common knowledge and used to update the common prior over the set of possible joint types as appropriate.

guarantees on the global optimality of the found policies, such as an exhaustive search method that would iterate over all possible joint policies. For common-payoff games, moving from a (Bayesian) Nash equilibrium to a correlated equilibrium solution concept does not reduce the complexity of this search. Although a correlated equilibrium that maximizes the payoffs to all robots can be found in polynomial time through linear programming, constructing the constraints of this linear program still involves enumerating over all possible joint policies.[7]

The effect of the second level of approximation is affected by the quality of the heuristic used. Improving the quality of the heuristic can involve more off-line computation, such as solving smaller horizon versions of a problem in order to estimate the future value of actions rather than relying on a Q_{MDP} heuristic. Regardless of heuristic quality, BaGA is effectively doing a one-step lookahead in order to evaluate the effects of taking an action. Using the same heuristic, solution quality could be improved, at an increase in on-line computational costs, by moving to multi-step lookahead.

In multi-step lookahead, rather than doing game-theoretic reasoning over only the current action choice, robots would find locally optimal policies of length n. Given a type space equal to the set of all possible joint histories at timestep t, robots would consider sequences of actions for the next n timesteps as the building blocks of their policies rather than just single actions. These sequences can best be thought of as n-step policy trees: the root of the tree gives the immediate action for a robot to take while action choices for the next $n - 1$ steps are conditioned on possible future observations. The expected value of such a policy tree would be the expected sum of the immediate rewards over the n steps plus the expected future value of the last action (leaf of the tree) as given by a heuristic. A policy π^t now tells robots which policy tree to follow for each of their possible types at timestep t. To select an action, a robot uses π_i^t to find the policy tree which is assigned to its (true) type θ_i^t and then implements the action given by the root of that policy tree. At timestep $t + 1$, robots construct a new game to find a new policy of length n rather than re-using any of the previously found policy trees because the expected value of the new policy tree provides a more accurate estimate of the expected payoffs of actions at timestep $t + 1$. This approach, sometime called receding-horizon control, reduces the negative effects of approximating

[7] It actually requires enumeration over all joint strategies where a strategy is an assignment of an action to each possible type. For general-sum problems, a strategy is deterministic whereas a policy is not and so the set of all joint policies is much larger than the set of joint strategies. For a common-payoff problem, however, there is no need to randomize over actions and so only deterministic policies are considered. In this case, policies and strategies can be thought of as equivalent and so the set of all possible joint policies is equal in size to the set of all joint strategies.

the future value of actions as $n \to T$, the horizon of the problem.

In addition to these two main kinds of approximation, the pruning of low-probability joint histories (i.e., joint types) results in additional approximation because all possible joint histories are no longer represented by BaGA. However, unlike the other two levels of approximation, the effects of low-probability pruning can be easily negated by reducing the pruning threshold to zero. Alternatively, more principled methods for reducing the number of joint histories that are maintained by BaGA can be applied. These methods are the focus of Chapter 4.

3.3.4 Sub-Game Perfect Equilibria

In Section 2.1.7, the notion of a sub-game perfect equilibrium was introduced. In sub-game perfect equilibria, the strategies of robots are guaranteed to best responses to each other at all parts of the tree, even in those sub-games that can not be reached due to earlier decisions.[8] While alternating maximization is able to find a locally optimal equilibria for the full POSG, in general, it is not able to find a sub-game perfect equilibria. This property results from the combination of using the sequence form and dynamic programming to solve the problem. In the sequence form, the expected value of a joint sequence that has zero probability of occurring is 0.0. Therefore, during a dynamic-programming back-up, the algorithm cannot distinguish between a joint sequence that has zero probability of occurring but would result in optimal reward if it could occur and one that also has zero probability but has a lower reward. As a result, it will select between the two in an arbitrary fashion. Even if the dynamic-programming algorithm were to result in a set of strategies that are a sub-game perfect equilibrium (due to how random choices were made), those strategies will have the same expected value as any other set of strategies that are the same for the reachable portions of the tree (but different for the unreachable parts). The algorithm therefore, cannot identify which of the two sets of strategies is superior.

The BaGA algorithm also does not find sub-game perfect equilibria. By definition, a sub-game perfect equilibrium will define a set of best-response strategies for every sub-game of the tree. But, BaGA only generates strategies for reachable sub-games of the tree. Indeed,

[8] The only reason that an equilibrium point for a multi-stage game would not satisfy sub-game perfection would be if it was found by solving a normal or sequence-form version of the original game. As explained in Chapter 2, in these two forms a sub-game perfect equilibrium policy and a policy that is identical for reachable sub-games of the tree but different for non-reachable sub-games, have identical evaluations and so both satisfy the the equations for finding best-response policies. Differences between the two policies would therefore be apparent only to a human and not a machine.

part of its computational savings results from pruning out parts of the tree that are commonly known to be unreachable. Therefore, because it does not define actions to take for unreachable sub-games, it cannot, by definition, find sub-game perfect equilibria. Furthermore, because of that pruning, if something were to occur that would make an unreachable sub-game of the tree suddenly possible, BaGA would not be able to properly represent the problem. However, for the type of problems examined in this thesis, the inability of BaGA to find sub-game perfect equilibria is not important because unreachable portions of the game tree remain unreachable. If there is the possibility of noisy actions (e.g., in a real robot problem), then that noise is taken into account in the problem dynamics.[9] This property allows BaGA to retain the appropriate joint types and still generate valid solutions.

3.4 Experimental Results

In this section, experimental results are presented for three different domains. The first is that of the *Lady and the Tiger*, which is a 2-agent problem for which the full POSG can be solved for small time horizons. The solution quality and computational complexity of BaGA is compared to that of the full POSG. Additionally, the effects of different utility functions on the performance of BaGA is analyzed. The second domain looked at is that of the *Multiple Access Broadcast Channel*. Like the *Lady and the Tiger*, the full POSG of this n-agent problem can be solved for small time horizons. It is included to demonstrate how BaGA handles problems with more than two agents. The final domain is an abstract 2-robot *Tag* problem and is much larger in size than the previous two problems. The full POSG cannot be constructed for *Robotic Tag* and so, instead, the performance of BaGA is compared to that of well known heuristics.

3.4.1 Lady and The Tiger

The *Lady and the Tiger* problem is a multi-agent version of the classic tiger problem (Kaelbling et al., 1998) created by Nair et al. (2003). In this two-state, finite-horizon problem, two agents are faced with the problem of opening one of two doors. One door has a tiger behind it, and the other a treasure. Whenever a door is opened, the location of the tiger is randomly reset and the game continues until the fixed time has passed. The crux of the

[9] This is somewhat related to the notion of "trembling hands" which can be used to resolve the sub-game perfect equilibrium problem (Selten, 1975). However, applying "trembling hands" unilaterally to BaGA would result in greatly reduced pruning and so noisy action execution is only modelled as necessary.

Lady and The Tiger problem is that neither agent can see the actions or observations made by its teammate, nor can it observe the reward signal and thereby deduce that the game has been reset. It is, however, necessary for each agent to reason about this information.

The set of states is $S = \{$*tiger-left*, *tiger-right*$\}$, which indicates whether or not the tiger is behind the left or right door. The initial distribution over S is $\{0.5, 0.5\}$. Each agent has three actions, $A_i = \{$*listen*, *open-left, open-right*$\}$, and two possible observations, $Z_i = \{$*hear-left*, *hear-right*$\}$. The transition, reward and observation emissions functions, T, R and O are given in Table 3.1. The reward function is such that it is always better for the agents to select the same action; a POMDP formulation of the problem in which agents can share their observations leads to only the actions <*listen*, *listen*>, <*open-left*, *open-left*> and <*open-right*, *open-right*> being used (i.e., the remaining joint actions are dominated). If agents cannot communicate, then they must reason about what action is best to take given what observations and actions their teammate may have taken. For example, if a door is opened by Agent 1 at timestep t, then the problem is reset, drastically changing the emission probabilities for Agent 2. If Agent 2 does not reason about the possibility that a door was opened, it will place too high a weight on its observation at timestep t during future decision making.

3.4.1.1 Comparison to Full POSG

While the *Lady and the Tiger* problem is small, it allows the policies constructed by building the full extensive-form game version of the POSG to be compared to those from the BaGA approximation. It also shows how even a very small POSG quickly becomes intractable. Table 3.2 compares performance and computational time for this problem with various time horizons. The full POSG version of the *Lady and the Tiger* was solved using the dynamic-programming version of alternating maximization on the sequence form with 20 random restarts. Increasing the number of restarts did not result in higher valued policies.

For the BaGA algorithm, each agent's type at timestep t consists of its entire observation and action history up to that point. For example, a possible type at timestep 4 is <*listen*, *hear-left*, *listen*, *hear-right*, *listen*, *hear-right*>. Each joint type θ completely defines a belief b_θ over world state. Joint types with probability less than 0.000005 are pruned during the construction of the type space Θ^t and the probability distribution over the joint-type space renormalized. If an agent's true history is pruned, then for action selection it is assigned the closest matched type in Θ_i^t. This match is found using Hamming distance.

Table 3.1: Transition, reward and observation emission functions for the *Lady and the Tiger* problem. All functions are symmetric with respect to the agents and so only half of the joint actions have been shown (e.g., <*open-left*, *open-right*> and <*open-right*, *open-left*> result in the same outcome). Because opening the door resets the tiger's position, useful observations are only made if both agents perform the *listen* action at the same time.

State	**Action**	***tiger-left***	***tiger-right***
tiger-left	<*listen*, *listen*>	1.0	0.0
tiger-right	<*listen*, *listen*>	0.0	1.0
*	<*open-left*, *>	0.5	0.5
*	<*open-right*, *>	0.5	0.5

(a) T

State	**Action**	**Reward**
*	<*listen*, *listen*>	-2.0
tiger-left	<*open-left*, *listen*>	-101.0
tiger-left	<*open-right*, *listen*>	9.0
tiger-left	<*open-left*, *open-left*>	-50.0
tiger-left	<*open-left*, *open-right*>	-100.0
tiger-left	<*open-right*, *open-right*>	20.0
tiger-right	<*open-left*, *listen*>	9.0
tiger-right	<*open-right*, *listen*>	-101.0
tiger-right	<*open-left*, *open-left*>	20.0
tiger-right	<*open-left*, *open-right*>	-100.0
tiger-right	<*open-right*, *open-right*>	-50.0

(b) R

State	**Action**	***hear-left***	***hear-right***
tiger-left	<*listen*, *listen*>	0.85	0.15
tiger-right	<*listen*, *listen*>	0.15	0.85
*	<*open-left*, *>	0.5	0.5
*	<*open-right*, *>	0.5	0.5

(c) O

Table 3.2: Computational and performance results for the *Lady and the Tiger*. Expected reward and computation time is shown for the full POSG solution. For the BaGA algorithm, rewards are averaged over 100000 trials with 95% confidence intervals shown. Timing experiments were performed on a Intel Pentium 4, 2.80 GHz machine. Memory, rather than computation requirements, prevented the full POSG from being solved for the horizon 7 problem.

Time Horizon	**Full POSG**		**BaGA**	
	Expected Reward	**Time(ms)**	**Average Reward**	**Time(ms)**
3	5.19	50	5.18 ± 0.15	1
4	4.80	1000	4.77 ± 0.07	5
5	7.02	25000	7.10 ± 0.12	20
6	10.38	900000	10.28 ± 0.21	50
7	–	–	10.00 ± 0.17	200
8	–	–	12.25 ± 0.19	700
9	–	–	15.28 ± 0.26	3500
10	–	–	15.07 ± 0.23	9000

At each timestep alternating maximization with 20 random restarts is used for policy construction. The heuristic used to calculate the utility of actions for the results shown in Table 3.2 was experimentally hand-tuned to maximize performance and makes use of the performance of policies found for shorter horizons. This heuristic is actually a collection of heuristics with the one that maximized performance being used for each horizon. In the following section, the effects of specific heuristics on performance is investigated more closely.

For those timesteps for which the full POSG could be solved, BaGA was able to find policies that resulted in similar performance but at a fraction of the computation time. For those timesteps in which the full POSG was not solved, the overall performance of BaGA could not be directly evaluated. One could come up with an estimation of the expected performance of the full POSG by splitting up a problem with a horizon of n into two or more smaller horizon problems and then applying their policies consecutively. For example, a problem with a horizon of 9 could be considered as three consecutive instances of the horizon 3 problem or as an instance of the horizon 4 problem followed by a horizon 5 problem. Policies that concatenate together smaller policies in this way can be represented by the chosen type space, and therefore can make up part of valid solutions to this problem.

For example, a policy calculated for a horizon of 6 is shown in Figure 3.3 (a). The equilibrium point with highest expected value found for both the full POSG and the BaGA with the complex heuristic is made up of an identical policy for each agent that can be described by the policy tree in Figure 3.3 (a). In this policy each agent performs the *listen* action twice, and then, if it receives the same observation twice in a row it opens the appropriate door, and otherwise it chooses *listen* again. Regardless of what action was selected on the third timestep, the agents *listen* on the fourth and fifth timesteps. Then, using only observations gathered on those two timesteps, each agent will open the door if its last two observations are the same. In other words, even if the agent does not open a door on the third timestep, it still ignores the observations it receives on that timestep because the event that at least one agent has opened a door has high enough probability to make this a bad observation upon which to condition decisions. Because the agents ignore this information, one might interpret this overall policy as the agents following their horizon 3 policy tree for three steps and then, on the fourth timestep, simply re-starting at the root of this policy tree.

Using a less complex heuristic, such as some of those discussed in the next section, a different local optimum was found in which each agent listens for the first five timesteps and then, if it hears four or five of the same observations, opens the appropriate door. This policy is shown as a policy tree in Figure 3.3 (b). This policy illustrates one of the downfalls of using a heuristic to evaluate the future value of actions. In this case, the heuristic used is overly optimistic because agents assume they will be able to share all observations in the future. Therefore, when making a decision at timestep 3, agents believe they will be able put off opening the door for another timestep (in order to make a more informed decision about which door to open), but still have enough time to get the observations needed to open a door on the final timestep. However, once the fourth timestep is reached, the agents have a better estimate of the fact that opening a door now and then performing a *listen* on the fifth timestep will not yield enough information to open a door on the final step. As a result they do not open the door on the fourth or fifth timesteps either. With a better heuristic, agents are able to evaluate the benefits of opening a door twice, each time on the basis of less information (and therefore with lower expected reward), against opening the door only once but with higher quality information.

BaGA required much less time to arrive at the 6-step policy than performing alternating maximization over the full POSG because it had to keep track of a fraction of the number of observation histories. In the full POSG, there were over a thousand information sets in its tree representation. Thus, the evaluation of the leaves (all policies of length 6) required evaluating roughly nine million joint sequences before that information could be backed up to earlier timesteps. In contrast, BaGA resulted in type spaces with size of $|\Theta_i^t|$ $=\{1,2,4,8,16,32\}$ for each agent meaning that the most expensive alternating-maximization step, which was on the final timestep, only had to evaluate 1024 joint sequences.

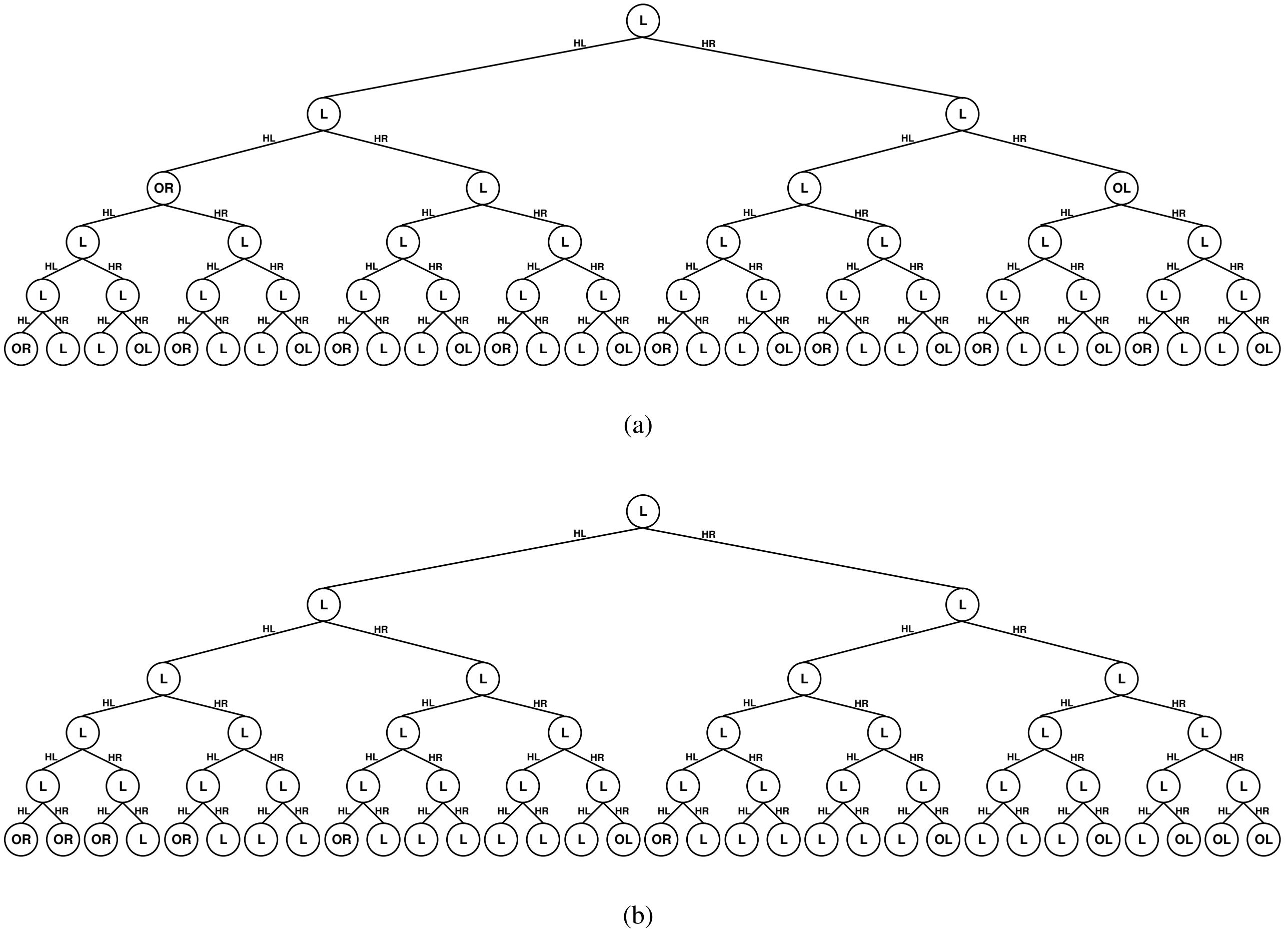
As the horizon of the problem increases, the pruning threshold of 0.000005 starts to reduce the number of joint types, which in turn reduces the number of individual types for each agent. For example, while the upper limit on the type space of a 10-step problem is $|\Theta_i^t| = 2^{t-1}$, in practice the sizes of the type spaces were $|\Theta_i^t|$ $=\{1,2,4,8,16,32,52,104,208,416\}$ which helped to keep the problem manageable.

3.4.1.2 Heuristic Function Evaluation

In order to evaluate the effects of the heuristic function upon performance, four different utility functions were used in the BaGA algorithm. A *selfish* version of each of these heuristics was also implemented in order to judge how performance is improved by doing a full step of game-theoretic reasoning on the current action selection. In a *selfish* policy, each agent constructs a belief state using only its own observations and then uses that belief state to evaluate the expected reward of actions given the current heuristic. The agent then implements what it believes to be the best action.

The first of the four utility functions, *simple-heuristic*, is a simplified version of the heuristic used the previous section. This heuristic is also based upon the performance of policies found for shorter time horizons, but has not been hand-tuned for each horizon value (i.e., rather than have a different heuristic for each horizon, the single heuristic that performed best for the most horizons was picked). This heuristic makes the assumption that, after the current timestep, the agents will play according to a previously found approximate solution.

The second utility function, *coop-POMDP*, uses a heuristic that assumes that the agents will be able to fully communicate after the next timestep. This heuristic was calculated using the publicly available *pomdp-solve* (Cassandra, 1999) to solve the POMDP in which a meta-agent has an action and observation space equal to the joint-action and -observation space of the original problem. The utility of a joint-action, joint-type pair, $\{a, \theta\}$, is the expected immediate reward for that joint action given the belief b_θ, plus the sum over the expected future value of each possible outcome of the joint action as given by the POMDP solution. For the *selfish* application of this heuristic, each agent simply constructs its current belief and then applies its part of the best joint action as given by the POMDP solution. This approach is an example of an extremely optimistic heuristic, which results in sub-optimal policies such as the one shown in Figure 3.3. The over-estimation of future reward causes


L
HL HR
L L
HL HR HL HR
OR L L OL
HL HR HL HR HL HR HL HR
L L L L L L L L
L L L L L L L L L L L L L L L L
OR L L OL OR L L OL OR L L OL OR L L OL OR L L OL OR L L OL OR L L OL OR L L OL
L
HL HR
L L
L L L L
L L L L L L L L
L L L L L L L L L L L L L L L L
OR OR OR L OR L L L OR L L L L L L OL OR L L L L L L OL L L L OL L OL OL OL

(a)

(b)

Figure 3.3: Example policies generated for a 6-step version of the *Lady and the Tiger* problem. The policies generated using alternating maximization on the full POSG were the same for each agent and so only one is shown (as a policy tree) in (a). While BaGA is able to find the same policy trees using the heuristic described in Section 3.4.1.1 and the single-POMDP (pessimistic) heuristic of Section 3.4.1.2, other heuristics result in a locally optimal policy in which both agents have a policy given by the policy tree in (b). The action and observations sets are shown using <L,OL,OR> and <HL,HR> respectively in order to save space.

the agents to not open a door at a (riskier) earlier step but rather to put off opening the door until they become more confident about the tiger's location. Due to the misconception that future observations will be pooled (and therefore, not as many steps will be needed to determine the tiger's new position), the agents believe they still have time to open the door twice even though it is no longer true.

The final two utility functions, *single-POMDP* (optimistic) and *single-POMDP* (pessimistic), make the assumption that while the agents will not be able to communicate after the next step, the other agent will be passive. As with *coop-POMDP*, the utility of a joint-action, joint-type pair is the expected immediate reward for the joint action, plus the sum of the expected value of each possible outcome of the joint action given the POMDP solution for each resulting belief. The POMDP formulation used to construct these utility functions only uses the individual action set of an agent and its observations (and associated emission probabilities). The reward function used, however, reflects whether or not the second agent is assumed to pick the same action (optimistic) or only pick *listen* (pessimistic). In the former case, the reward function is overly optimistic because it assumes that the agents will always open doors together, while in the latter it is overly pessimistic as it assumes the agents will never open a door together. What makes the *single-POMDP* (optimistic) utility function different from that of *coop-POMDP* is that only individual observations and their emission probabilities are used to solve the POMDP. For the POMDP solution, it means more *listen* actions are required to generate beliefs for which opening a door is optimal, and for BaGA it means that agents are not as prone to over-estimating the amount of reward that they can receive in a small number of timesteps.

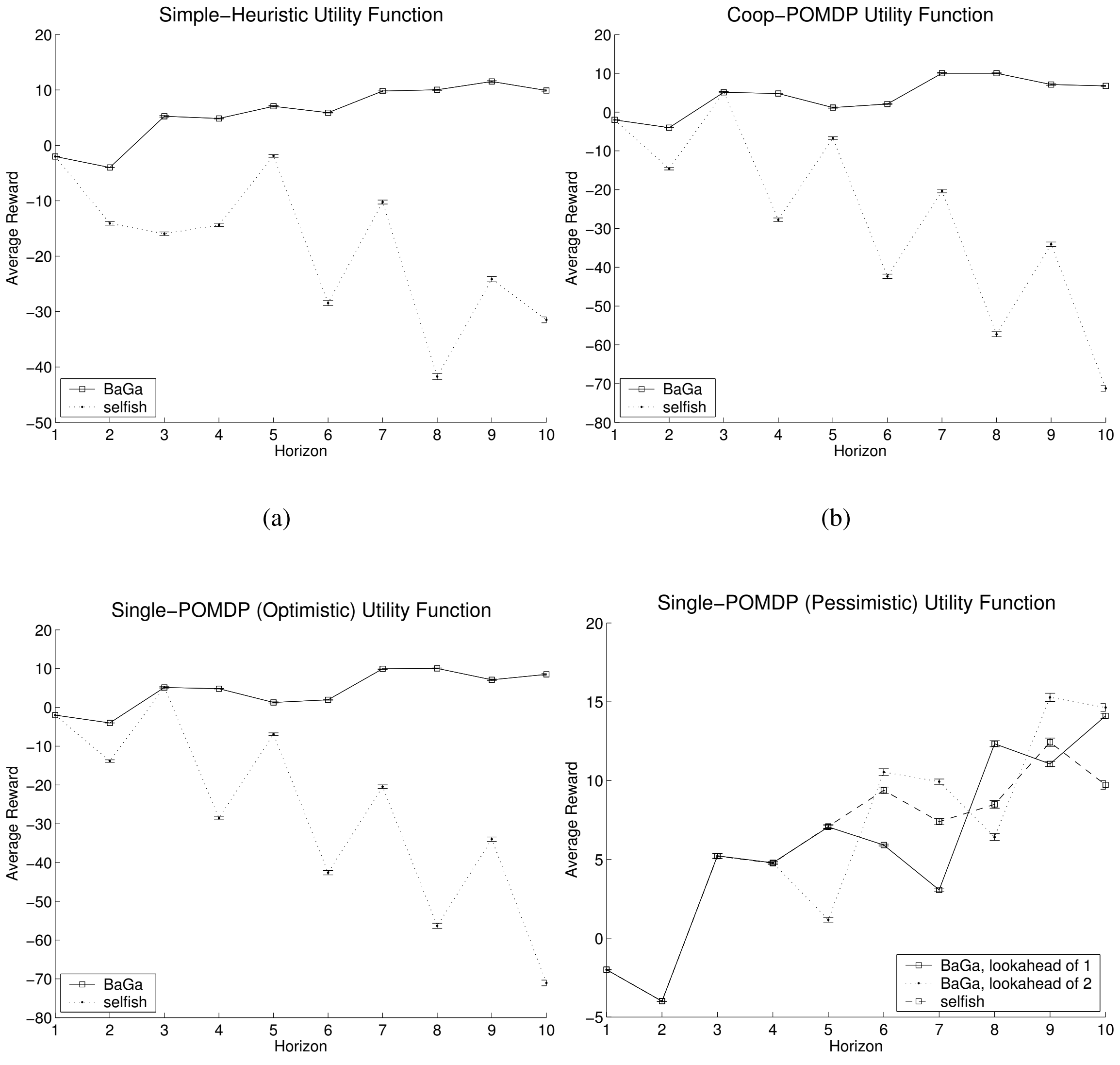
For these problems BaGA ran with 200 random restarts as time was not an issue. The *single-POMDP* (pessimistic) utility function was also run with a quasi additional step of lookahead. It was not a true 2-step lookahead version of BaGA. Instead, the function, when evaluating the expected future value of a <*listen*,*listen*> action, propagates forward all possible observations and then, rather than converting those directly into belief states and looking up their value in the POMDP solution, calculates what happens in the true reward space if the agents are able to coordinate in the next timestep. Regardless of what action is taken in this hypothetical second step, the outcomes are converted into belief states and their expected future values are looked up in the POMDP solution. The result is a fifth utility heuristic that falls somewhere between *coop-POMDP* and *single-POMDP*: it assumes that the agents will be able to communicate (and therefore coordinate) on the next timestep but will not be able to communicate with each other after that.

As shown in Figure 3.4, in all but the *single-POMDP* (pessimistic) case, BaGA clearly performs better than the *selfish* policy that uses the same utility function. The one step of game-theoretic reasoning over the observations and action selection of other agents substantially improves the quality of the resulting policy. As expected, the *selfish* implementation of *single-POMDP* (pessimistic) does much better than the other *selfish* implementations, because the *single-POMDP* (pessimistic) utility function provides a decent estimate of how well a single player can do. The POMDP solution is constructed using only a single agent's observation emission probabilities and uses a reward function that assumes the other agent will never open a door. As a result, a *selfish* agent is much more conservative (performs more *listen* actions before opening a door) when using this heuristic than it is in the other three cases. While no one instance of BaGA outperformed the *selfish* heuristic, BaGA achieved better performance on alternate timesteps with either the basic heuristic or with a version that simulated another lookahead in reward space. This result reflects the cyclical nature of the problem and how on certain timesteps it pays to be more conservative when estimating future reward than on others (see the discussion in Section 3.4.1.1 on the optimal 6-step policy for an example).

These results, however, do not mean that the BaGA approach is invalid, it just emphasizes that the choice of heuristic is important to the overall solution quality. Even with a lower quality heuristic (e.g., one that overemphasizes future coordination), BaGA can result in much higher performance than a selfish approach in which there is no attempt to model the decision-making process of the other agents.

3.4.2 Multiple Access Broadcast Channel

The *Multiple Access Broadcast Channel* problem (MABC) is designed to simulate a communications network in which there are multiple nodes, each of which can receive and transmit messages. In this network, only one message can be transmitted at a time and the overall goal of the system is to maximize the throughput, or number of messages successfully transmitted. Each node has a buffer that is limited to contain only one message. If node i's buffer is empty at timestep t, then a new message will arrive on timestep $t+1$ with probability p_i. There is no communication available to the nodes and so each node must independently decide when to transmit a message. However, nodes receive (possibly noisy) observations about whether or not a message is successfully transmitted: they are informed if a collision occurred, if a message was successfully transmitted or if the system was idle (no node transmitted a message). If a collision does occur, then any messages involved in the collision are lost.



(a)

(b)

(c)

(d)

Figure 3.4: Comparison of different utility functions for the *Lady and the Tiger*. For all cases, joint types with probability less than 0.000005 were pruned and performance was averaged over 100000 trials with 95% confidence intervals shown. In (d), the performance of both the basic utility function and one that attempts to simulate a 2-step lookahead in the reward space are shown. BaGA used 200 random restarts to find each policy.

Table 3.3: The local transition function for the *Multiple Access Broadcast Channel* problem. This function governs the transitions between the local state of agent i with new message rate p_i.

Local State	**Local Action**	***full***	***empty***
full	*transmit*	p_i	1.0-p_i
full	*idle*	1.0	0.0
empty	*idle*	p_i	1.0-p_i

This canonical decentralized MABC problem was formulated by Ooi & Wornell (1996) and later recast as a fully cooperative POSG by Hansen et al. (2004). In this latter version, nodes are only informed if a collision occurred or did not occur; they cannot distinguish between the system being idle and a successful transmission by a single node. It is this reduced observation set that is used in this thesis. This problem has been included because it is an example of a n-agent POSG.

The state space for this domain is the cross-product of the buffer states of each of the n agents, $S_i = \{$*full*,*empty*$\}$, which has a size of 2^n. In this version of the problem, the initial global state is unknown. Each agent has two possible observations, *collision* and *no-collision*, in addition to full knowledge of its own local state. The observation space is $Z_i = \{$*collision*,*no-collision*$\} \times \{$*full*,*empty*$\}$ except for the first timestep in which $Z_i = \{$*full*,*empty*$\}$ as no observation about collisions has yet been received. This problem is implemented with no noise in the observations and so each agent receives an identical observation of either *no-collision* or *collision*. The problem, however, is still partially observable because, in addition to having only knowledge of its local state and own action, an observation of *no-collision* does not, by itself, allow a single agent to distinguish between a teammate successfully sending a message or no one sending a message.

Each agent has two possible actions, $A_i = \{$*transmit*, *idle*$\}$; however, if an agent has a local state of *empty*, it can only choose the *idle* action. This restriction is used to reduce the overall number of action choices that must be represented and reflects the fact that the *transmit* action has no effect on an empty buffer. The local state transition of each agent is shown in Table 3.3. The team receives a reward of 1 for each message successfully transmitted and otherwise it receives nothing.

Like the *Lady and the Tiger* problem, MABC is small enough to allow the full POSG to be constructed and solved for short time horizons. The full POSG was solved using the dynamic-programming version of alternating maximization on its sequence form with 10 random restarts. Table 3.4 compares performance and computational time for this problem with various numbers of nodes, time horizons and two sets of new message rates p_i.

For the BaGA algorithm, each agent's type at timestep t consists of its entire observation and action history up to that point. For example, a possible type at timestep 3 is $<\{$*full, transmit*$\}$, $\{$*no-collision, empty, idle*$\}$, $\{$*no-collision*, *full*$\}>$. Each joint type θ completely defines a state in S, s_θ, because the problem is collectively fully observable. Observations are therefore used as common knowledge to help prune impossible joint types. For example, it is common knowledge that all agents receive the same observation and so any joint type with conflicting observations has zero probability. Furthermore, if the policy found at timestep $t-1$ has only one agent transmitting, then it is impossible for a collision to have occurred and so any joint type with the observation *collision* can be cut. Additionally, joint types with probability less than 0.000001 are pruned during the construction of the type space Θ^t and the probability distribution over the joint-type space renormalized. If an agent's true history is pruned, then for action selection it is assigned the closest matched type in Θ^t_i. This match is found using Hamming distance. At each timestep alternating maximization with 20 random restarts is used for policy construction

Three different heuristic functions were considered for evaluating the utility of actions. The most simple of the heuristics, *greedy*, defines the value of a joint action a to be only its immediate value given the global state defined by the joint type θ. This heuristic is defined entirely by the reward function. The second heuristic function, *fully-cooperative*, assumes that after the current timestep, the agents will have full communication. The utility of a joint action, joint type pair, $\{a, \theta\}$, is therefore the immediate reward for that joint action given the state s_θ, plus the sum over the future value of each possible next global state s'_θ as given by the solution to a fully observable version of the problem.

The final heuristic, *no-communication*, makes no assumptions on the information available to the team after the current timestep. As with *fully-cooperative,* the expected value of $\{a, \theta\}$ at timestep t is defined to be the immediate reward given the global state s_θ plus the sum over the expected value of each possible next state s'_θ. Unlike *fully-cooperative,* this expected value of s'_θ does not come from the fully observable problem, but rather from the solution to the full POSG with starting conditions s'_θ and a horizon of $t-1$. The solutions to full POSGs representing all possible starting conditions and time horizons were precalculated in order to implement this heuristic. It was possible to do so because this heuristic does not require the most computationally expensive conditions of $\{n = 3, T = 4\}$ or $\{n = 4, T = 3\}$ to be calculated for each possible global states, but rather only for

$\{n = 3, T = 3\}$ and $\{n = 4, T = 2\}$.

As with *Lady and The Tiger*, a *selfish* version of each heuristic function was also evaluated. For the *selfish* policy, each agent constructs a belief state over the the global state S based on its own local state and the new message rates of its teammates, p_{-i}. Using that belief state, the agent chooses the joint action with highest expected reward, given the appropriate heuristic for evaluating utility, and then implements its part of that joint action.

Using the *no-communication* heuristic for a utility function, BaGA is able to perform similarly to the full POSG, but at a fraction of the computational cost. As seen in Table 3.4, the savings in time is a direct effect of the smaller set of joint types that are represented in the BaGA algorithm. Unlike the full POSG, BaGA interleaves planning and execution, and so it is able to prune away large parts of the game tree due to early decisions. It also does not need to represent joint types that it knows to be impossible such as those that include a *collision* observation even though only one node transmitted. If the new message rates, p_i, are such that one node has a much higher probability of receiving a message than the other nodes, then the optimal solution to the POSG has only that one node transmitting. The other nodes are always idle in order to avoid a collision. This type of policy allows BaGA to prune away more joint types than in the case where $p_i = p_j$ for all $i \neq j$. The sequence form for the full POSG, however, cannot benefit from this pruning.

The *no-communication* heuristic is very accurate: it uses smaller versions of the full POSG to evaluate the future expected value of taking actions. As anticipated, as the heuristic used for the utility function increases in accuracy, the overall performance of BaGA also increases. This fact can be seen in Table 3.5 in which the performance of the three heuristics, *greedy*, *fully-cooperative*, and *no-communication* are compared. Results for the *selfish* implementation of each policy are also included to show how one step of game-theoretic reasoning improves performance over blindly applying the base heuristic. The *greedy* heuristic is the least accurate of the three heuristics because it only uses immediate reward to evaluate actions, but it is also the easiest to evaluate. The *fully-cooperative* heuristic is more accurate because it includes an estimation of future expected reward. It can therefore identify when it is advisable for a node to hold off on transmitting on the current timestep in order to transmit on a future timestep. The estimate of future expected reward, however, assumes that the nodes will have full communication after the current timestep. It is therefore less accurate than *no-communication*, in which the future expected value is calculated using the original problem dynamics. However, it is much easier to evaluate the solution to a fully-cooperative version of the game than it is to solve the full POSG with horizon $T-1$ for each possible global state.

Table 3.4: Computational and performance results for MABC when the initial global state is unknown. Expected reward and computation time is shown for the full POSG solution. For the BaGA algorithm, rewards for the *no-communication* utility heuristic are averaged over 10000 trials with 95% confidence intervals shown. The average number of joint types at each timestep is shown for BaGA and, unless stated otherwise, 95% confidence intervals were 0.005 or smaller. Timing experiments were performed on a Intel Pentium 4, 2.80 GHz machine.

Parameters	**Full POSG**			**BaGA**		
	Exp. Reward	**Time(ms)**	$\vert\Theta^t\vert$	**Avg. Reward**	**Time(ms)**	$\vert\Theta^t\vert$
$n = 2, T = 5,$ $p = \{0.4, 0.4\}$	2.96	243000	{4, 100, 2704, 73984, 2027776}	2.96 ± 0.02	35	{4, 12, 36, 107.99 ± 0.01, 323.96 ±0.04}
$n = 2, T = 5,$ $p = \{0.7, 0.3\}$	3.77	243000	{4, 100, 2704, 73984, 2027776}	3.77 ± 0.02	30	{4, 12, 32, 80, 239.91 ± 0.04}
$n = 3, T = 3,$ $p = \{0.4, 0.4, 0.4\}$	1.82	36000	{8, 1000, 140608}	1.84 ± 0.02	90	{8, 36, 144}
$n = 3, T = 3,$ $p = \{0.75, 0.5, 0.25\}$	2.44	36000	{8, 1000, 140608}	2.40 ± 0.01	85	{8, 36, 136.92 ± 0.16}
$n = 3, T = 4,$ $p = \{0.4, 0.4, 0.4\}$	2.61	6960000	{8, 1000, 140608, 20123648}	2.62 ± 0.02	370	{8, 36, 144, 576.99 ±0.01}
$n = 3, T = 4,$ $p = \{0.75, 0.5, 0.25\}$	3.31	6960000	{8, 1000, 140608, 20123648}	3.31 ± 0.01	330	{8, 36, 128.05 ±0.02, 480.12 ± 0.06}
$n = 4, T = 3,$ $p = \{0.4, 0.4, 0.4, 0.4\}$	1.90	5570000	{16, 10000, 7311616}	1.91 ± 0.01	6700	{16, 144, 864}
$n = 4, T = 3,$ $p = \{0.8, 0.6, 0.4, 0.2\}$	2.60	5570000	{16, 10000, 7311616}	2.58 ± 0.01	1800	{16, 108, 575.72 ± 0.08}

Table 3.5: Comparison of three different utility functions for MABC: *greedy, fully-cooperative* and *no-communication*. For all cases, joint types with probability less than 0.000001 were pruned and performance was averaged over 10000 trials with 95% confidence intervals shown. The average sum of joint types over each timestep is shown for BaGA with 95% confidence intervals. Results are also included for the selfish application of each heuristic.

Parameters	***greedy***			***fully-cooperative***			***no-communication***		
	BaGA		***selfish***	**BaGA**		***selfish***	**BaGA**		***selfish***
	Avg. Reward	$\sum \|\Theta^t\|$	**Avg. Reward**	**Avg. Reward**	$\sum \|\Theta^t\|$	**Avg. Reward**	**Avg. Reward**	$\sum \|\Theta^t\|$	**Avg. Reward**
$n = 2, T = 5,$ $p = \{0.4, 0.4\}$	2.40 ± 0.02	1364.00 ± 0.00	2.38 ± 0.02	2.96 ± 0.02	483.98 ± 0.02	2.34 ± 0.02	2.96 ± 0.02	484.85 ± 0.05	2.41 ± 0.02
$n = 2, T = 5,$ $p = \{0.7, 0.3\}$	3.74 ± 0.02	367.32 ± 0.11	2.90 ± 0.02	3.77 ± 0.02	367.89 ± 0.05	2.90 ± 0.02	3.77 ± 0.02	367.91 ± 0.04	2.90 ± 0.02
$n = 3, T = 3,$ $p = \{0.4, 0.4, 0.4\}$	1.77 ± 0.02	272.18 ± 0.03	1.29 ± 0.02	1.76 ± 0.02	272.00 ± 0.00	1.27 ± 0.02	1.84 ± 0.02	188.00 ± 0.00	1.29 ± 0.02
$n = 3, T = 3,$ $p = \{0.75, 0.5, 0.25\}$	2.41 ± 0.01	181.06 ± 0.16	1.22 ± 0.17	2.40 ± 0.01	180.93 ± 0.16	1.22 ± 0.02	2.40 ± 0.01	180.92 ± 0.16	1.21 ± 0.02
$n = 3, T = 4,$ $p = \{0.4, 0.4, 0.4\}$	2.53 ± 0.02	1137.84 ± 0.24	1.72 ± 0.02	2.63 ± 0.02	763.99 ± 0.01	1.71 ± 0.02	2.62 ± 0.02	763.99 ± 0.01	1.74 ± 0.02
$n = 3, T = 4,$ $p = \{0.75, 0.5, 0.25\}$	3.24 ± 0.02	694.79 ± 0.74	1.63 ± 0.02	3.32 ± 0.01	652.12 ± 0.07	1.63 ± 0.02	3.31 ± 0.01	652.17 ± 0.08	1.61 ± 0.02
$n = 4, T = 3,$ $p = \{0.4, 0.4, 0.4, 0.4\}$	1.89 ± 0.02	1024.00 ± 0.00	1.03 ± 0.02	1.89 ± 0.02	1024.00 ± 0.00	1.03 ± 0.02	1.91 ± 0.02	1024.00 ± 0.00	1.03 ± 0.02
$n = 4, T = 3,$ $p = \{0.8, 0.6, 0.4, 0.2\}$	2.58 ± 0.01	699.73 ± 0.08	0.75 ± 0.01	2.59 ± 0.01	699.69 ± 0.09	0.74 ± 0.01	2.58 ± 0.01	699.72 ± 0.08	0.74 ± 0.01

The lower performance of the *greedy* heuristic compared to the other two is most noticeable in the case when all nodes have the same new message rate. Unlike the case in which one node has a much higher p_i than the rest (and therefore leads to a policy in which it is the only node to transmit), all nodes have equal status and so the future effects of actions becomes more important. Additionally, because p_i is set fairly low, it is now possible for a solution to have policies in which two nodes transmitting messages on the same timestep. These policies do not actually result in a lot of collisions because the probability that both nodes have a full buffer (and therefore actually transmit at the same time), is very low.

For the most part, the *fully-cooperative* and *no-communication* heuristics perform similarly with the only exception being the $n = 3, T = 3, p_i = 0.4$ case in which *no-communication* outperforms the other two heuristics. This result is promising because the *fully-cooperative* heuristic is easier to compute and so it can be more easily be extended to problems with longer time horizons as will be done in the following chapter.

In all the results presented, the network starts out with an unknown global state. This leads to policies in which, if one node has a much higher new message rate, it is the only one which ever implements a transmit action. If BaGA (or alternating maximization on the full POSG) is started with a known global state (such as all nodes having full buffers), then the optimal policy changes. For example, if $n = 3, T = 3$ and all nodes start with full buffers, then regardless of the new message rate of each node, a different node will transmit on each timestep leading to a guaranteed reward of 3. In this fashion, BaGA is able to generate policies that performed equivalently to the results in Hansen et al. (2004) for $n = 2, p = \{0.9, 0.1\}, T \leq 4$, and a known initial global state (e.g., all buffers full).

3.4.3 Robotic Tag

The *Robotic Tag* domain is a 2-robot version of Pineau et al.'s *(2003a) Tag* problem. In this problem, two robots have 100 timesteps to herd and capture an opponent that moves with a known stochastic policy in the discrete environment shown in Figure 3.5. Once the opponent is herded into the goal cell, the two robots must coordinate on a *tag* action in order to receive a reward and finish the task.

The state space for this problem is the cross-product of the two robot positions, $r_i = \{s_0, \ldots, s_{28}\}$, and the opponent state $op = \{s_0, \ldots, s_{28}, s_{tagged}\}$. The size of the state space $S = r_1 \times r_2 \times op$ is 25230 states. All three agents start in independently selected random positions and the game is over when $op = s_{tagged}$. Each robot has five actions,

$A_i = \{$*north, south, east, west, tag*$\}$, and all actions have deterministic effects (e.g., an action of *east* will always move the robot right a cell unless the robot is at the eastern boundary of the environment). A shared reward of -1 is imposed for each robot's motion action while the *tag* action results in a reward of +10 to the team if both robots select *tag* and the conditions shown in Figure 3.5 are met. Otherwise, this action results in a reward of -10 to the team.

Each robot is able to detect its current cell position with certainty and so has full observability of its own local state. Each robot however, is only able to sense only its current cell and so the position of the opponent is completely unobservable unless the opponent is in the same cell as the robot. Two versions of this problem are considered: in the first, the position of the two robots are fully observable to each other (*known-teammate-position*) and in the second, a robot only observes its teammate's position if they are in the same cell (*unknown-teammate-position*). There is no communication between the robots and so they cannot share observations, nor their position in the case of *unknown-teammate-position*. The starting position of each robot is, however, always known with certainty. The opponent always selects actions with full knowledge of the positions of the robots and has a stochastic policy that is biased towards maximizing its distance to the two robots. In this fashion, even though multiple agents can be located in the same cell, the robot team can still move in such a way as to influence the position of the opponent.

This problem is interesting with respect to decentralized decision making for several reasons. First, it is a much larger problem than that traditionally looked at in the POSG literature. Second, because the opponent is attempting to maximize its distance to the robot team the robots must coordinate their actions in order to 'herd' the opponent towards the goal cell. This coordinated herding is necessary even in a fully observable version of the problem and is only made harder by the robots having such limited sensors. Finally, the robots must also coordinate on selecting the *tag* action at the same time and in the right place in order to win.

A fully observable policy was calculated for a robot team that could always observe the opponent using dynamic programming and assuming 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 utility in the BaGA algorithm. This utility function assumes that, after the game-theoretic reasoning of the current timestep, the robots will have full observability of the problem.

In the first version of the problem, *known-teammate-position,* the set of types, Θ_i, for each

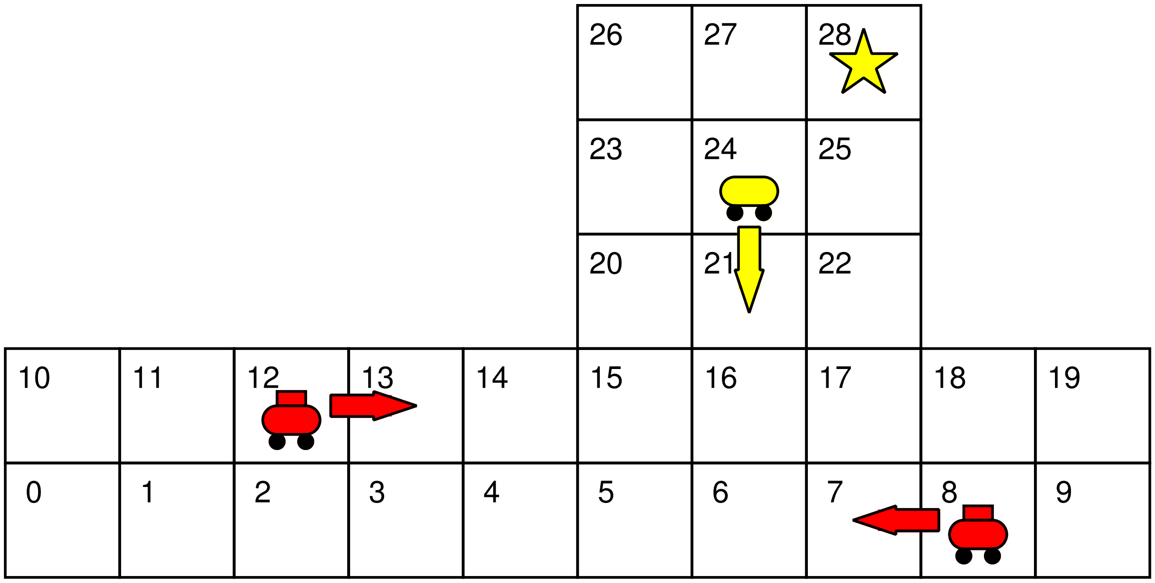


Figure 3.5: Abstract Wean Hall environment for *Robotic Tag*. This picture shows the environment in which the two robots are trying to capture the opponent. They can only successfully tag the opponent if it is in the cell marked with the star and at least one of the robots is also in that cell with the other robot in one of Cells 25, 27 or 28.

robot consists of all possible histories of observations of the opponent. It is not necessary to include information about the actions taken by each robot because their positions are completely known to the team and actions are deterministic (and therefore, any joint history that has an action for a robot given by π^t that is inconsistent with the robot's position at timestep $t + 1$ can be pruned).[10] While this version of the problem has a branching factor with an upper bound of three (on each timestep there are as many as three times the number of possible joint histories, or types, as there were on the previous timestep), in practice many fewer types are kept at each timestep. The common knowledge of teammate position allows any type for which the policy assigns an action that does not match the robot's current position to be pruned from further consideration. Because of this pruning, the size of the type space for a robot rarely got above 20 in the experimental runs and was frequently much smaller. As a result, each robot actually has a fairly good idea of what observations its teammate has made.

[10] Alternatively, one can consider the local state of each robot to be part of its observations about the global state of the problem and augment the type space accordingly. All types that do not conform to the true local state of the robot, however, will be pruned because in the *known-teammate-position* version of this problem this local state information is known to be common knowledge.

In *unknown-teammate-position*, a history tracks the robot's position and action taken at each timestep as well as its observations of its teammate and the opponent. This information allows the probability distribution over joint types to be modified both by using the policies π_i computed by BaGA and through mutual observations (or lack thereof). This version of the problem also has a branching factor with an upper bound of three (due to perfect sensing of both the teammate and opponent); however, there is no longer that common knowledge of teammate position to keep the size of the type space bounded. Instead, over time, the size of the type space grows. In the experimental runs, a higher pruning threshold of 0.0025 was used for the *unknown-teammate-position* condition but still, the total number of types was about an order of magnitude greater than that of the *known-teammate-position* variant.

For either version of the problem, each joint type θ contains enough information to define a belief state b_θ over the position of the opponent in the environment and the positions r_1 and r_2. It is possible, however, that θ is not a valid joint history and so during the construction of the set of possible joint histories at timestep $t+1$, any joint history that does not result in a valid belief state is pruned (i.e., its probability of occurring is zero). For example, two histories that have the robots seeing the opponent in two different cells cannot combine to create a valid joint history. If observations of the teammate are part of the histories, then a valid joint history will have teammate observations that are consistent with the robot's position as given by their histories. Note that this is a direct consequence of the sensor model being used in this domain and is not true for all problems.

Given a valid joint type, θ, the utility of taking the actions $a_1$and a_2 is:

$$u(a_1, a_2, \theta) = \sum_{s \in s_{op}} b_\theta(s) Q(\{r_1, r_2, s\}, \{a_1, a_2\})$$

where $Q(\{r_1, r_2, s\}, \{a_1, a_2\})$ is the Q-value of taking the joint action $\{a_1, a_2\}$ in the state given by the positions r_1, r_2 and $op = s$. Because the definition of a Q-value is the reward in the current state plus the discounted future reward (assuming an optimal policy is followed from then on), this heuristic combines the immediate expected reward of taking the actions a_1 and a_2 with the future expected reward rather than calculating them separately. For this problem, joint histories with probability less than 0.00025 were pruned from consideration in the *known-teammate-position* case and 0.0025 in the *unknown-teammate-position* case. If a robot's true history is pruned away then it implements the action for the type with which it has the smallest Hamming distance. In the next chapter, a better method for type matching is discussed.

Two common heuristics for partially observable problems were implemented in addition to BaGA. In these two heuristics, each robot maintains an individual belief state based only upon its own observations and uses that belief state to make joint action selections. It then implements its half of the joint action it selected. In the case of *unknown-teammate-position*, each robot also maintains a believed position of its teammate that it updates by assuming that its teammate will implement the other half of the joint action it selects at each timestep and by using any positive observations of its teammate (the only information about action selection to which it has access). That believed position is then used as the value for r_{-i} in its action-selection process.

In the Most Likely State (MLS) heuristic, a robot selects the most likely opponent state from its belief state and then implements 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. This Q_{MDP} heuristic is similar to the *selfish* application of the *Lady and The Tiger's* utility functions in that each robot implements the action given by the BaGA utility function but for its independently constructed belief state.

Table 3.6 compares the performance of MLS, Q_{MDP} and BaGA for both the *known-teammate-position* and *unknown-teammate-position* conditions. Success is the percentage of trials in which the opponent is caught within 100 timesteps. Also included is how a team with full individual observability of the problem performs. While this is an upper bound on performance in this domain, it is not intended as a realistic estimate of how the optimal solution to this problem, given the true individual partial observability, would perform. A more realistic bound would be found by solving the fully cooperative POMDP version of this problem; however, the problem is too large to do so using current approximation techniques.

With an opponent that tries to maximize its distance from the team, a good strategy is for each robot to go towards different ends of the main corridor and then 'herd' the opponent towards the goal cell. Herding can be accomplished without ever actually making a positive identification of the opponent's position. For example, using the BaGA policy, each robot makes its way down to one of the two ends of the corridor and then waits a couple of timesteps until its teammate reaches the other end. By this time, it is highly likely that, if the opponent was in this corridor, it would have started to make its way towards the middle. The robots then start to move towards the middle of the corridor in lock-step, driving the opponent up into the room at the top. The robots then move into this room in such a way as to force the opponent into the upper right corner.

Table 3.6: Results for *Robotic Tag* in the Wean Hall environment. Average cumulative reward for the team as well as average number of iterations to capture the opponent are shown with results averaged over 10000 trials with 95% confidence intervals shown. % success is the percentage of trials in which the opponent was successfully tagged within 100 timesteps. Results are also shown for the performance of the fully observable policy under full observability. In the *known-teammate-position* variation, joint types with probability less than 0.00025 are removed from the type space and in the *unknown-teammate-position* this threshold was raised to 0.0025.

Algorithm	**Average Reward**	**Average Iterations**	**% Success**
known-teammate-position condition			
Fully Observable	-17.61 ± 0.24	14.80 ± 0.12	100.0%
BaGA	-73.26 ± 1.32	39.24 ± 0.59	86.8%
MLS	-106.32 ± 1.51	54.72 ± 0.11	74.2%
Q_{MDP}	-168.70 ± 1.48	78.88 ± 0.57	41.0%
unknown-teammate-position condition			
BaGA	-88.15 ± 1.52	45.69 ± 0.67	77.5%
MLS	-186.85 ± 1.63	82.74 ± 0.64	23.4%
Q_{MDP}	-202.04 ± 0.81	96.23 ± 0.33	4.9%

Depending on the starting conditions of the two robots, BaGA does not always follow this policy exactly. For example, consider the robot trajectories shown in Figure 3.6. In this case, the two robots start near the top room, and so they move towards the goal cell in an attempt to quickly catch the opponent (Figure 3.6 (a)). If the opponent was there then it would be captured; however, it is not, and so the robots move back down into the corridor (Figure 3.6 (b)) and start herding the opponent towards the goal (Figures 3.6 (c) through (f)). While this sort of strategy pays off enough to make it worth while in expectation, if the opponent is not in the top room initially, the discovery of this fact usually comes at the cost of a failed *tag* action initiated by the robot that is not in the goal cell.

In other runs, because the opponent's behaviour is only biased towards moving away from the team, the low-probability event of the opponent not moving away from the end of a corridor causes the robot that is handling that end of the corridor to start to move towards the middle without herding the opponent in front of it. If the robot sees the opponent it is able to correct its policy; otherwise, once this it discovered through a miscoordination of the tag action at the goal cell, the robots go back and are successful. A robot, however, does wait multiple timesteps at the end of a corridor, which minimizes the probability that the opponent will not move. Overall, the robots are able to capture the opponent about 87% of the time within 100 steps, averaging 40 timesteps on successful trials.

With the MLS heuristic, the robots will only pick coordinated actions if they have similar belief states. If they do have similar beliefs, then they will both select similar states as the most likely position for the opponent and follow a similar herding strategy to the one above, but modified to take into account where they think the opponent actually is. For example, if they start in the middle of the corridor, then the belief about the opponent's position quickly becomes tri-modal as the opponent will move away into either the room or opposing ends of the corridor. Depending on which mode has higher probability, the robots will either sweep towards the ends of the hall or into the room directly. As more observations are made, the distribution over the modes will shift, which can cause a robot to decide that a new location is more likely. For example, if a robot is moving towards the left end of the corridor because it believes one of those positions to be the most likely state for the opponent but then switches to believing the room is more likely, it will not go right to the end of the corridor but instead turn around. Therefore, if the opponent was actually at the end of the corridor, it will not be herded. Furthermore, if the robots start in very different locations, their belief states will still be multi-modal, but can be quite different from each other, leading each robot to select a different position for the opponent and as a result be less effective with its herding. Eventually, as more time passes, the belief states of the robots will become more accurate, allowing them to capture the opponent. It takes about 15 more timesteps on average for MLS to do so than it does BaGA.

Q_{MDP} also results in robots creating multi-modal beliefs about the opponent's position; however, because each robot will pick the joint action with highest expected reward, this heuristic can fall into the common trap of using such an approach. As discussed by Cassandra (1998), Q_{MDP} assumes that any uncertainty will disappear after the current action is executed and so if there is one action that is essentially reward neutral (receives similar reward for each possible state), then it will be selected by Q_{MDP} because the algorithm thinks it can still do well on the future (fully observable) steps. But of course, if that action has little or no effect on the belief state then it will continue to be selected, causing a lack of progress towards the goal. For *Robotic Tag,* if the best action for each mode of the belief is very different, then no single action is best for all cases and so actions which have the robots stay in their current positions or move towards the center of the environment appear better (i.e., are reward neutral). As a result there are certain robot positions or sequences of actions that lead the team to stop moving or to oscillate between two locations. This behaviour is reflected in the much lower success rate of Q_{MDP}. MLS does not fall into this same trap because the robots always choose actions that are best for a single position. Un-

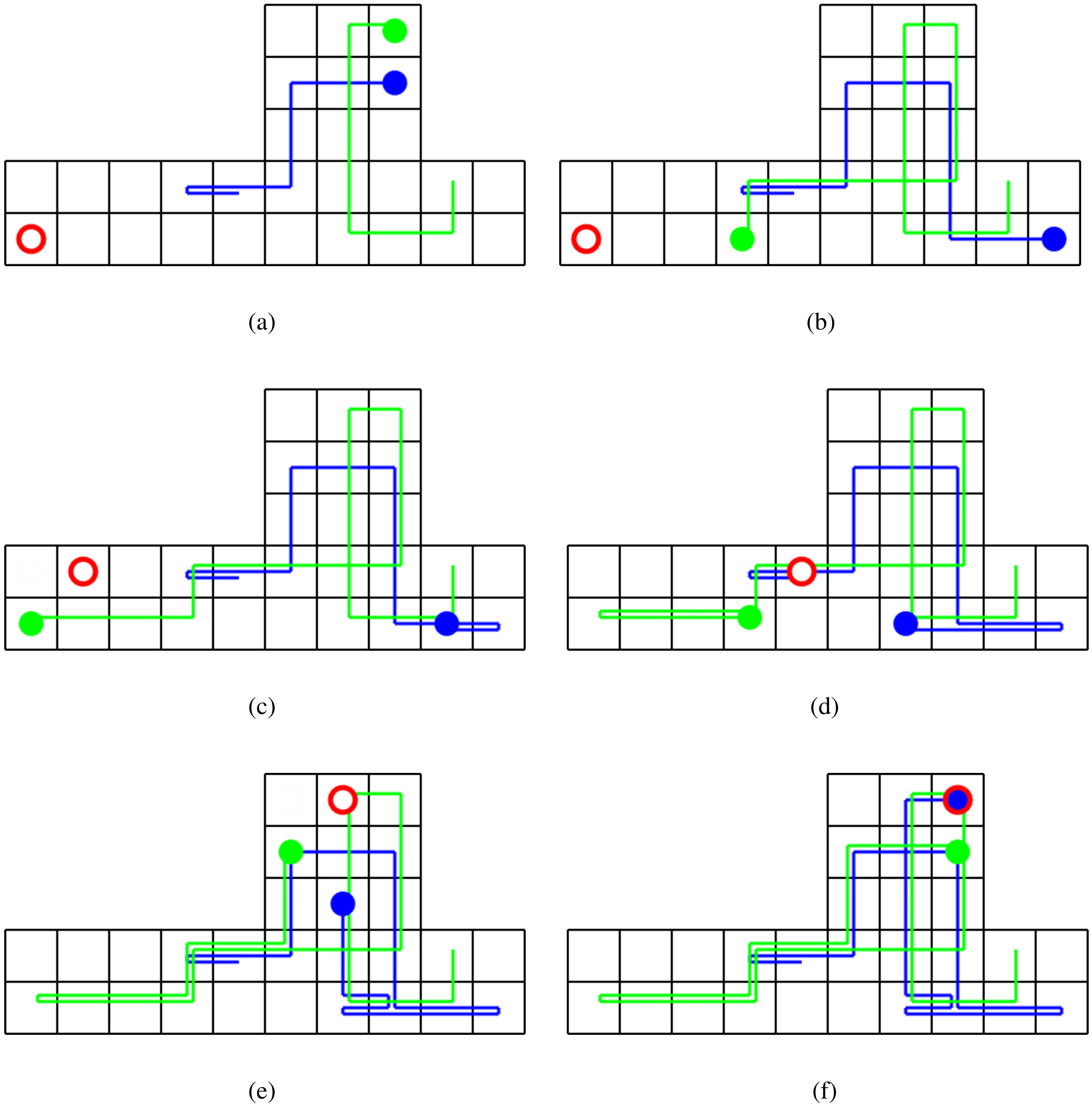


Figure 3.6: Example of a BaGA policy for the *known-teammate-position* variation of *Robotic Tag*. The trajectories taken by the robots demonstrate how the team coordinates to capture the opponent without any communication. The robots first search the top room and then move out into the ends of the corridors. Members of the robot team are shown as the solid circles.

less this most likely opponent position rapidly flip-flops between two locations, the robots are able to make progress. While BaGA does use Q_{MDP} as its heuristic, for the most part, the additional step of game-theoretic reasoning on top of Q_{MDP} prevents the robots from engaging in this type of behaviour. There is, however, one configuration that does cause the robots to stop moving or oscillate. It occurs when one robot is in Cell 7 and the other in Cell 20 and and is caused by how the cell configuration, sensor model and the opponent's policy interact with each other to create belief states. The majority of the trials in which the team does not catch the opponent seem to be caused by the robots entering this configuration, and not being able to leave it.

The *unknown-teammate-position* modification highlights the importance of reasoning about the actions and experiences of others. With MLS and Q_{MDP}, the robots assume that their teammate will implement the other half of the joint action they select and use that assumption to update a believed position of the teammate used for future action selection. This incorrect assumption makes the believed position of the teammate quickly diverge from the true position with only positive observations of the teammate able to rectify the situation. In this problem, however, robots rarely move into the same cell. Furthermore, each robot must use this believed location of its teammate to propagate its belief about the opponent's position because the opponent is trying to move away from the team. As a result, unlike in the *known-teammate-position* version, robots might actually hold a belief about the opponent's position that is totally inconsistent with its true position. For example, a robot might believe that the opponent is in the top room because it is at one end of the corridor and it believes its teammate is at the other. In reality, if the teammates are beside each other (but in separate cells), there is also a high probability that the opponent is just at the other end of the corridor rather than the low (or sometimes zero) probability believed by the robot. This is a self-reinforcing problem: as the belief state of the two robots starts to diverge, the joint actions they select differ leading them to update the believed position of the teammate incorrectly, which in turn causes their belief states to diverge from each other even more.

As the believed teammate and opponent position becomes more and more incorrect, the joint action selected by a robot becomes less appropriate and less likely to match the one selected by its teammate, which leads to sub-optimal behaviour. For example, in MLS and Q_{MDP} robots frequently go down the same branch of the hallway because they both incorrectly make the assumption that their teammate will go down the other branch.

With BaGA and its maintenance of a probability distribution over joint types, however, the robots are able to better track the true position of their teammate, allowing them to effectively maintain consistent belief states over the opponent's position and therefore coordinate their actions. As a result, while MLS and Q_{MDP} controllers for the *unknown-teammate-position* case show a huge degradation in performance, BaGA does not. Instead, it performs much more similarly to how it did in *known-teammate-position*, both with respect to the percentage of successful runs and average reward. If one examines the joint type that is most probable given the true type of a robot, that joint type will usually reflect the true position of its teammate even though the robots have never seen each other (aside from common knowledge of starting positions). Because each robot has a very good idea about where its teammate is, they are still able to perform the herding maneuver required to sweep the opponent into the goal cell.

An example of a BaGA policy for the *unknown-teammate-position* case is shown in Figure 3.7. In Figures 3.7 (a) and (b), the members of the team make their way to the ends of the corridors. This drives the opponent out of the right-hand corridor and up towards the top room. After a few timesteps, the robots start to make their way back towards the center of the environment (Figures 3.7 (c) and (d)), and then up into the top room. Finally, they coordinate on trapping the opponent and tagging it (Figure 3.7 (f)).

3.5 Summary

In this chapter, two algorithms for finding approximate solutions to POSGs with common payoffs were presented. The first, an alternating-maximization algorithm, is used to find locally optimal solutions to an entire POSG at once. While it can be used to find solutions for small problems like the *Lady and the Tiger* and MABC, this algorithm still has high memory and computational costs, making it inappropriate for bigger problems like *Robotic Tag*.

The second algorithm presented, BaGA, is the core contribution of this thesis. BaGA finds approximate solutions to a POSG by transforming the original problem into a sequence of Bayesian games representing each timestep. In each of these smaller games, the robots reason about all possible joint histories of observations and actions that might have occurred in order to find policies for each member of team. Each policy is a mapping from a robot's set of possible individual histories to its sets of actions. So long as each robot builds and solves the same sequence of games, the team can coordinate on action selection even in the absence of communication. Heuristic evaluations of expected future reward and the pruning of low-probability joint histories are used to help keep computation tractable. This process results in policies that are locally optimal with respect to the heuristic function

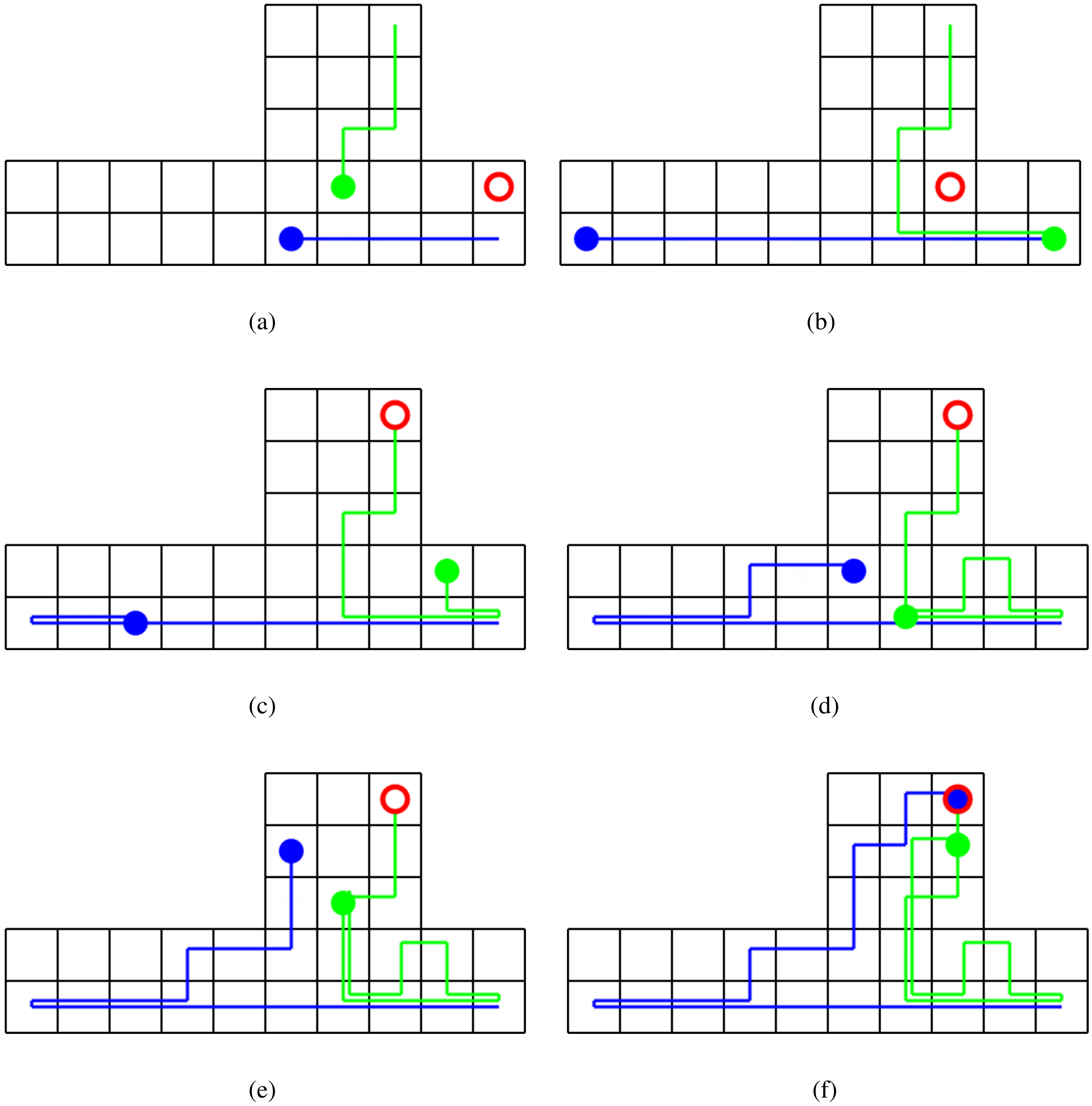


Figure 3.7: Example of a BaGA policy for the *unknown-teammate-position* variation of *Robotic Tag*. The trajectories taken by the robots demonstrate how the team coordinates to capture the opponent even when the robots do not know where their teammate is with certainty. Members of the robot team are shown as the solid circles.

used.

This basic BaGA algorithm is able to find solutions for *Lady and the Tiger* and MABC that perform comparably to those generated by alternating maximization. For the larger *Robotic Tag* domain, the trajectories taken by BaGA-controlled robots demonstrate how BaGA results in better coordination than two common heuristics for decision-theoretic control. Over the next two chapters, additional refinements will be made to BaGA in order to further improve its efficiency and performance.

Chapter 4

Improving Efficiency with Clustering

The benefit of using BaGA over the full POSG for robot controllers is that it calculates a good approximation of the optimal policy tractably. The overall desirability of this approximation is therefore closely tied to how fast BaGA can construct these policies. The required computation time, however, is directly related to the number of individual and joint histories that are maintained in the type and joint-type spaces. Limiting the algorithm to use only a representative fraction of the total number of histories is key to improving the overall efficiency of BaGA and, therefore, to applying this approach to real robot problems.

In this chapter, different ways of limiting the number of histories maintained in the joint-type space by the BaGA algorithm are compared. The simplest of these methods is the pruning used by BaGA to remove joint types from Θ^t that have probability lower than some threshold. While this scheme is an efficient means of keeping the number of types, or histories, low, it generates a crude approximation of the real joint-history space. Common sense dictates that better performance would be achieved by grouping together similar histories; after all, when humans make decisions they do not consider the exact sequence of actions and observations they have made since birth, but only those features that are relevant to their current decision. BaGA-Cluster captures that intuition by grouping together histories that result in the same decisions being made now and in the future (Emery-Montemerlo et al., 2005).

For example, in the *Lady and the Tiger* problem, BaGA applied to a 6-step version of the problem results in a policy in which an agent listens twice, opens a door if it receives the same observation of the tiger's position, listens twice again and then again opens a door if it received the same two observations of the tiger's position. For all histories in which Agent 1 opens a door on the third timestep, by the time the last timestep is reached, the best action for it to take will not be based on any of the observations it made in timesteps 1 through 3, because, if a door is opened, then the tiger's position is reset and those observations provide no useful information. This notion extends to Agent 2's action selection. Agent 2 does not have to worry about the actual observations made by Agent 1 before timestep 3: given Agent 1 opened the door, those observations have no bearing on the current position of the tiger and so they all lead to the same joint belief when combined with any one Agent 2's own individual histories. Therefore, they also all have the same impact on the expected value of actions. The only important thing from Agent 2's perspective is, given one of its observation histories, what is the probability that Agent 1 opened a door on timestep 3? This quantity can be found by summing the conditional probabilities of all of Agent 1's histories in which it does open the door.

This idea can be extrapolated to any sequence of observations and actions that ends with a door being opened. Regardless of what those observations were, for all those histories the agent currently believes that the tiger has just reset its position. For any instantiation of those past observations and actions, the same action would be now be selected by the agent: to *listen*. These histories also have similar effects on the decision-making process of the teammate. A great deal of computational savings could be made if, instead of reasoning about all of those possible histories, the agent (and its teammate) could group them together. Of course, stating that similar histories should be grouped together is a nebulous statement that requires a formal definition of what is meant by 'similar'.

4.1 Low-Probability Pruning

In the original implementation of BaGA, the number of types maintained per robot was kept manageable by pruning out low-probability joint types. If the probability of a joint type, $P(\theta^t)$, was below some threshold, it was cut and the probability over the remaining joint types, $P(\Theta^t)$, renormalized. The allowable individual types for each robot were then taken to be those that existed in the set of remaining joint types.

At run time, a robot's true individual history may be represented by one of the types that was pruned. In this case, the best match to the true history is found amongst the retained types, and the robot acts as if this best match was the history of observations and actions that occurred instead. In BaGA, this match was originally made using the Hamming distance. There are better ways to define the best match. A simple one that works well is to look at *reward profiles*: the reward profile for an individual type θ_i^t is the vector $r^{\theta_i^t}$ whose elements are:

$$r_a^{\theta_i^t} = E(u(a, \theta^t)|\theta_i^t). \tag{4.1}$$

That is, $r_a^{\theta_i^t}$ is the robot's belief about how much reward it will receive, both now and in the future, when the team performs the joint action a, given its history as represented by θ_i^t and the utility function u. This expectation is calculated using the conditional probability distribution $P(\Theta^t|\theta_i^t)$ induced over the set of joint types by θ_i^t. Similarly, if there is a true history h_i^t for which there is no associated θ_i^t because of low-probability pruning, its reward profile $r^{h_i^t}$ can be found. This approach, however, requires more work as the probability distribution $P(\Theta^t|h_i^t)$ must be calculated.

Given the reward profiles for the true history h_i^t and some candidate match θ_i', the two are compared using their *worst-case reward difference*:

$$\max_{a \in A} |r_a^{h_i^t} - r_a^{\theta_i'}| \tag{4.2}$$

and the match that has the smallest worst-case reward difference from the true history selected. At run time, the robot executes the action given by $\pi_i^t(\theta_i^{'})$ if θ_i' is the best match according to this criterion. Reward profiles are used because a history should only be matched to a type for which the same action would be selected now and in the future. In comparison, directly comparing the expected rewards of types (i.e., the reward generated by the best individual action to take for each type) does not meet this requirement because different actions can lead to the same payoffs. For example, in the *Lady and the Tiger* problem, the types $\{$*listen*, *hear-left*, *listen, hear-left*$\}$ and $\{$*listen*, *hear-right, listen, hear-right*$\}$ both lead to an agent opening a door. The immediate expected value of opening the (best) door is the same for both types, and the expected future reward is also the same because, in both cases, the position of the tiger will be reset. The actual door to open, however, is very different. If an agent's true history is $\{$*listen*, *hear-left*, *listen, hear-left*$\}$, and it uses only expected reward to match that history to one of these two types, then the agent could match its history to the wrong one and end up opening the left door, even though it should open the right door. By comparing reward profiles, however, this difference would be identified.

4.1.1 Limitations of Low-Probability Pruning

Low-probability pruning is a very simple way to reduce the number of types maintained at each timestep of the BaGA algorithm. It requires no additional steps to the basic algorithm given in Algorithm 3.2, which already includes a *pruningThreshold* parameter. This kind of pruning can be effective in small domains such as the *Lady and The Tiger* because relatively few joint histories have to be pruned to keep computation manageable; however, it is a crude approximation to pretend that a history's importance for planning is proportional to its probability. When there are a large number of histories that each have a low probability of occurring but together represent a substantial fraction of the total probability mass, low-probability pruning can remove all of them and cause a significant negative change in the predicated outcome of the game.

4.2 Clustering

Instead of blindly removing low-probability histories, it is necessary to make sure that each history that is pruned can be adequately represented by some other similar history that is retained in the set of types. That is, histories need to be clustered into groups that have similar predicted outcomes, and then a representative history for each group used to build up the type space for each robot. If done properly, the probability distribution over these clusters should be an accurate representation of the true distribution over the original histories. As the experimental results will show, making sure to retain a representative type from each cluster of observation histories can yield a substantial improvement in the reward achieved by the team over retaining only higher probability types. In fact, improvement is seen even in runs in which the robots' true histories were not pruned: the performance of the team depends not just on what the robots have seen but also on how well they reason about what they might have seen.

At each timestep t, the BaGA-Cluster algorithm starts with the set of joint types, Θ^{t-1} and, using the associated joint history represented by each θ^{t-1}, constructs all one-step extensions by appending each possible joint action and observation. From these joint histories of length t, all possible individual histories for each robot are identified. Clustering is then performed on these individual histories to produce a final set of history clusters C_i^t for step t. Each history cluster c_i^t is represented by an exemplar history h_i^t (generally the individual history within the cluster with highest probability), and this set of representative individual histories becomes the type space Θ_i^t for each robot. Unlike low-probability pruning, clustering operates on individual histories rather than joint types. This way, each robot can determine, without communication, to which cluster its true history belongs. The corresponding representative joint histories, or joint-type space Θ^t, is the cross product of all of the exemplars for each history cluster.

Once clustering has been completed, a reduced Bayesian game is constructed using those histories that now make up Θ^t. It is solved to find a policy for each robot that assigns an action to each of its history clusters. As with low-probability pruning, at run time, worst-case reward difference is used to match the true history of a robot to one of its clusters and select an action (with a quick, first pass done to see if a robot's true history directly matches one of the exemplar histories exactly).[1]

There are many different types of clustering algorithms (Everitt et al., 2001), but in this thesis a version of agglomerative clustering is used. In BaGA-Cluster, each history starts off in its own cluster and then similar clusters are merged together until a stopping criterion is met. The final set of clusters becomes the type space Θ_i^t for the robot. Similarity between clusters is determined by comparing the reward profiles either of representative elements from each cluster (i.e., its exemplar history) or of the clusters as a whole. The reward profile for a cluster c_i, r^{c_i} is defined by a straightforward generalization of Equation 4.1:

$$r_a^{c_i} = E(r_a^{h_i} | h_i \in c_i) = E(u(a,h)|c_i).$$

If each cluster c_i is initialized to contain only one history h_i then $r^{c_i} = r^{h_i}$ and $P(c_i) = P(h_i)$. This initialization allows the reward profile of the larger cluster $c_i \cup c_i'$ to be calculated as:

$$r_a^{c_i \cup c_i'} = [P(c_i) r_a^{c_i} + P(c_i') r_a^{c_i'}]/[P(c_i) + P(c_i')].$$

Worst-case reward difference, as defined in Equation 4.2, could be used for determining the similarity two clusters. However, because there is information about the prior probability of both clusters, i.e., $P(c_i)$ and $P(c_i')$, *worst-case expected loss* is used instead (the probability

[1] Alternatively, the algorithm could keep track of all histories that are contained within each cluster and use that information to match a robot's true history to a cluster at run time. However, in order to keep this matching process a simple look up, the set of histories making up each cluster would have to be propagated forward for each timestep. While there would still be computational savings from using only the exemplars for building and solving the Bayesian game at each timestep (and for generating the next set of clusters), maintaining the set of all possible individual histories and their associated clusters requires $O(|A_i||Z_i|)$ operations for each of the $O(|Z_i|^{t-1})$ possible individual histories of each robot.

Algorithm 4.1: High-level overview of the BaGA-Cluster algorithm. Elements shown in bold are those that represent changes from the basic BaGA algorithm of Chapter 3.

```
Construct all possible joint histories from the set of individual **exemplar** histories given by $\Theta_i^{t-1}$ and
  the policy $\pi^{t-1}$. Update the probability distribution over the set.
**For each robot construct the set of all individual histories $H_i^t$ and their probability distribution.**
**Cluster together all individual histories with similar expected reward profiles until no more
  clustering can be done.**
Construct and solve the corresponding Bayesian game, $BG^t$, **using the final clusters, $C_i^t$, as the type
  space $\Theta_i^t$.**
Find the best match to the true history $h_i^t$ **from the set $C_i^t$. The robot's matching type $\theta_i^t$ is the type
  representing $c_i^t$.**
Implement the action given by the solution, $a_i^t = \pi_i^t(\theta_i^t)$.
```

of a robot's true observation history is not available at run time and so worst-case reward difference is still used for matching a true history to a cluster). Worst-case expected loss between two histories h_i and h_i' is defined to be:

$$\max_{a\in A}\left([P(h_i)|r_a^{h_i} - r_a^{h_i\cup h_i'}| + P(h_i')|r_a^{h_i'} - r_a^{h_i\cup h_i'}|]/P(h_i\cup h_i')\right).$$

Worst-case expected loss between two clusters is defined analogously in terms of $r_a^{c_i\cup c_i'}$:

$$\max_{a\in A}\left([P(c_i)|r_a^{c_i} - r_a^{c_i\cup c_i'}| + P(c_i')|r_a^{c_i'} - r_a^{c_i\cup c_i'}|]/[P(c_i) + P(c_i')]\right).$$

Worst-case expected loss is a good measure of the similarity of the reward profile for two clusters because it indicates how much loss in reward one could expect by merging the clusters together. As with histories, it is important to determine if two clusters have similar reward profiles (rather than just similar expected reward) because robots will end up performing the same action for each element of the cluster. Furthermore, augmenting a comparison of expected reward with a check to make sure it was generated by the same action is not sufficient. The expected value of non-optimal actions can give an indication of whether or not there are some other differences between the histories worth preserving, especially as the quality of the heuristic used for calculating the future expected value of actions in u degrades. A high-level overview of the clustering algorithm is shown in Algorithm 4.1.

Two different algorithms for selecting and merging pairs of clusters were considered. These algorithms are low-probability clustering and minimum-distance clustering. Low-probability clustering is designed to be similar to the low-probability pruning algorithm, while minimum-distance clustering is designed to produce a better clustering at a higher computational cost.

4.2.1 Low-Probability Clustering

In low-probability clustering, the set of clusters (each one containing a single history) is ordered randomly and then a single pass is made through the list of clusters. Each cluster is tested to determine whether its probability is below a threshold; if so, it is removed from the list and merged with its nearest remaining neighbour as determined by the worst-case expected loss between their exemplar histories. The neighbour's exemplar history stays the same, and its probability is incremented by the probability of the cluster being merged in.

This clustering method relies on randomization, so the synchronization of the random number generators of the robots is crucial to ensure that each robot performs the same series of clustering operations and arrives at identical final clusters. This synchronization requirement, however, is already met by the original BaGA algorithm because of the alternating-maximization algorithm's dependence on randomization.

Low-probability clustering has some similarities to low-probability pruning. In both approaches, a history is considered as a candidate for pruning or clustering if it is less likely to occur. As a result, a robot's true history is much more likely to match one of the exemplar histories at execution time. Like low-probability pruning, low-probability clustering has the drawback that all low-probability histories must be matched to another cluster even if they are far away from all other candidates. Unlike pruning, however, it maintains a distribution over joint histories that more closely matches the original distribution. By randomly ordering the set of candidate clusters, it is also possible that individual histories that, on their own, have too low a probability to keep, will have other low-probability histories matched with them and therefore survive to the final set of histories.

4.2.2 Minimum-Distance Clustering

Like low-probability clustering, minimum-distance clustering does not operates on clusters of joint histories but rather groups of individual histories. In minimum-distance clustering, the two most similar clusters are repeatedly found and merged together. The similarity measure used is that of worst-case expected loss between the two reward profiles. The algorithm stops when the closest pair of clusters is too far apart, or when a minimum number of clusters is reached. By setting the maximum-allowable distance between clusters for which merging will still occur, a tradeoff between the amount of loss in reward one is willing to incur and computational speed (minimizing the number of clusters) can be made.

The exemplar history for each cluster is simply the highest probability individual history in that cluster. This representative element is only used for generating the set of all possible joint histories at the next timestep; cluster merging and matching of the true observation history to a cluster are both done using the clusters' reward profiles.[2] That is, at run time, each robot matches its true history h_i^t to the type with which it has the minimum worst-case reward difference:

$$\arg\min_{\theta_i^t \in \Theta_i^t} (\max_{a \in A} |r_a^{h_i^t} - r_a^{\theta_i^t}|)$$

where each type θ_i^t corresponds to a cluster c_i^t and $r^{\theta_i^t} = r^{c_i^t}$.

Unlike single-pass low-probability pruning, minimum-distance clustering is a multi-pass algorithm and therefore requires more computation time than low-probability pruning. It does, however, result in a more accurate representation of the original type space because outliers in reward-profile space, regardless if they are high or low probability, are still maintained. With respect to picking one of these two types of clustering, one must trade off between the speed of clustering with the quality of the resulting clusters. It is also possible to combine clustering with low-probability pruning by removing any joint history with probability less than some threshold from the set of all possible joint histories of length t before constructing the set of clusters.

Table 4.1 provides a comparison of low-probability pruning, low-probability clustering and minimum-distance clustering. The key difference between pruning and clustering is that the former operates on joint histories while the latter operates on the set of individual histories for each robot. BaGA-Cluster, therefore, requires each robot to cluster not only its own histories, but also those of its teammates in order to guarantee that each robot constructs and solves the same Bayesian game at timestep t.

[2] With respect to implementation, the matching process actually has a first pass in which each exemplar history is compared to the true observation history to see if they are identical. If this pass is unsuccessful, then the reward profile for the true observation history is generated and reward profile matching performed. Generating this reward profile, however, is expensive and so the success of the first pass is maximized by setting the exemplar of each cluster to be its highest probability individual. Theoretically, other alternatives, such as the history that minimizes intra-cluster expected loss, could be used.

Table 4.1: Comparison of the different methods for limiting the size of the joint-type space. Low-probability pruning, low-probability clustering and minimum-distance clustering are compared with respect to the space upon which they act, the parameter used to select an element of that space for pruning/clustering, the outcome of the operation, the complexity of the operation and, finally, the advantages and disadvantages of each approach. In all cases, the matching of the true observation history to a maintained history is done by finding the best match in reward-profile space.

Method	**Space**	**Parameter**	**Outcome**	**Complexity**	**Advantages**	**Disadvantages**
Low-Probability Pruning	Joint histories	Prior probability of joint history θ, i.e., $P(\theta)$ given $P(\Theta)$	Joint history not meeting parameter threshold is removed from space	No additional costs, occurs during generation of all possible joint histories	Fast	Poor approximation of original distributions, Low-probability events still have impact on expected performance
Low-Probability Clustering	Individual histories	Prior probability of individual history θ_i, i.e., $P(\theta_i)$ given $P(\Theta)$	Individual history not meeting parameter threshold is merged with nearest neighbour in reward space	Single pass once set of all individual histories created from set of joint histories	Single pass, Groups of low-probability histories that are close enough can cluster together and survive	Low-probability outliers in reward-profile space can be lost
Minimum-Distance Clustering	Individual histories	Maximum expected loss between two history clusters c_1 and c_2	If distance between two closest clusters is below parameter threshold, then clusters are merged	Multi-pass once set of all individual histories created from set of joint histories	Outliers in reward-profile space remain even if low probability, More intuitive parameter to set	Multi-pass

```
Algorithm 4.2: $BayesianGameWithClustering$
Input: $I, \Theta, A, P(\Theta), Z, S, T, R, O, randSeed$
Output: $\pi, \Theta', P(\Theta')$
begin
    $setSeed(randSeed)$
    for $a \in A, \theta \in \Theta$ do
        $u(a, \theta) \leftarrow heuristicValue(a, beliefState(\theta))$
    $\pi \leftarrow findPolicies(I, \Theta, A, P(\Theta), u)$
    $\Theta'$,$P(\Theta') \longleftarrow MinimumDistanceClustering(I, \Theta, A, P(\Theta), \pi, u, Z, S, T, R, O)$
end
```

4.3 Algorithms for Minimum-Distance Clustering

Algorithms for the implementation of minimum-distance clustering are shown in Algorithms 4.2, 4.3 and 4.4. As with BaGA, the outer loop of BaGA-Cluster is given by Algorithm 3.1; however, Algorithm 3.2, *BayesianGame*, is replaced with Algorithm 4.2, *BayesianGameWithClustering*. The function *matchToType* is also assumed to use reward-profile space matching rather than Hamming distance as it did in Chapter 3. In the algorithm *BayesianGameWithClustering*, a Bayesian game is constructed and solved using the dynamics of the overall POSG, a specific joint-type space Θ, and the probability distribution over that space $P(\Theta)$. The final step in the algorithm is to create the joint-type space and distribution over that space for the next timestep of the problem. In this case, minimum-distance clustering is used to create of this space.

Algorithm 4.3, *MinimumDistanceClustering*, generates all possible one-step extensions of each joint type $\theta \in \Theta$ and stores them in the set of proposed joint types Γ and proposed individual types Γ_i. A cluster c_x is then created for each of a robot's types $\gamma_i \in \Gamma_i$. The exemplar history for cluster c_x is set to γ_i and the cluster's reward profile r^{c_x} calculated (η is used in these calculations to represent a generic member of Γ). Then, Algorithm 4.4 is used to repeatedly identify the two closest clusters, c_1 and c_2, by finding the pair of clusters within set C_i with the smallest maximum expected loss that would occur if they were merged. If this maximum expected loss is smaller than a specified threshold, *threshold*, the two clusters are merged together. If c_1 has higher probability of occurring, then the exemplar history c_1 is adopted as the exemplar history of the merged cluster. Otherwise, the exemplar history of cluster c_2 is used. When this algorithm terminates, the exemplar histories in each of the remaining clusters and the probability distribution over them is copied into the set of final types Θ'_i and final joint types Θ'.

If low-probability clustering were to be used instead of minimum-distance clustering, Al-

Algorithm 4.3: *MinimumDistanceClustering*

```
Input: $I, \Theta, A, P(\Theta), \pi, u, Z, S, T, R, O$
Output: $\Theta', P(\Theta')$
begin
    $\Gamma \longleftarrow \emptyset$
    $\Gamma_i \longleftarrow \emptyset, \forall i \in I$
    for $\theta \in \Theta$,$z \in Z$, $a \in A$ do
        $\gamma \leftarrow \theta \cup z \cup a$
        $P(\gamma) \leftarrow P(z, a|\theta)P(\theta)$
        $\Gamma \leftarrow \Gamma \cup \gamma$
        $\Gamma_i \leftarrow \Gamma_i \cup \gamma_i$,$\forall i \in I$
    /*initializing clusters */
    for $i \in I$ do
        $x \leftarrow 0$
        $C_i \leftarrow 0$
        for $\gamma_i \in \Gamma_i$ do
            $c_x \leftarrow \gamma_i$
            $P(c_x) \leftarrow \sum_{\eta \in \Gamma, \eta_i = \gamma_i} P(\eta)$
            for $a \in A$ do
                $r_a^{c_x} \leftarrow \sum_{\eta \in \Gamma} P(\eta|\gamma_i)u_i(a, \eta)$
            $C_i \leftarrow C_i \cup c_x$
            $x \leftarrow x + 1$
    /*performing clustering */
    for $i \in I$ do
        while $minDist < threshold$ do
            $c_1$,$c_2$,$minDist \leftarrow TwoClosestClusters(C_i, r)$
            if $P(c_1) < P(c_2)$ then
                $P(c_1) \leftarrow P(c_1) + P(c_2)$
                for $a \in A$ do
                    $r_a^{c_1} \leftarrow (P(c_1)r_a^{c_1} + P(c_2)r_a^{c_2})/(P(c_1) + P(c_2))$
                $C_i \leftarrow C_i - c_2$
                $P(c_2) \leftarrow 0$
                for $c_{-i} \in C_{-i}$ do
                    $P(c_{-i} \cup c_1) \leftarrow P(c_{-i} \cup c_1) + P(c_{-i} \cup c_2)$
                    $P(c_{-i} \cup c_2) \leftarrow 0$
            else
                $P(c_2) \leftarrow P(c_2) + P(c_2)$
                for $a \in A$ do
                    $r_a^{c_2} \leftarrow (P(c_1)r_a^{c_1} + P(c_2)r_a^{c_2})/(P(c_1) + P(c_2))$
                $C_i \leftarrow C_i - c_1$
                $P(c_1) \leftarrow 0$
                for $c_{-i} \in C_{-i}$ do
                    $P(c_{-i} \cup c_2) \leftarrow P(c_{-i} \cup c_1) + P(c_{-i} \cup c_2)$
                    $P(c_{-i} \cup c_1) \leftarrow 0$
    $\Theta' \leftarrow C$
    $P(\Theta') \leftarrow P(C)$
end
```

Algorithm 4.4: $TwoClosestClusters$

```
Input: $C, r$
Output: $c_1, c_2, minDist$
begin
    $minDist \leftarrow 0$
    for $c_x \in C$ do
        for $c_y \in C$ do
            $expectedLoss \leftarrow 0$
            for $a \in A$ do
                $r_a^{c_x \cup c_y} \leftarrow (P(c_x)r_a^{c_x} + P(c_y)r_a^{c_y})/(P(c_x) + P(c_y))$
                $expectedLoss \leftarrow \max\{expectedLoss, P(c_x)|r_a^{c_x} - r_a^{c_x \cup c_y}| + P(c_y)|r_a^{c_y} - r_a^{c_x \cup c_y}|\}$
            if $expectedLoss < minDist$ then
                $minDist \leftarrow expectedLoss$
                $c_1 \leftarrow c_x$
                $c_2 \leftarrow c_y$
end
```

gorithm 4.3 would be modified to perform a single pass through the set of possible clusters C_i of each robot. If a cluster c_x is below the pruning threshold, then its closest match c_y is found in the set C_i. Any probability mass is transferred to c_y and c_x is removed from C_i. Low-probability pruning occurs during the construction of all one-step extensions of the joint types of timestep t. If a proposed new joint type γ has probability lower than the threshold, it is simply not added into the set Γ.

4.3.1 Computational Complexity

All three ways of reducing the number of types requires the enumeration of all possible one-step extensions to the joint-type space of the previous timestep, which has a cost of $|\Theta^{t-1}||Z||A|$ or an upper bound of $O(|\Theta^{t-1}|\big(|Z_*||A_*|\big)^n)$ (reminder, the subscript $*$ denotes the biggest set of any one robot). Low-probability pruning does not require additional computation, as it prunes during the construction of Γ. Clustering, however, does.

Minimum-distance clustering requires the reward profile of each cluster to be calculated. As each cluster is initialized with one of a robot's possible types, each robot starts with $|\Gamma_i| = |\Theta_i^{t-1}||Z_i||A_i|$ clusters. The algorithm must then iterate over all possible types in Γ_{-i} in order to calculate the reward profile for a cluster leading to an overall cost of $|\Gamma_*|^n$. Each iteration of clustering itself requires $O\big(|C_*|^2\big)$ operations to find the two closest clusters. On the first iteration, $|C_*| = |\Gamma_*|$ and on the k-th iteration $|C_*| = |\Gamma_*| - k + 1$. In the worst case, there will be only one final cluster and so $|\Gamma_*| - 1$ iterations will be required. As a result, the upper bound on the cost of minimum-distance clustering is $|\Gamma_*|^n + n\big(\sum_{k=1}^{|\Gamma_*|-1}(|\Gamma_*| - k)^2\big)$ which is $O\big(|\Gamma_*|^n + n|\Gamma_*|^3\big)$.

Low-probability clustering also requires the reward profile of clusters to be calculated; however, only a single pass will done through the set of clusters for each robot. This procedure results in a cost of $O\big(|\Gamma_*|^n + n|\Gamma_*|^2\big)$.

4.3.2 Additional Issues

It is possible to construct problems in which there will be two clusters with identical (or nearly identical) reward profiles that should not be merged together. This situation would occur if the two clusters lead to very different conditional probability distributions over teammates' clusters because it suggests that, in the future, differences between the two clusters need to be preserved. For example, consider the case where Robot 1 has two clusters, *a* and *b,* and Robot 2 has two clusters *c* and *d*. Clusters *a* and *b* have identical reward profiles; however, while the joint types $\{a,c\}$ and $\{b,d\}$ are valid, the joint types $\{a,d\}$ and $\{b,c\}$ are not because they are made up of individual types that are inconsistent with each other. For example, in *Robotic Tag* a joint type in which the robots observe the opponent in different cells or observe a teammate in a cell that does not correlate to the teammate's true position is not valid. A validity vector can be defined for each cluster in which a one or a zero is assigned to each possible set of teammates' clusters indicating whether or not the robot's current cluster results in a valid or non-valid joint type in combination with them.

Generally, non-valid joint types do not exist in the set of all possible joint types because they have zero probability of occurring. It is only during the clustering process that one starts to consider them because the final set of joint types is the cross-product of the final set of clusters for each robot. So, if clusters a and b are merged together, because their reward profiles are the same, then (assuming that a becomes the exemplar of the cluster) both $\{a,c\}$ and $\{a,d\}$ would have to be considered valid joint types or some information will be lost. Clearly, the solution is to not merge a and b, leading to to an additional check during the clustering process in which, if two clusters do not have the same validity vector, then they cannot be a candidate for merging no matter how similar their reward profiles are.

This problem is actually not just limited to whether or not two clusters lead to similar validity vectors, but whether or not the two clusters lead to similar conditional probability distributions over joint types. However, one can think of these validity vectors as a thresholded approximation of these probability distributions. To fully address this issue, agglomerative clustering could be replaced with divisive clustering in which clusters are repeatedly split if elements within the clusters lead to different reward profiles or different probability distributions over the clusters of others.[3] Divisive clustering would require multiple passes, not just within the set of clusters for each robot but also over the set of robots because splitting one robot's cluster may lead to splits in another robot's clusters.

It should be note that in the types of problems examined in this thesis, clusters with similar reward profiles generally do have similar distributions over the clusters of others. Usually, unless a problem has been specifically constructed to highlight this issue, differences in probability distributions translate into differences in expected reward and reward profiles. For this reason, agglomerative clustering has been used, with the additional validity vector check for the MABC and robotic tag domains.

4.4 Experimental Results

In order to test the effectiveness of BaGA-Cluster, the algorithm was implemented in two domains: the *Lady and the Tiger* and the *Multiple Access Broadcast Channel* problems of Section 3.4. In this section, the performance of policies using low-probability pruning, low-probability clustering and minimum-distance clustering is analyzed.

4.4.1 Lady and The Tiger

Table 4.2 shows the performance of BaGA and BaGA-Cluster for a 10-step version of the *Lady and the Tiger*. In all cases, the *simple-heuristic* of Section 3.4.1.2 was used to evaluate the future expected value of actions and alternating maximization was run with 200 random restarts. BaGA was first run with no pruning or clustering to determine the total number of joint types maintained over the 10 steps of the problem. It took about 30 minutes on an Intel Pentium 4, 2.8 GHz machine to generate a solution that had an average reward of 10.68 to the team. In Table 4.2, the percentage of true histories retained indicates what fraction of the time an agent's true history of observations was present in its set of types. As the low-probability pruning threshold is raised from 0.000001 to 0.001, it can be seen that removing too many joint types from consideration negatively impacts performance. Negative

[3] This type of clustering can be thought of as an extension of the work on model minimization in MDPs (Dean & Givan, 1997) and POMDPs (Pineau et al., 2003b) to Bayesian games because it groups together types with the same action payoffs and induced probability distributions over the types of others.

outcomes occur even if the actual observations of the agents always match exactly to one of the retained histories such as the case where the threshold is 0.000005. As previously discussed, this situation is due to low-probability events that do not necessarily occur in practice, but can still impact decision making.

If low-probability clustering is applied, many fewer histories can be maintained without compromising performance. For example, low-probability clustering with a threshold of 0.05 (i.e., a type is pruned if it occurs with probability less than 0.05) results in about 2% of the total number of possible joint types. Each agent's true history of observations can be found amongst the retained types about 70% of the time with reward profile matching used the other 30%. The resulting performance is still equivalent to that of the full algorithm, but at a fraction of the computational cost. If the probability threshold is set too high, however, performance does start to degrade. The distribution over joint types is no longer being approximated well because histories that are too different are being aliased together. Minimum-distance clustering is able to avoid this problem at a higher level of clustering because it lets the system cluster more histories together while still representing the joint-history space well. Using BaGA-Cluster, the total number of joint types can be dropped to as low as 109, with only 55% of the true observation histories appearing as exemplars in the type space, without loss in performance. An additional advantage of minimum-distance clustering is that its parameter, maximum-allowable expected loss, is easier to interpret than the probability threshold used in low-probability clustering. For example, if the minimum-distance threshold is set to 0.1 it means that one is willing to accept up to 0.1 per timestep in lost reward in return for savings in computation time.

With this domain, it is not possible to see that low-probability clustering, for the same number of candidate types, runs faster than minimum-distance clustering; however, in the tag problems looked at in Chapter 6, it is indeed the case.

In Figure 4.1, an example of the computational and space savings of BaGA-Cluster is shown for a 6-step version of the *Lady and the Tiger*. At the first timestep, each agent only has one possible history, and therefore one cluster, because it has no observations about the world. After performing a *listen* action, the agent now has two possible observations histories: $\{$*listen*, *hear-left*$\}$ and $\{$*listen*, *hear-right*$\}$. Again, no clustering can be done because these two histories have very different reward profiles. On the third timestep, however, some histories can be clustered together with no loss of expected reward. Because the agent knows that its teammate has listened on the first two timesteps (from π^1 and π^2, which only include *listen* actions), the only important piece of information for decision making is whether or not it heard the same observation twice and, if so, which one observation it was.

Table 4.2: Clustering and performance results for a 10-step version of the *Lady and the Tiger* problem. Results are averaged over 100000 trials with 95% confidence intervals shown. The number of joint histories is the sum of all of the joint histories maintained over the 10 steps. Policy computation time is the percentage of time taken to compute a solution relative to the first condition in which all histories are maintained (a running time of 199018*ms*). The % of true histories retained is the percentage of steps in which the true agent's history is present in the set of retained history. Performance is not dependent on this value but rather on how well the entire set of possible histories is approximated. Timing experiments were performed on a Intel Pentium 4, 2.80 GHz machine.

Condition	Reward	# Joint Types	Computation Time	% True History Retained
All possible joint histories maintained	10.68 ± 0.10	231205	100.00%	100.0%
Low-prob. pruning with cutoff of 0.000001	10.66 ± 0.10	74893	56.28%	100.0%
Low-prob. pruning with cutoff of 0.000005	9.89 ± 0.13	34600	51.98%	100.0%
Low-prob. pruning with cutoff of 0.001	5.32 ± 0.26	1093	1.54%	93.8%
Low-prob. clustering with threshold 0.01	10.69 ± 0.09	7715	4.81%	91.3%
Low-prob. clustering with threshold 0.05	10.68 ± 0.10	563	0.25%	71.2%
Low-prob. clustering with threshold 0.1	6.38 ± 0.21	201	$< 0.10\%$	63.3%
Min.-distance clustering with max. loss 0.01	10.58 ± 0.10	335	$< 0.10\%$	56.4%
Min.-distance clustering with max. loss 0.1	10.72 ± 0.09	177	$< 0.10\%$	56.1%
Min.-distance clustering with max. loss 0.5	10.69 ± 0.10	109	$< 0.10\%$	55.1%

If the agent hears two different observations, it does not matter at this point in which order those observations were made, and so the histories {*listen*, *hear-left*, *listen, hear-right*} and {*listen*, *hear-right*, *listen, hear-left*} are grouped together.

For timestep 4, rather than generate all possible one-step extensions of the original four histories from timestep 3, only those extensions of the final set of clusters for timestep 3 need to be propagated forward. So, instead of starting the next step of clustering with all of the eight possible histories of length four, there are only six candidates. If a door is opened, as it is with four of the possible types at timestep 4, the problem is reset, so the observation received by the agent is irrelevant. This situation allows, without any loss in expected reward, for all of those types to be grouped together. Furthermore, regardless of what the teammate's type is, any one of those four types result in the same joint belief. The last two candidate types cannot be grouped together because they lead to different joint beliefs and therefore different reward profiles.

On the fifth timestep no clustering can be done, although clustering on previous timesteps has reduced the number of types to six from the sixteen possible histories of length five. What is perhaps not obvious is why the types {*listen, hear-right, listen, hear-left, listen, hear-right, listen, hear-left*} and {*listen, hear-right, listen, hear-left, listen, hear-left, listen,*

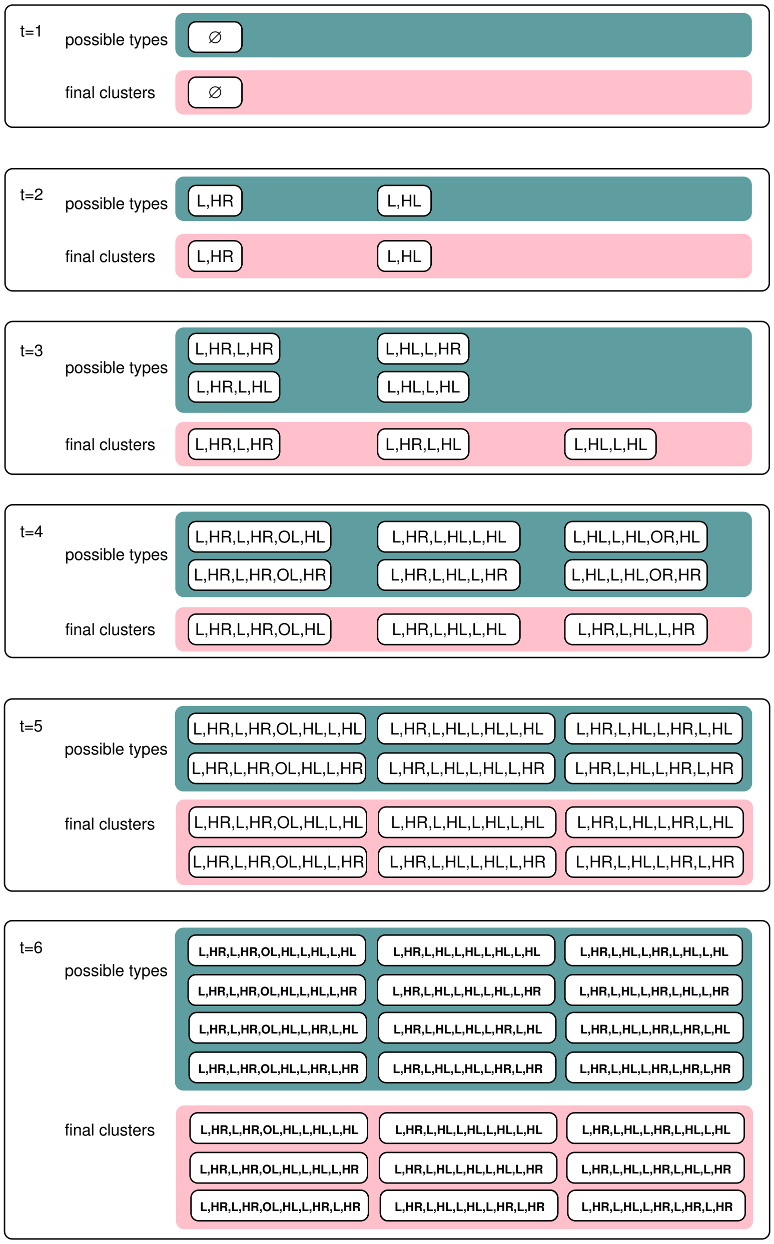


Figure 4.1: Clustering example for a 6-step version of the *Lady and the Tiger*. These are the clusters that are generated for each timestep during the generation of the policy tree shown in Figure 3.3, with the cluster exemplar shown in each case. In order to save space, the observation and action set are shown as <HL,HR> and <L,OL,OR> respectively with $\emptyset$ as the empty observation set. Without clustering, the size of the type space is $|\Theta_i^t|$ ={1,2,4,8,16,32} and with clustering it is $|\Theta_i^t| = \{1, 2, 3, 3, 6, 9\}$.

hear-right} have not been grouped together; after all, each one has two instances of each possible observation and at timestep 2, observation order was found not to matter. On this timestep, however, order does matter because these two types lead to very different beliefs when paired with a teammate's type in which a door is opened on the third timestep. In this case, the only relevant observation of the tiger's current position is the last observation, which is different in the two cases. However, two types of length 5 that do have the same observation on the last timestep are also not grouped together: if either is paired with a type in which the teammate does not open a door, then all observations can be valid and a different joint belief will occur. These differences in joint beliefs lead to differences in reward profiles, which prevent the clustering from being done.

By the last timestep, there are only twelve surviving candidate types rather than the 32 possible histories of length six. These types can be grouped into nine clusters as now, regardless of whether paired with a type in which a door is opened or not, the specific order of the last two observations no longer matters.

4.4.2 Multiple Access Broadcast Channel

Table 4.3 shows the performance of BaGA-Cluster for various conditions of the *Multiple Access Broadcast Channel* problem. In all cases, the *full-cooperative* heuristic of Section 3.4.2 was used to evaluate the future expected value of actions and alternating maximization was run with 20 random restarts. While *no-communication* was found to be a slightly more accurate heuristic for utility evaluation, it was not used for BaGA-Cluster because of the prohibitive cost of precalculating the required values for $T < 9$ that would have been necessary for the problems with time horizon 10. As the purpose of this experiment is to analyze how the performance of BaGA-Cluster changes as the number of joint types is decreased, the specific heuristic used is not as important as is the consistent use of the same heuristic.

In Table 4.3, low-probability pruning with a threshold of 0.000001 is compared to that with a threshold of 0.01. While low-probability pruning with the higher threshold is able to generate solutions faster, as evidenced by the smaller number of total joint types maintained by BaGA, those solutions are not as high quality, because the remaining joint types are no longer representative of the true distribution over possible joint types. This effect can especially be seen in the case of $n = 4, T = 10, p_i = 0.4$ in which, for about 85% of trials, low-probability pruning with a threshold of 0.01 resulted in all possible joint types being pruned before the final timestep. This result occurs when, based upon the joint types

Table 4.3: Comparison of different clustering results for MABC using the *fully-cooperative* utility function. Performance is averaged over 10000 trials with 95% confidence intervals shown. The average sum of joint types over each timestep is shown for BaGA with 95% confidence intervals. Pruning refers to low-probability pruning with a cutoff as shown while clustering refers to minimum-distance clustering with a maximium-allowable expected-loss threshold as shown. If memory restrictions prevented the calculation of a solution for a certain clustering condition, it is indicated with a "–".

Parameters	**pruning, cutoff 0.000001**		**pruning, cutoff 0.01**		**clustering, max. loss zero**		**clustering, max. loss 0.01**	
	Avg. Reward	$\sum \lvert\Theta^t\rvert$	**Avg. Reward**	$\sum \lvert\Theta^t\rvert$	**Avg. Reward**	$\sum \lvert\Theta^t\rvert$	**Avg. Reward**	$\sum \lvert\Theta^t\rvert$
$n = 2, T = 5,$ $p = \{0.4, 0.4\}$	2.96 ± 0.02	483.98 ± 0.02	2.97 ± 0.02	112.00 ± 0.00	2.95 ± 0.02	138.96 ± 0.02	2.96 ±0.02	20.00 ± 0.00
$n = 2, T = 5,$ $p = \{0.7, 0.3\}$	3.77 ± 0.02	367.89 ± 0.05	3.62 ± 0.02	114.00 ± 0.00	3.77 ± 0.02	76.00 ± 0.00	3.79 ±0.02	20.00 ± 0.00
$n = 3, T = 3,$ $p = \{0.4, 0.4, 0.4\}$	1.76 ± 0.02	272.00 ± 0.00	1.65 ± 0.02	58.00 ± 0.00	1.77 ± 0.02	196.92 ± 0.01	1.56 ± 0.02	24.00 ± 0.00
$n = 3, T = 3,$ $p = \{0.75, 0.5, 0.25\}$	2.40 ± 0.01	180.93 ± 0.16	2.37 ± 0.01	64.00 ± 0.00	2.39 ± 0.01	24.00 ± 0.00	2.40 ± 0.01	24.00 ± 0.00
$n = 3, T = 4,$ $p = \{0.4, 0.4, 0.4\}$	2.63 ± 0.02	763.99 ± 0.01	2.61 ± 0.02	96.00 ± 0.00	2.60 ± 0.02	458.36 ± 1.55	2.60 ± 0.02	32.00 ± 0.00
$n = 3, T = 4,$ $p = \{0.75, 0.5, 0.25\}$	3.32 ± 0.01	652.12 ± 0.07	3.24 ± 0.02	96.00 ± 0.00	3.31 ± 0.01	32.00 ± 0.00	3.30 ± 0.01	32.00 ± 0.00
$n = 3, T = 10,$ $p = \{0.4, 0.4, 0.4\}$	–	–	7.31 ± 0.03	256.02 ± 0.01	–	–	7.29 ± 0.03	80.00 ± 0.00
$n = 3, T = 10,$ $p = \{0.75, 0.5, 0.25\}$	–	–	8.38 ± 0.03	315.05 ± 0.02	8.63 ± 0.02	80.00 ± 0.00	8.62 ± 0.02	80.00 ± 0.00
$n = 4, T = 3,$ $p = \{0.4, 0.4, 0.4, 0.4\}$	1.89 ± 0.02	1024.00 ± 0.00	1.59 ± 0.02	50.00 ± 0.00	1.87 ± 0.02	868.02 ± 0.96	1.67 ±0.02	48.00 ± 0.00
$n = 4, T = 3,$ $p = \{0.8, 0.6, 0.4, 0.2\}$	2.59 ± 0.01	699.69 ± 0.08	2.55 ± 0.01	69.99 ± 0.00	2.60 ± 0.01	507.20 ± 0.20	2.59 ± 0.01	48.00 ± 0.00
$n = 4, T = 4,$ $p = \{0.4, 0.4, 0.4, 0.4\}$	–	–	2.02 ± 0.03	96.01 ±0.00	2.67± 0.02	4581.54 ± 3.80	2.33 ± 0.03	64.00 ± 0.00
$n = 4, T = 4,$ $p = \{0.8, 0.6, 0.4, 0.2\}$	–	–	3.38 ± 0.01	99.25 ± 0.02	3.46 ± 0.01	1494.59 ± 1.23	3.48 ± 0.01	64.00 ± 0.00
$n = 4, T = 10,$ $p = \{0.4, 0.4, 0.4, 0.4\}$	–	–	6.88 ± 0.10[a]	213.29 ± 0.08	–	–	6.72 ± 0.06	160.00 ± 0.00
$n = 4, T = 10,$ $p = \{0.8, 0.6, 0.4, 0.2\}$	–	–	8.70 ± 0.03	307.49 ± 0.09	–	–	9.11 ± 0.02	159.76 ± 0.03

[a] 85% of the runs using this condition actually resulted in all joint types being pruned and so results are averaged over the remaining 15%.

that are believed possible by BaGA, an 'impossible' observation is received by at least one agent. Of course, such an observation is not really impossible; it is the result of too many joint types being pruned from consideration. While the remaining 15% of the trials for this pruning threshold and parameters did result in viable solutions to the problem, this example underlines how important it is to maintain a good approximation of the true set of possible joint types.

If minimum-distance clustering is used instead of low-probability pruning, then a smaller number of joint types can be represented without adversely affecting the quality of the solutions. The MABC domain has an interesting property that, unless a maximum-allowable expected-loss threshold of zero is used, each agent has only two clusters per timestep: one to represent all types with a full buffer at that timestep and one to represent all those with an empty buffer (i.e., this type of clustering occurs whether the expected loss parameter is set to 0.01 as in Table 4.3 or 0.0001, etc.). This characteristic produces an extremely compact representation of the type space. It does, however, lead to some loss in performance, especially when all agents have the same new message rate, because different types with the same current buffer state for an agent can still lead to different probability distributions over the types of others based on the agent's previous buffer states. In turn, these differences in probability distributions lead to differences in expected reward profiles. Even with this loss in performance, the resulting clusters are a much more compact and intuitive representation of the true joint-type space of the problem than that achieved by low-probability pruning with a threshold of 0.01. With a single exception ($n = 3, T = 3, p_i = 0.4$), minimum-distance clustering with a threshold of 0.01 also outperforms this low-probability pruning condition.

If a maximum-allowable expected-loss threshold of zero is used, only those types with identical expected reward profiles will be merged, and no loss in performance is observed compared to low-probability pruning with a threshold of 0.000001. The amount of clustering done by BaGA-Cluster with this threshold varies based upon the new message rates of the agents. If one agents has a higher new message rate then more clustering occurs. Indeed, in the case of $n = 3, p = \{0.75, 0.5, 0.25\}$, the types are clustered into those with a full buffer on the current timestep and those with an empty buffer as they were for BaGA-Cluster with a maximum-allowable expected loss of 0.01. This type of clustering is why BaGA-Cluster with a maximum-allowable expected loss of zero can generate a solution to the $n = 3, T = 10, p = \{0.75, 0.5, 0.25\}$ scenario but not that of $n = 3, T = 10, p_i = 0.4$, which would require too many clusters to be computationally viable. While this BaGA-Cluster condition results in excellent performance, overall it uses a much higher number of clusters than minimum-distance clustering with a threshold greater than zero and so cannot be applied to the largest problems such as $n = 4, T = 10$. As such, the relatively small loss in reward generated by increasing the maximum-allowable expected loss seems a worthwhile tradeoff for the increase in computational and space savings. BaGA-Cluster with a maximum-allowable expected loss of 0.01 also has the benefit that the number of clusters per agent is time invariant unlike the other pruning and clustering conditions examined in this section. This invariance would allow it to be applied to problems of indefinite length.

4.5 Summary

In this chapter, the efficiency of BaGA was improved through the addition of history clustering. Rather than reason over the set of all possible histories, BaGA-Cluster groups together sets of histories that have the same, or similar, expected reward profiles. Policies for the robots are now conditioned on these clusters of histories. Computational savings are achieved because BaGA no longer has to reason over all possible histories but instead does so over a reduced number.

Three different methods were implemented for achieving this reduction: low-probability pruning, low-probability clustering and minimum-distance clustering. While each method has its own advantages and disadvantages, minimum-distance clustering generates the set of clusters that best represents the overall history space.

The computational savings of clustering is phenomenal. BaGA-Cluster is able to generate solutions to *Lady and The Tiger* in less than 0.1% of the processing time required by BaGA, without any loss in performance. Similar results are observed in MABC.

Chapter 5

Extending BaGA to Robot Teams with Communication

In this chapter, communication is used to both improve the performance of a robot team and to further reduce the computational costs of the BaGA algorithm. While the BaGA algorithm, as presented in Chapters 3 and 4, solves problems in which no communication is available to the team, it can be applied to a wider range of communication problems. The previous focus was on problems without any communication; the addition of communication can only improve performance of the team as it allows the robots to synchronize their observation histories. Aside from fixed communication policies, the BaGA algorithm can also be used to generate run-time communication policies because it helps to determine both when to communicate and what to communicate. In this chapter, a modification of BaGA-Cluster is presented in which a few, but critically placed, communication acts greatly improve the overall performance of the system at a fraction of the computational cost of the full POSG solution.

In order to make the most efficient use of bandwidth, the questions of when and what to communicate must be answered. Communication acts that improve the quality of decision making are desirable, while those that do not share important information between robots are not necessary. If communication policies are not efficient, then the overall system will suffer from scalability problems; while the current bandwidth might be sufficient to handle full communication between robots, increasing the number of robots will place too large a burden on the communication framework. A dynamic communication policy that makes run-time decisions in a principled manner is better able to exploit the gains in performance afforded by specific communication acts than a static policy.

5.1 Run-Time Communication Policies

The most obvious way to generate optimal communication policies for a POSG is to augment the action set A to include communication actions in addition to domain actions. These communication actions may take the place of domain actions (*or-communication*) or they can be chosen in combination with domain actions (*and-communication).* In this thesis, only *and-communication* is considered: that is, robots will always select a domain action at each timestep but are also free to communicate. At timestep t, robots first decide whether or not to communication and then, if any communication action should take place, all robots on the team are able to implement a domain action conditioned on the specific information received. Otherwise, robots will implement a domain action conditioned on their own observation history.

A globally optimal run-time communication policy can be found by building a POSG that represents all possible *and-communication* acts at each timestep and then solving that POSG exactly. The result is a policy tree (or conditional plan) for each robot that, given an observation history h_i^t, tells the robot whether or not to communicate and then tells it what action to take in response to each possible set of communicated information that the robot can receive from its teammates. This optimal policy allows for multiple robots to communicate at once or for no robots to communicate at all depending on what observations are actually made. By finding such a policy in a backward fashion (i.e., sub-game perfection), communication actions are only taken if they result in higher expected payoff to the team.

While the inclusion of communication actions into the action set seems a relatively straightforward way to generate communication policies, it is also not feasible to find the corresponding globally optimal policy. POSGs are already computationally intractable; increasing the size of the action set to include communication actions only makes the problem harder. Furthermore, directly applying the BaGA-Cluster algorithm to such a POSG does not address the difficulties that arise through the increased action set. As discussed in previous chapters, the computational requirements of each Bayesian game are dominated by the number of histories (or clusters of histories) represented in the type space. The inclusion of communication actions via *and-communication* increases the total number of possible histories by a factor exponential in the number of agents because each history must now also include an instantiation of the possible information received from each teammate at timestep t (with a null placeholder, $\emptyset$, to indicate no communicated information received).[1]

[1] It is possible that there are domains in which the size of the type space will remain small enough to allow for BaGA-Cluster to be performed directly. In general, most of the histories generated at timestep t will not

Instead, the BaGA-Comm algorithm computes domain policies and the expected value of those policies both with and without communication. In this approach, robots no longer create universal communication policies (i.e., a communication policy conditioned on each possible history) but rather determine, at run time, if communication of their current observation history will make a difference in the performance of the team. If communication is warranted based upon a given criterion (e.g., increase in expected performance), then the robot transmits information to the rest of the team. Upon receiving any communicated information, each robot creates and solves a final Bayesian game that reflects any shared information. Section 5.2.1 discusses the complexity of using this BaGA-Comm approach, and compares it to using BaGA-Cluster to solve a POSG that includes all *and-communication* acts.

Finally, with all communication policies there are two questions that must be answered: what to communicate and when to communicate. Generally, the second question is easier to answer and in many implemented systems the communicated information is left overly inclusive in order to avoid leaving out any important information (e.g., Gutmann et al. (1999); Brusey et al. (2001); Emery et al. (2001); Khoo & Horswill (2002)). This approach is not the most effective use of bandwidth and the clustering version of the BaGA-Cluster algorithm offers a simple, yet elegant, solution to the problem of bandwidth utilization.

5.1.1 What To Communicate

The purpose of clustering is to group together sets of observation and action histories that lead to similar expected reward profiles when combined with the histories of other robots. That is, it identifies a set of groups that are distinct in some important way (e.g., expected reward profiles). If robots transmit the cluster (i.e., the identifying number of the type θ_i^t to which their observed history best matches), rather than the observed history itself, two problems are solved. First, transmitting an identifying type number takes less bandwidth than transmitting a history of all observations and actions made since the last communication act (especially as the length of the problem increases). It is guaranteed that a finite number will be used by limiting the maximum number of clusters that can be generated and therefore the size of the type space $|\Theta_i^t|$.

survive to timestep $t + 1$ because only those extensions that include the information actually communicated at timestep t will have positive prior probability. The problem is that each surviving history still has an exponential number of possible one-step extensions. Domains with a high level of additional constraints (e.g., knowing the position of all teammates) may still remain manageable.

Second, via its construction, each cluster encapsulates all information pertinent to decision making and allows the robots to ignore any aspects of history that are irrelevant. For example, consider the *Lady and the Tiger* problem in which an agent is deciding what to do two timesteps after it has opened a door. The observations it received before opening the door are now irrelevant to the decision-making process as they give no information about the current position of the tiger; only the observations made on a following *listen* action may have relevance.[2] As a result, when implementing the BaGA-Cluster algorithm all possible histories fall into only one of two clusters: the agent either received the observation *hear-left* or the observation *hear-right* after its most recent *listen* action.

By making use of the work already done by BaGA-Cluster, it is not necessary to decide what subset of an agent's observation history will lead to the largest information gain. Instead, the agent only transmits the information that makes a difference with respect to policy generation and expected reward: the cluster, or type, number. With the *Lady and the Tiger* problem, this approach allows a teammate to ignore the irrelevant information: what observations were made before the door was opened, and focus on the relevant information: the robot has since performed a *listen* action and heard observation *x*. This approach is not the only way to answer the question of what to communicate (for example see Rosencrantz et al. (2003) and Nettleton et al. (2003)), but it is a way to answer the question that leverages work already done by BaGA-Cluster.

Communication acts are assumed to be broadcast to the entire team and not just robot-to-robot. They do not, however, force a synchronization between the entire team: if robot i informs the team of its history's cluster, the rest of the team does not have to inform i of their histories' clusters.

5.1.2 When To Communicate

Informally, the BaGA-Comm algorithm has each robot find a policy for the team given communication is not possible and the policy for the team given everyone knows its true cluster. The robot then compares those two policies and communicates its cluster number if appropriate. BaGA with clustering is used to build and solve each of these two games.

In BaGA-Cluster, at timestep t, robots construct the set of all possible joint histories based on an exemplar history from each cluster at timestep $t-1$, the policy π^{t-1} and any information communicated at timestep $t-1$. Minimum-distance clustering is then performed

[2] In turn, their relevance depends on whether or not the teammate opened a door at the same time.

on each set of possible individual histories to construct the final set of clusters for each robot, C_i^t. A Bayesian game BG^t is constructed using the set C^t as its joint-type space Θ^t and a proposed policy π^t found. If no communication is permitted, a robot would match its observed history h_i^t to the most appropriate cluster in C_i^t and then implement the action given by its policy for the corresponding type, $a_i^t = \pi_i^t(\theta_i^t)$. This process is then repeated at timestep $t + 1$.

Instead, with BaGA-Comm, the robots decide if communicating with their teammates is beneficial. As before, each robot matches its observed history h_i^t to one of its clusters c_i^t in C_i^t. It then creates a new Bayesian game, BGC_i^t, using the set $\{c_i^t, C_{-i}^t\}$ as the joint-type space Θ^t. In this Bayesian game, robot i has only one possible type, its current cluster, while the other robots still have all of their possible clusters. The probability distribution over this subset of joint types is found by normalizing the probability distribution over the set of joint types in which i's component is equal to c_i^t. This game is smaller than the original Bayesian game and therefore faster to solve. The outcome of this game is a proposed policy, $\sigma^{i,t}$, for the entire team. A superscript of i has been added to indicate that each robot will construct a different game BGC_i^t and therefore find a different proposed policy $\sigma^{i,t}$.

After finding $\sigma^{i,t}$ each robot decides, based upon π^t and $\sigma^{i,t}$, whether or not to communicate. This communication decision can be based on many different factors but only four types of communication policies will be considered here. The first is a fixed communication policy in which robots communicate every x steps. This type of policy does not actually require the generation of both π^t and $\sigma^{i,t}$. Instead, each robot can wait to receive the type number of each of its teammates and then construct a Bayesian game consisting only of those types. In fact, this game is no longer a Bayesian game but a regular normal-form game because the joint-type space contains only one element.

For this fixed communication policy, if a robot has only one cluster and therefore type, there is no need to transmit that information as all the robots will already know its type. In practice, it means that the total number of communication acts for each robot under such a policy can be fewer than T/x where T is the total number of timesteps in the problem.

The first of the dynamic communication policies considered is that in which a robot decides to communicate if the difference between the expected value of the policies $\sigma^{i,t}$ and π^t, given the cluster its current observation history falls into, is greater than the cost of communication. If the robot's true history h_i^t falls into the cluster c_i^t, which is in turn represented by the type θ_i^t then this decision is equivalent to deciding if:

$$|E\big(u(\sigma_i^{i,t}(\theta_i^t), \theta^t)|\theta_i^t\big) - E\big(u(\pi_i^t(\theta_i^t), \theta^t)|\theta_i^t\big)| > commCost.$$

This expected-value-difference communication policy, EVD, is able to trade off the expected gain in performance to the team with the cost of communication. As with the fixed communication policy, nothing is communicated if the robot only has one cluster. While this robot does not have to construct BGC_i^t, all of its teammates will unless they too only have one possible type.

The second dynamic communication policy considered is that in which a robot communicates if it calculates a policy $\sigma^{i,t}$ that is different from the policy π^t for either itself or any of its teammates. In other words, communication is prompted if either $\pi_i^t(\theta_i^t) \neq \sigma_i^{i,t}(\theta_i^t)$ or $\pi_j^t(\theta_j^t) \neq \sigma_j^{i,t}(\theta_j^t)$, $\forall j \in -I$ and $\forall \theta_j^t \in \Theta_j^t$. This rule is a policy-difference, or PD, communication decision. It does not take into account the cost of communication but looks only at differences in actions. The motivation for the use of PD is that BaGA-Comm uses a heuristic to calculate the expected future value of policies: should that heuristic not be accurate enough to properly evaluate the true difference in expected value between policies $\sigma^{i,t}$ and π^t (and therefore determine if that difference is really greater than the communication cost), but still be accurate enough to maintain the proper ordering between the two (i.e., that $E(\sigma^{i,t}) \neq E(\pi^t)$), then this type of communication decision still makes appropriate decisions.

The final type of dynamic communication policy is one that looks at the computational costs associated with finding π^t and it can be used in combination with EVD or PD. If $|C_i^t|$, the final number of clusters for robot i at timestep t, is greater than some threshold k, then the robot communicates its best-matched cluster's number to the team. Because the time required to solve a Bayesian game is directly related to the number of different history clusters tracked for each robot, this decision generates communication policies that trade off bandwidth constraints with computation constraints.

Once all communication decisions have been made, those robots that have decided to communicate broadcast their current type to the team. In BaGA-Comm a robot must communicate information to the entire team and not just to a subset of its teammates in order to satisfy the assumption of this thesis that communication is a source of common knowledge. Because BaGA-Cluster requires that only common knowledge be used during the construction of the Bayesian game at each timestep, if only a subset of the robots know a piece of information, it cannot be used by them to modify the prior probability over the set of possible joint types.

Algorithm 5.1: High-level overview of the dynamic communication decision algorithm for BaGA-Comm. Elements shown in bold are those that represent changes from the BaGA-Cluster algorithm of Chapter 4. The criterion for communication used is an increase in expected reward (EVD).

```
1: Construct all possible joint histories from the set of history clusters given by $\Theta_i^{t-1}$, the policy $\pi^{t-1}$, **and any communicated information** at timestep $t-1$. Update the probability distribution over the set.
2: For each robot, construct the set of all individual histories $H_i^t$ and its probability distribution.
3: Cluster together all individual histories with similar expected reward profiles until no more clustering can be done.
4: Construct and solve the corresponding Bayesian game, $BG^t$, using the final clusters, $C_i^t$, as the type space $\Theta_i^t$.
5: Find the best match to the true history $h_i^t$ from the set $C_i^t$ and **calculate the expected reward for the cluster $c_i^t$.**
6: **Construct and solve a new Bayesian game, $BGC_i^t$, in which robot $i$ has only one type $\theta_i^t = c_i^t$ but all other robots have the type set given by $\Theta_{-i}^t = C_{-i}^t$.**
7: **If the expected reward for the new game (including communication costs) is greater than that found in Step 6, communicate $\theta_i^t$ to the rest of the team.**
8: **If any communication takes place, construct and solve a final Bayesian game, $BG'^t$, using communicated types and the modified probability distribution that the communicatd knowledge induces over the joint set $\Theta^t$.**
9: Implement the action given by the solution, $a_i^t = \pi_i'^t(\theta_i^t)$.
```

Provided that there is at least one communication act, each robot constructs a final Bayesian game BG'^t using a joint-type space based on the set $\{c_j^t, C_l^t\}$, where j is the set of robots that communicated their cluster number and l is the set of robots that did not. The solution to this game is the final policy π'^t and each robot executes its action $a_i^t = \pi'^t(\theta_i^t)$. If no new information is received by a robot, BG'^t does not have to be constructed, as the final policy π'^t is simply equal to the policy π^t. The process is then repeated at timestep $t+1$.

Algorithm 5.1 gives a high-level overview of this communication algorithm. With respect to implementation, this type of communication can be thought of as having two phases: each robot plans, waits a (short) set length of time to see if it will receive any new information and then either implements its original, no-communication policy or uses the new information to re-plan. If such a wait time is not feasible for the domain, then robots simply use their no-communication policies. The resulting run-time communication policies are robust to communication failures so long as the robots are able to detect that communication failed.[3] The robustness follows because the robots not receiving the communication will use π^t by default and the communicating robot, upon realizing the failure of its teammate to receive its message, can switch back to using π^t. The POSG framework allows the

[3] Examples of such a robust communication protocol can be found in Stone & Veloso (1998), Barbuceanu & Fox (1995) and Emery et al. (2002). Using these protocols, robots are able to detect failed communication acts. A similar system could be used in BaGA-Comm.

robots to continue to make use of their observations, even if communication fails.

5.2 Algorithms for Run-Time Communication

Algorithms 5.2 through 5.5 detail how BaGA-Comm finds run-time communication policies. At each timestep a Bayesian game is constructed and solved using BaGA-Cluster to generate a proposed policy π^t, and joint-type space for the next timestep Θ^{t+1}. Each robot then finds the best match to its current history, θ_i^t, in reward-profile space. Using θ_i^t, it then makes a communication decision ω_i^t. If ω_i^t is true, then robot i communicates θ_i^t to all the other robots in the team. Each robot maintains a vector, $\psi^{i,t}$, of any types that have been communicated to it and, upon receipt of any θ_j^t, each robot updates $\psi^{i,t}$ to include θ_j^t. Finally, if any robot communicated, then Algorithm 5.5 is used to find a new policy and joint-type space for the team that takes into account any communicated information. Each robot then implements the action given by policy π_i^t for type θ_i^t.

Algorithm 5.3 is responsible for making the communication decision and, as shown, does so using the policy-difference condition. It first constructs a set of joint types using the set of clustered types Θ^t passed into it, as well as the known type of robot i, θ_i^t (which is passed into Algorithm 5.3 as the parameter ψ_i). The new set of joint types, Γ, only includes those joint types that have θ_i^t as an element. η is used to represent a generic element of Γ during probability renormalization over the final set of joint types. Algorithm 5.4 is then used to find a proposed policy σ using Γ. *JustSolveBayesianGame* (Algorithm 5.4) is used rather than *BayesianGameWithClustering* because the proposed type space for the next timestep, Γ', is never going to be used by Algorithm 5.2 and so it does not need to be generated. If the policy σ is different than the one found without the team members knowing robot i's type, then i's decision is to communicate and so ω_i is set to true. Obviously, the PD condition can be replaced with any other run-time decision such as EVD or even a fixed communication decision such as communicating every x timesteps.

Finding the final policy is very similar to making the communication decision, only Algorithm 5.5 takes in an entire vector of any communicated individual types and uses all of them to construct a new set of possible joint types Γ^t. Given this set and the probability distribution over it, *BayesianGameWithClustering* is used to find a policy σ and the joint-type space Γ' for the next timestep. Finally, Algorithm 5.2 uses this information to set the joint-type space Θ^{t+1} equal to Γ' and policy π^t equal to σ.

Algorithm 5.2: $PolicyConstructionAndExecutionWithCommunication$

```
$I, \Theta^0, A, P(\Theta^0), Z, S, T, R, O$
Output: $r, s^t, \pi^t, \forall t$
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)
            $setRandSeed(rs^t)$
            /*set all known types to null */
            for $j \in I$ do
                $\psi_j^{i,t} \leftarrow \emptyset$
            $\pi^t, \Theta^{t+1}, P(\Theta^{t+1}) \leftarrow$
                $BayesianGameWithClustering(I, \Theta^t, A, P(\Theta^t), Z, S, T, R, O, rs^t)$
            $h_i \leftarrow h_i \cup z_i^t \cup a_i^{t-1}$
            $\theta_i^t \leftarrow matchToType(h_i, \Theta_i^t)$
            $a_i^t \leftarrow \pi_i^t(\theta_i^t)$
            $\omega_i^t \leftarrow CommunicateDecision(I, i, \theta_i^t, \Theta^t, A, P(\Theta^t), Z, S, T, R, O, \pi^t, rs^t)$
            if $\omega_i^t = true$ then
                /*broadcast $\theta_i^t$ to all other robots */
                for $j \in I$ do
                    $\psi_i^{j,t} \leftarrow \theta_i^t$
        for $i \in I$ do (in parallel)
            if $\exists i$ such that $\omega_i^t = true$ then
                $setRandSeed(rs^t)$
                $\pi^t, \Theta^{t+1}, P(\Theta^{t+1}) \leftarrow FindFinalPolicy(I, \psi^{i,t}, \Theta^t, A, P(\Theta^t), Z, S, T, R, O, rs^t)$
                $a_i^t \leftarrow \pi_i^t(\theta_i^t)$
        $s^{t+1} \leftarrow T(s^t, a_1^t, ..., a_n^t)$
        $r \leftarrow r + R(s^t, a_1^t, ..., a_n^t)$
end
```

5.2.1 Computational Complexity

These communication policies have two levels of approximation with respect to the optimal communication policy described at the start of Section 5.1. First, BaGA-Comm makes use of heuristics for calculating the future value of actions rather than backing up the optimal policy. Secondly, rather than each robot generating a Bayesian game approximating the t-th step of a POSG that models all possible communication and domain actions (and therefore all possible combinations of communication and no communication), each robot generates two Bayesian games: one with no communication factored in and one in which all its teammates know its cluster but it receives no information from them. Therefore, if the

Algorithm 5.3: $CommunicateDecision$

```
Input: $I, i, \psi_i, \Theta, A, P(\Theta), Z, S, T, R, O, \pi, randSeed$
Output: $\omega_i$
begin
    $\omega_i \leftarrow false$
    $\Gamma \leftarrow \emptyset$
    /*add all joint types to $\Gamma$ that have $\psi_i$ as an individual type for $i$ */
    for $\theta \in \Theta$ do
        if $\theta_i = \psi_i$ then
            $\gamma \leftarrow \theta$
            $P(\gamma) \leftarrow P(\theta)$
            $\Gamma \leftarrow \Gamma \cup \gamma$
    for $\gamma \in \Gamma$ do
        $P(\gamma) \leftarrow P(\gamma) / \sum_{\eta \in \Gamma} P(\eta)$
    /*note that $\Theta$, and therefore $\Gamma$, already clustered */
    $\sigma \leftarrow JustSolveBayesianGame(I, \Gamma, A, P(\Gamma), Z, S, T, R, O, randseed)$
    if $\sigma \mathrel{!}= \pi$ then
        $\omega_i \leftarrow true$
end
```

Algorithm 5.4: $JustSolveBayesianGame$

```
Input: $I, \Theta, A, P(\Theta), Z, S, T, R, O, randSeed$
Output: $\pi$
begin
    $setSeed(randSeed)$
    for $a \in A, \theta \in \Theta$ do
        $u(a, \theta) \leftarrow qmdpValue(a, beliefState(\theta))$
    $\pi \leftarrow findPolicies(I, \Theta, A, P(\Theta), u)$
end
```

Algorithm 5.5: $FindFinalPolicy$

```
Input: $I, \omega, \psi, \Theta, A, P(\Theta), Z, S, T, R, O, randSeed$
Output: $\sigma, \Gamma', P(\Gamma')$
begin
    $\Gamma \leftarrow \emptyset$
    /*add all joint types to $\Gamma$ that contain any individual types given by
      $\psi$ */
    for $\theta \in \Theta$ do
        $add \leftarrow true$
        $i = 0$
        while $add = true$ && $i < |I|$ do
            if $\psi_i$ != $\emptyset$ && $\theta_i$ != $\psi_i$ then
                $add \leftarrow false$
            $i \leftarrow i + 1$
        if $add$ then
            $\gamma \leftarrow \theta$
            $P(\gamma) \leftarrow P(\theta)$
            $\Gamma \leftarrow \Gamma \cup \gamma$
    for $\gamma \in \Gamma$ do
        $P(\gamma) \leftarrow P(\gamma)/\sum_{\eta \in \Gamma} P(\eta)$
    /*Need to find $\sigma$ but also generate set of types for next time-step */
    $\sigma, \Gamma', P(\Gamma') \leftarrow BayesianGameWithClustering(I, \Gamma, A, P(\Gamma), Z, S, T, R, O, randseed)$
end
```

performance of the team can only be improved by all robots communicating and not just one, that communication policy will not be generated.

The calculation of these run-time communication policies are, however, computationally efficient compared to the optimal communication policy: at worst three Bayesian games must be constructed and solved, BG^t, BGC_i^t and BG'^t. BGC_i^t is smaller than BG^t and BG'^t is guaranteed to be no larger than BG^t. The solution to BG^t is found using a sequence-form representation of the Bayesian game and alternating maximization. As described in Section 2.2.1, for an arbitrary Bayesian game, each robot will have $|A_i||\Theta_i|$ different possible sequences. If the utility function requires linear time to compute, then alternating maximization has computational complexity of $O(\big(|A_*||\Theta_*|\big)^n)$.

Solving BG^t has an upper bound on computational complexity of $O\big((|A_*||C_*^t|)^n\big)$ where $|C_*^t|$ the size of the largest set of clusters of any robot (in general $|C_*^t|$ is exponential in t).[4] Alternatively, if one tried to construct a Bayesian game representing a timestep of the POSG that includes all possible *and-communication* actions, then the number of possible

[4] Note that this formula is the computational complexity for solving BG^t and does not include the cost of finding the types/clusters for timestep $t + 1$. See Section 3.3.1 for more details.

sequences for each robot is increased by a factor exponential in the number of robots. This Bayesian game would actually be solved as a 2-step game: the resulting policy tells each robot whether to communicate its cluster or not, and then, based on whatever information it receives from the team, what domain action to do. As described in Section 2.1.5, such a multi-step game requires a sequence to be created and evaluated for every possible path through the 'game-tree'. Constraints are then used during policy construction to enforce the requirement that robot i must make the same communication decision for every sequence that includes the same cluster c_i^t. In this case, a single robot's path includes a communication and action decision for every possible $\{$cluster, information received from teammates$\}$ pair. If teammate *j* has $|C_j^t|$ possible clusters, it can either broadcast one of those cluster numbers or not broadcast any information leading to $\prod_{j\in -I}(|C_j^t|+1)$ different possible combinations of received information for i. As a result, robot i has an upper bound on its number of possible $\{$cluster, information received from teammates$\}$ pairs of $|C_i^t|(|C_*^t|+1)^{n-1}$. With two possible communication actions and $|A_i|$ domain actions, each robot has at most $(2|A_i|)[|C_i^t|(|C_*^t|+1)^{n-1}]$ different possible sequences, resulting in an upper bound on the computational complexity for finding the solution of $O\big((2|A_*|)^n|C_*^t|^{n^2}\big)$. This quantity is a factor of $2^n|C_*^t|^n$ greater than the computations performed by BaGA-Comm for the same timestep of the problem.

As a final note, the computational savings by BaGA-Comm's myopic approach does mean that robots cannot make use of 'negative information'. In the optimal communication policy, if a robot does not communicate then that information is used by the team to adjust the probability distribution over future possible observation histories (i.e., the team can ignore histories for which that robot would have communicated as it is common knowledge that they have not occurred). In order for BaGA-Comm to make similar use of this negative information, each robot would have to know whether or not each of its teammates would communicate for each one of their possible clusters (i.e., find a cluster-to-communication mapping). This approach would require each robot to not only build the game BGC_i^t to find its own communication decision, but to also build and solve the series of games BGC_j^t in which all robots know that robot j's history is in cluster c_j. Each robot then applies the communication criterion to BG^t and each BGC_j^t to come up with a cluster-to-communication policy for each of its teammates. So long as the algorithm is constructed such that each robot finds the same policies, the team can now incorporate both positive and negative information. Like before, if a robot communicates its cluster number at timestep t, then only those joint histories that include that cluster will have non-zero prior probability timestep $t+1$. However, if a robot does not communicate, then all joint histories that correspond to clusters for which it would communicate, will have zero prior probability at timestep $t+1$. While this approach would allow for negative information to be included, instead of just solving one game of the form BGC_i^t, each robot must now solve BGC_j^t for all its possible clusters and for all of its teammates' possible clusters. This increase raises the number of games to be solved from 1 to $O(|C_*^t|^n)$, bringing the complexity of such an algorithm closer to that of applying BaGA-Cluster to the POSG that includes *and-communication* acts. For this reason, negative information is not used in BaGA-Comm.

5.3 Experimental Results

In order to test the effectiveness of the dynamic communication policies generated by BaGA-Comm, the algorithm was tested in two domains: the *Lady and the Tiger* problem and the *Robotic Tag* problem of Section 3.4.3. In this section, the performance of both fixed and run-time communication policies in these domains is analyzed.

5.3.1 The Lady and The Tiger

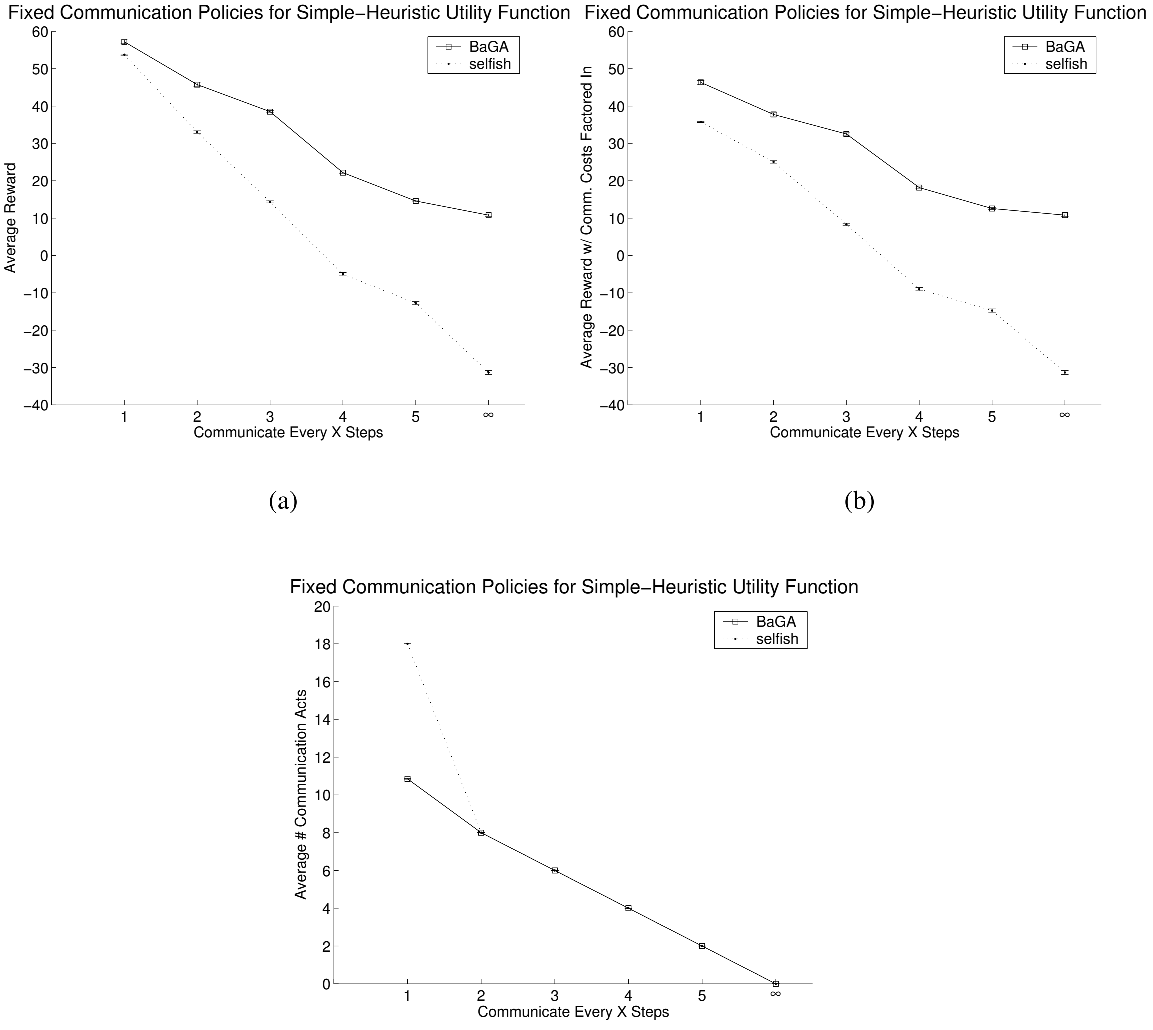
A fixed communication policy in which agents communicate their cluster number every x steps was first implemented. Communication is assigned a cost of 1.0 (i.e., the same as a *listen* action) and agents do not communicate on the first timestep of the problem (because they have no observations), nor if they only have one cluster. An x of ∞ is used to represent the case of no communication. Performance was compared to a *selfish* implementation, also using the fixed communication policy, for both the *simple-heuristic* and *coop-POMDP* utility functions. Results can be seen in Figures 5.1 and 5.2 respectively. In both cases, BaGA-Comm was implemented using minimum-distance clustering with a maximum-allowable expected-loss threshold of 0.01. A communication act in the *selfish* version consists of transmitting all observations and actions since the last communication act; however, it is assumed to have the same cost as transmitting a cluster number.

With this type of fixed communication policy, both BaGA-Comm and the *selfish* implementation perform best with full communication and worst with no communication, even when the costs of communication are factored into reward. With BaGA-Comm, however, the agents are still able to properly reason about the other agent's actions and observations in between communication acts and therefore performance does not drop off as quickly as x increases in value, as it does with the *selfish* version. With the *coop-POMDP* heuristic, average reward does not decrease monotonically as x is increased. For the *selfish* version, performance is definitely better with certain values of x, while BaGA-Comm does better with $x = 5$ than it does with $x = 4$.

This result makes sense because, in this domain, communicating at certain times can be better than others. For example, if an agent is about to open a door, then sharing observations with its teammates will either cause both agents to open the same door or neither to open a door. Overall, this behaviour will increase average reward. If the value of x is such that agents do not happen to communicate on these kinds of timesteps, then lower performance occurs. As expected, this effect on performance is most apparent with the *selfish* application of the *coop-POMDP* heuristic because, for this heuristic, *selfish* agents open doors the most frequently. Therefore, they will be impacted the most by the correlation between different values of x and on which timesteps they open doors.

If it is the case that when and not just how often an agent communicates affects performance, then run-time communication decisions become very important. Table 5.1 shows the performance of the EVD and PD communication decisions for BaGA-Comm. BaGA-Comm was implemented with minimum-distance clustering with a maximum-allowable expected-loss threshold of 0.01. EVD and PD result in similar performance and number of communication acts to each other. For the *Lady and the Tiger* domain, the policy $\sigma^{i,t}$ tends to satisfy both the EVD and PD conditions at the same time. The result of such a communication act is to either coordinate the agents on opening the same door or to inform them that they have conflicting observations and that no door should be opened.

If one were to look at the policies generated by *simple-heuristic* or *coop-POMDP* in the absence of communication, they are actually quite similar. Each agent needs to receive a majority of observations of one kind before it will commit to opening a door, and each agent will *listen* for many more timesteps in a row than it does in the communicating policies. Because each agent cannot directly pool its observations with its teammate, it can only infer a probability distribution over the set of all possible joint-observation histories and will therefore be more conservative and require higher quality evidence of its own before committing to opening a door. It is therefore interesting that, when communication is added, *coop-POMDP* performs better than *simple-heuristic*. This is, however, a result of how the two utility functions estimate the future expected value of actions. While *coop-POMDP* is very optimistic, *simple-heuristic* does not assume that in the future agents will be able to share their observations. As a result, *simple-heuristics* leads to policies in which agents are more conservative about when they would open doors because it does not overestimate the number of times that they will be able to open doors in the remaining timesteps. If commu-

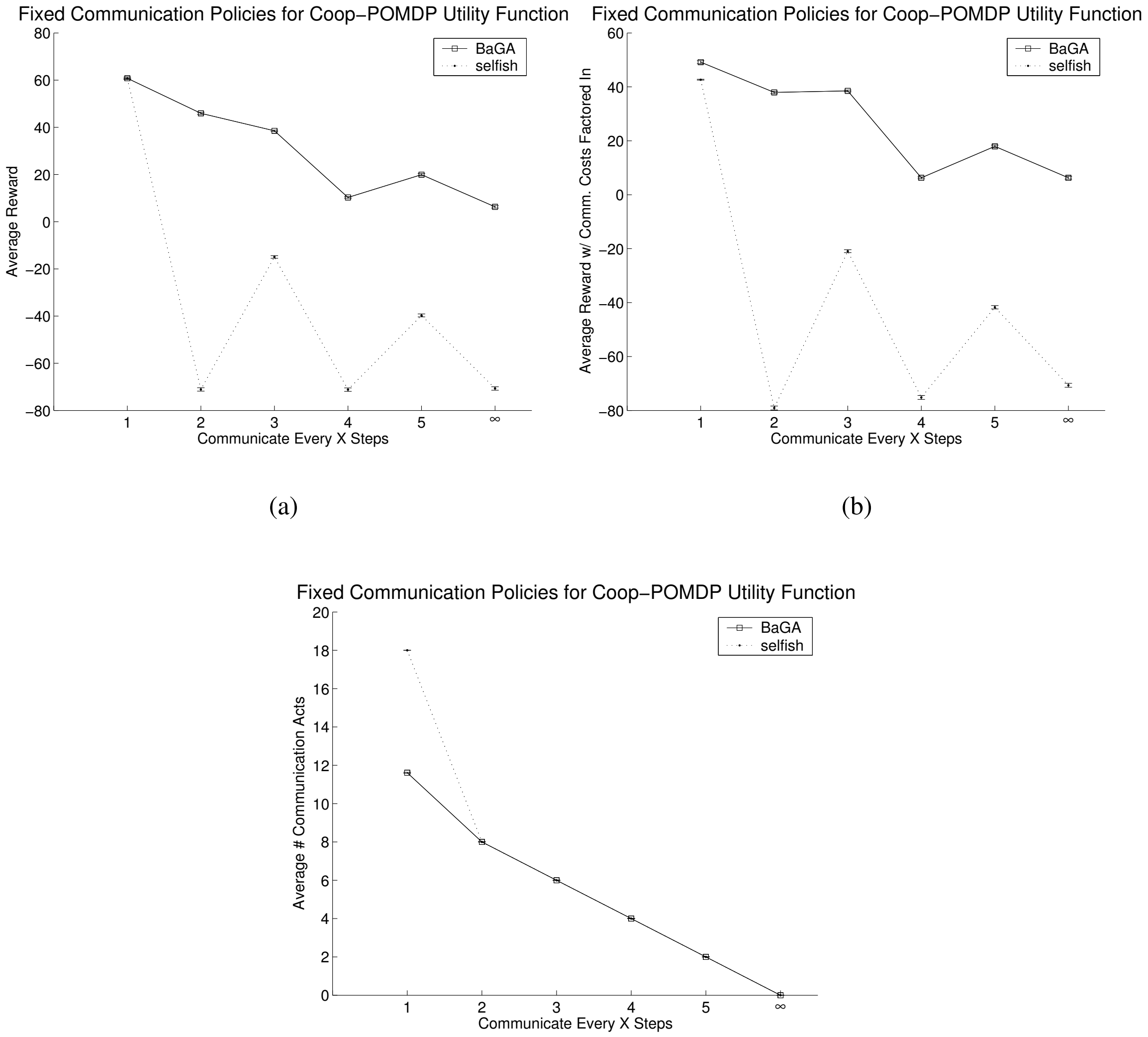


(a)

(b)

(c)

Figure 5.1: Fixed communication policy results for a 10-step version of the *Lady and The Tiger.* The BaGA-Comm and *selfish* algorithms used the *simple-heuristic* for estimating the future value of actions. Each fixed communication policy communicates every x steps, with $x = \infty$ meaning never communicate. BaGA-Comm was implemented with minimum-distance clustering with a threshold of 0.01 and used 200 random restarts for alternating-maximization calls. The average cumulative reward to the team without communication costs factored in is shown in (a), while the average reward with communication costs included in shown in (b). (c) shows the average number of communication acts for the team over the 10 timesteps. The difference in the number of communication acts for the $x = 1$ case is caused by a BaGA-Comm controlled agent not needing to communicate on those timesteps in which it has only one type. All results are averaged over 10000 trials, and 95% confidence intervals are shown as error bars.



(a)

(b)

(c)

Figure 5.2: Fixed communication policy results for a 10-step version of the *Lady and The Tiger.* The BaGA-Comm and *selfish* algorithms used *coop-POMDP* for estimating the future value of actions. Each fixed communication policy communicates every *x* steps, with $x = \infty$ meaning never communicate. BaGA-Comm was implemented with minimum-distance clustering with a threshold of 0.01 and 200 random restarts used for alternating-maximization calls. The average cumulative reward to the team without communication costs factored in is shown in (a), while the average reward with communication costs included in shown in (b). (c) shows the average number of communication acts for the team over the 10 timesteps. The difference in the number of communication acts for the $x = 1$ case is caused by a BaGA-Comm controlled agent not needing to communicate on those timesteps in which it has only one type. All results are averaged over 10000 trials, and 95% confidence intervals are shown as error bars.

Table 5.1: Run-time communication policy results for a 10-step version of the *Lady and the Tiger*. BaGA-Comm was implemented with minimum-distance clustering with a threshold of 0.01 and 200 random restarts for each instance of alternating maximization. All results are averaged over 10000 trials with 95% confidence intervals shown. Figures 5.3 and 5.4 compare the performance of BaGA-Comm with EVD to various fixed communication policies.

Algorithm	# Comm. Acts	Reward	Reward+Comm.Costs	Average # Clusters
simple-heuristic				
EVD	5.16 ± 0.02	38.97 ± 0.30	33.81 ± 0.30	20.59 ± 0.05
PD	5.13 ± 0.02	39.05 ± 0.31	33.92 ± 0.31	20.62 ± 0.05
selfish, just about to open door	4.94 ± 0.01	39.74 ± 0.10	34.80 ± 0.10	n.a.
coop-POMDP				
EVD	9.79 ± 0.01	57.53 ± 0.54	47.74 ± 0.54	16.92 ± 0.03
PD	9.79 ± 0.01	57.79 ± 0.53	48.00 ± 0.53	16.93 ± 0.03
selfish, just about to open door	9.71 ± 0.01	58.45 ± 0.17	48.74 ± 0.17	n.a.

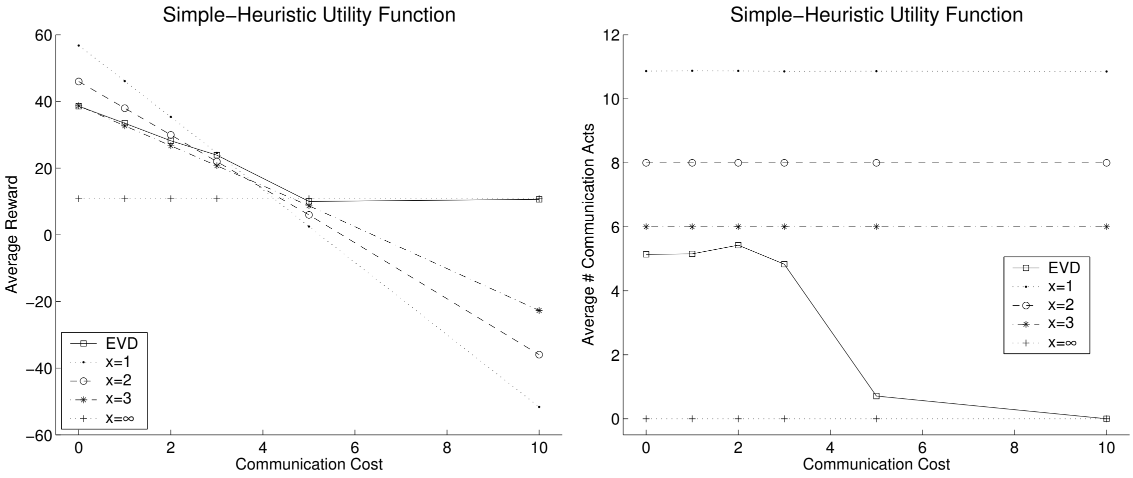
nication is allowed, however, the assumptions governing the *coop-POMDP* utility function become more realistic as agents will be able to share future observations. *coop-POMDP*'s run-time communication policies are therefore more aggressive and involve more communication.

For example, with *coop-POMDP*, the agents communicate their cluster number after receiving a single observation. If their observations match then they both open the appropriate door. At best, the agents will be able to open five doors over the 10 steps of the problem. Even though opening doors based on the evidence of just two observations is riskier than gathering more evidence, the average performance of such a policy is quite high. With *simple-heuristic*, the agents wait until they have received two observations before communicating. If three out of these four total observations are the same, then the agents will open the appropriate door. With this policy, the agents will only have three chances to open the door and so do not have as many opportunities to achieve high reward. They do, however, act upon higher quality information (as it incorporates more evidence) and so the loss in average reward compared to *coop-POMDP* is less than that which would be gained by opening one more door.

BaGA-Comm with either *simple-heuristic* or *coop-POMDP* is very aggressive in the number of communication acts it makes with both EVD and PD. As a result, the type space of each agent stays very small and it was not necessary to implement any communication decisions based on the size of this space. With *simple-heuristic,* because each agent communicates after two *listen* actions, the size of the type space of each agent does not grow above three for any single timestep, while for *coop-POMDP* the type space has a size limited to two.

Regardless of the utility function chosen, each policy takes the form of an agent gathering some amount of information (that amount being dependent on the utility function chosen) and then sharing that information with its teammate to decide whether or not to open a door. The agents, as a result, always select the same action: they either both listen or both open the same door. These joint actions correspond to the joint actions that are non-dominated in a POMDP version of the problem in which all observations are shared between the team (by non-dominated, it is meant that, in such a POMDP, only these joint actions will ever be executed). The same is not true of BaGA-Cluster when the agents could not communicate. However, intuitively, as more communication is added into the POSG model, the agents should start to act more and more as they would in the POMDP version of the problem with full communication. As expected, the BaGA-Comm policies satisfy this intuition.

The policies generated by BaGA-Comm inspired a communication heuristic for *selfish* agents in which an agent will communicate its observation history to its teammate if it has received enough information to be willing to open a door. That way, so long as it does not have conflicting information, the teammate will also open that door. If the teammate does have conflicting observations (for example, it has heard the tiger behind the left door twice while the other agent heard it behind the right door twice), then it generally has also decided to open a door and so will communicate that information to the original agent as well. In this case, the two agents would then both pick the *listen* action in order to gather more evidence. In other words, agents will always perform the same action. This communication heuristic, when combined with the *selfish* implementation of either utility function, performs extremely well. The average reward achieved by the team, once communication costs have been factored in, is the same or similar than how it did with full communication ($x = 1$) but with less than half the number of communication acts needed. Even without communication costs factored in, this communication heuristic outperformed the fixed communication policy with any value of x greater than 1. This heuristic is also able to achieve performance equal or better than that of BaGA-Comm using either the EVD or PD communication decision at a fraction of the computational cost. Of course, this heuristic does not attempt to make any tradeoffs between communication costs and performance; however, it does show that run-time communication policies generated through BaGA-Comm can be used to inspire communication heuristics that work very well.



(a) (b)

Figure 5.3: Effects of varying the communication costs for a 10-step version of the *Lady and The Tiger*. The performance of both fixed communication policies and EVD are compared. All policies use *simple-heuristic* for estimating the future value of actions. Each fixed communication policy is to communicate every *x* steps, with $x = \infty$ meaning never communicate. BaGA was implemented with minimum-distance clustering with a threshold of 0.01 and 200 random restarts used for alternating maximization. The average cumulative reward to the team with communication costs factored in is shown in (a). (b) shows the average number of communication acts for the team over the 10 timesteps. All results are averaged over 10000 trials but the error bars representing 95% confidence intervals have been omitted in order to improve clarity.

5.3.1.1 Varying Communication Cost

Up to this point, communication costs have been held fixed at 1.0. Figures 5.3 and 5.4 graph the effects of varying communication costs. The performance of both fixed communication policies and BaGA-Comm with the EVD condition are compared for a 10-step version of the *Lady and the Tiger*. As the cost of communication increases, BaGA-Comm is able to reduce the number of communication actions it takes until it starts to perform more similarly to BaGA without communication (indicated by the $x = \infty$ case). The fixed communication policies, however, are not able to take the cost of communication into account and so their performance drops off drastically.

Figure 5.3 (a) shows the average reward, including communication costs, generated by these policies using the *simple-heuristic* utility. The average number of communication

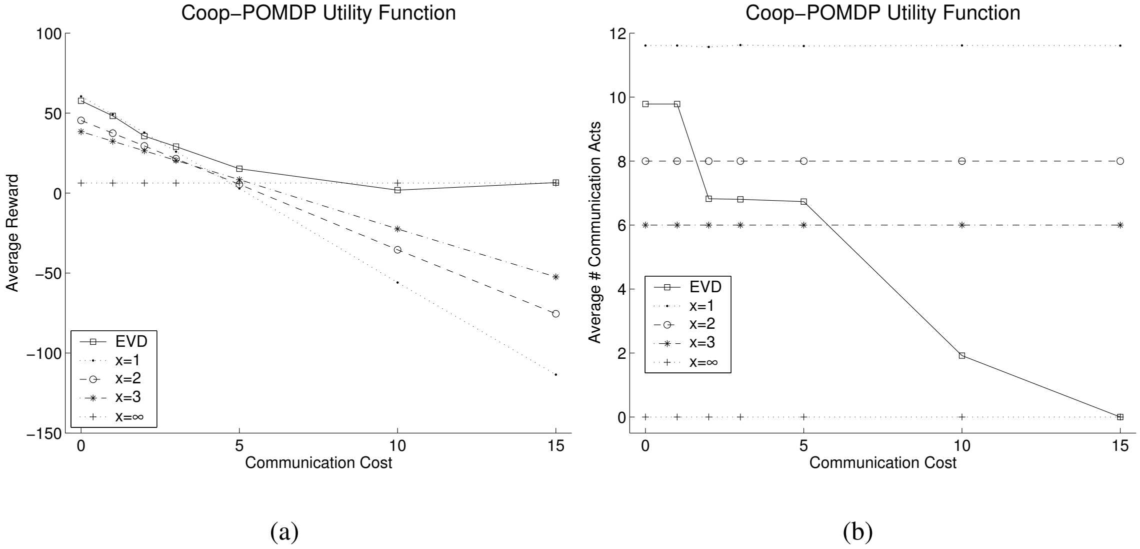


Figure 5.4: Effects of varying the communication costs for a 10-step version of the *Lady and The Tiger.* The performance of both fixed communication policies and EVD are compared. All policies use *coop-POMDP* for estimating the future value of actions. Each fixed communication policy is to communicate every *x* steps, with $x = \infty$ meaning never communicate. BaGA was implemented with minimum-distance clustering with a threshold of 0.01 and 200 random restarts used for alternating maximization. The average cumulative reward to the team with communication costs factored in is shown in (a). (b) shows the average number of communication acts for the team over the 10 timesteps. All results are averaged over 10000 trials but the error bars representing 95% confidence intervals have been omitted in order to improve clarity.

acts is shown in Figure 5.3 (b). The fixed communication policy of communicate every step but the first, i.e., $x = 1$, averages 10.87 communication acts per run (rather than the expected 18.00), because agents do not need to communicate if they have only one possible type.

The *simple-heuristic* utility was designed for the case of no communication and as a result, does not accurately model future expected reward when the agents are able to communicate frequently. Therefore, the fixed communication policies of $x = 1$ and $x = 2$ do better than EVD when communication costs zero or 1.0 per act. EVD, however, performs similarly to the $x = 3$ case for these communication costs, even though it performs one fewer communication act per run. As the cost of communication increases to 2.0, EVD starts to perform as well as the $x = 2$ case, and by the time communication costs have reached 3.0, it is performing similarly to the $x = 1$ case. It is postulated that the slight increase in communication acts when communication costs 3.0 is caused by EVD communicating at different timesteps for certain sets of observations (e.g., if EVD pushes off communication at an earlier timestep, this will effect what actions are taken later and, for certain histories, actually end up triggering the EVD condition more often).

By the time communication costs are increased to 5.0, EVD is barely communicating and it stops entirely when the costs are raised to 10.0. At this point its performance is equivalent to BaGA-Cluster without any communication. BaGA-Cluster without communication ($x = \infty$) does not entirely provide a lower bound on the performance of EVD (as seen when communications costs 5.0), because of EVD's myopic nature and the use of a heuristic for the utility function. EVD's success is dependent on being able to accurately compare an expected increase in performance with the cost of communication. At some point, however, unless the heuristic used to calculate this distance is exact, EVD will err and result in either more or fewer communication acts than is 'optimal'. For *simple-heuristic,* this point seems to occur when communication costs 5.0. Once the cost is raised to 10.0, these inaccuracies have less effect.

A comparison of the performance of these policies was also made using the *coop-POMDP* heuristic function, and can be seen in Figure 5.4. As *coop-POMDP* is a much better estimate of future reward when agents can communicate, EVD performs similarly to the $x = 1$ case for communication costs of 1.0, 2.0 and 3.0. As communication costs increase above 3.0, EVD outperforms all of the fixed communication policies. By this point, EVD has reduced its average number of communication acts by about three. As with *simple*-heuristic*,* the performance of BaGA-Cluster without communication does not entirely provide a lower bound to the performance of EVD. When communication costs 10.0, EVD still communicates 1.92 times on average. However, by the time communication has increased in cost to 15.0, EVD stops communicating and behaves like the $x = \infty$ case.

5.3.2 Robotic Tag

Communication policies were also implemented for the *unknown-teammate-position* version of *Robotic Tag*. This variant was selected because BaGA requires a large number of types in order to find good solutions without communication, and so it is a good candidate for communication decisions designed to reduce computational costs. BaGA-Comm was used with a combination of low-probability pruning with a cutoff of 0.00025 and minimum-distance clustering with a threshold of zero, except for the case of no communication in which these parameters were 0.0025 and 0.0001 respectively (in this latter case, the lack of communication required that the pruning/clustering parameters be raised in order to keep the joint-type space manageable).

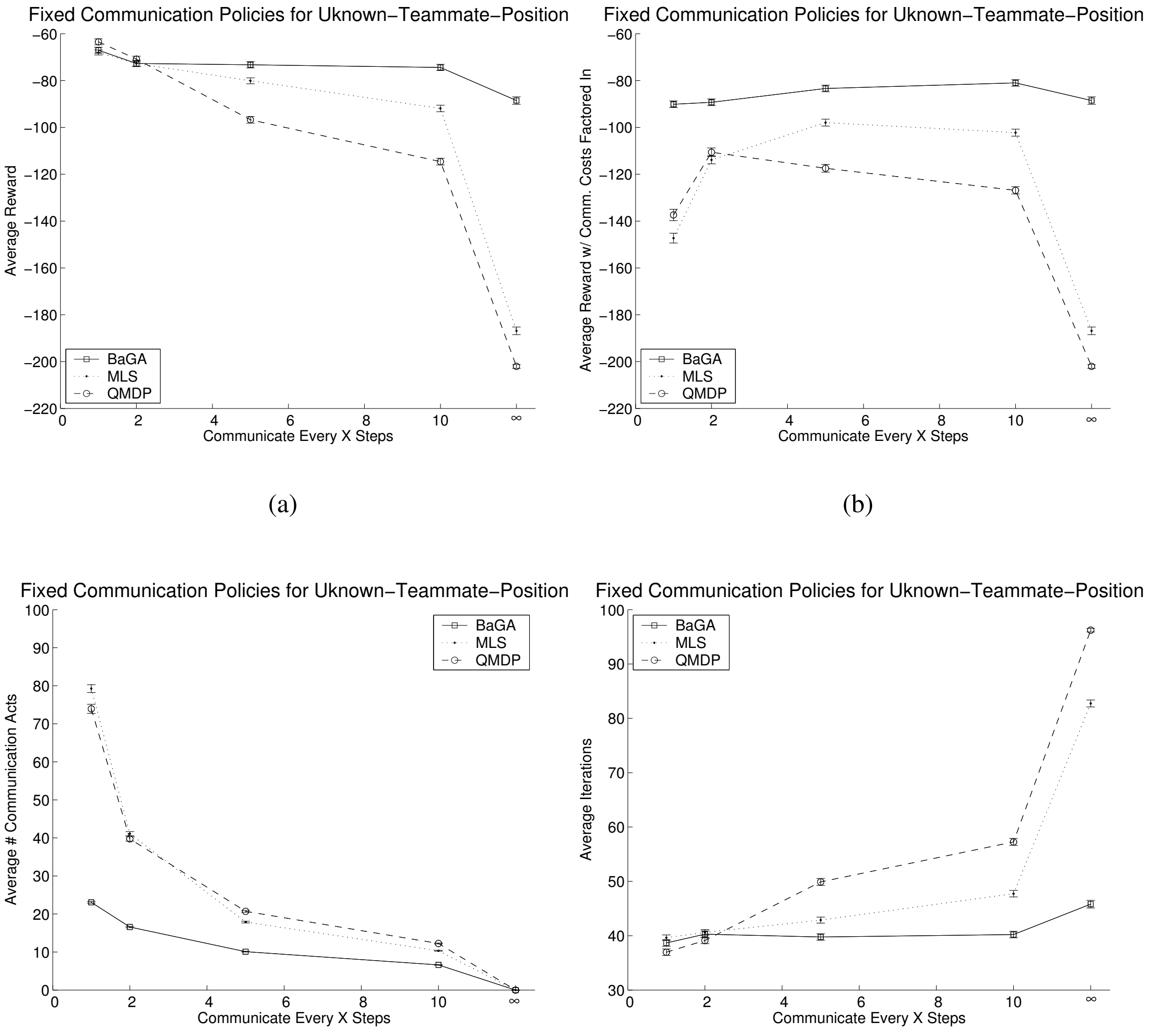
First, the fixed communication policy of broadcasting information every x timesteps was implemented with results shown in Figure 5.5 for both BaGA-Comm and the MLS and Q_{MDP} heuristics. As with the *Lady and the Tiger* domain, communication is assumed to have a cost of 1.0 regardless if it is the broadcast of a cluster number or, as in the case of MLS and Q_{MDP}, the history of observations and actions made since the last communication act. In BaGA-Comm, even for $x = 1$, a robot does not communicate every timestep because there is no need to do so if it only has one cluster. For this reason, the number of communication acts performed by BaGA is always less than twice the number of iterations taken to capture the opponent. MLS and Q_{MDP}, however, do not have anything similar of which to take advantage and thereby reduce overall communication.

MLS and Q_{MDP}, after an initial increase in average reward (with communication costs included), experience a rapid drop-off in performance as the number of communication acts decreases. As discussed in Section 3.4.3, the coordination of the team becomes compromised if the robots do not have an accurate idea of where their teammate is located. While communication is able to re-sync the believed position of the teammate with its true position, in between communication acts these positions will start to drift apart again. As x increases, the robots spend more of their time uncoordinated and therefore performance suffers.

For values of x greater than 1, BaGA-Comm is able to reason about histories in between communication acts. If one looks at the average reward with communication costs factored in, BaGA-Comm (and, to a limited extent, the other two heuristics) performs better with fewer communication acts. In fact, BaGA-Comm is fairly stable with respect to average reward and number of iterations taken to capture the opponent, even as the number of communication acts decreases. It is, however, a result of using a fixed communication policy, because the robots do not necessarily communicate at key points in the problem. For this domain, a critical decision-making point is when the two robots are positioned to tag the opponent in the goal cell but the opponent is not actually there. One of the robots will be aware of this fact and the communication of this piece of information makes a difference to the overall performance of the game. While a fixed communication policy cannot make the decision to communicate at this critical time, a run-time communication policy can.

The first type of run-time communication examined is that of EVD. The EVD communication policy takes about three timesteps more to capture the opponent than a fixed communication policy with $x = 1$, but it leads to a higher average reward once the cost of communication is factored in. With EVD, the average number of communication acts per trial is less than one (compared to about 23 for the $x = 1$ policy). If one looks at when the robots do communicate using EVD, the primary time they do so is when the robots are set up to tag the opponent and one robot is in the goal cell. If the opponent is there, it is not necessary for the robot in that cell to transmit that information to its teammate so long as the policy for the teammate, π^t_{-i}, has it also performing a *tag* action with high probability. If the *tag* action will not be performed with high enough probability, then the robot in the goal cell does transmit the relevant information (i.e., its cluster number that encapsulates that information) so that they will both do *tag*. If the opponent is not there, but the teammate will be performing a *tag* action with high probability according to π^t_{-i}, then the robot tells its teammate its cluster number and the miscoordination of the *tag* action is avoided. Because miscoordinating on the *tag* action has a penalty of 10 while the cost of communication is 1.0, these types of scenarios trigger the EVD condition. EVD will also be triggered if a robot is in the same cell as the opponent during an early timestep of the problem.

EVD was also combined with the computational-tradeoff decision in which robots communicate if their current number of clusters exceeds a threshold of $k = 20$. Adding the additional cluster-threshold condition to EVD does not result in the team catching the opponent faster, but it does greatly reduce the total number of clusters maintained by each robot, which translates to faster planning. On its own, the cluster-threshold condition does perform better than no communication at all, but not nearly as well as EVD or PD, which



(a)

(b)

(c)

(d)

Figure 5.5: Fixed communication policy results for the *unknown-teammate-position* variation of *Robotic Tag.* Each fixed communication policy communicates every x steps, with $x = \infty$ meaning never communicate. BaGA-Comm was implemented with a combination of low-probability pruning with a cutoff of 0.00025 and minimum-distance clustering with a threshold of zero (i.e., two clusters are only merged if they have identical reward profiles), except for the case of $x = \infty$ in which those parameters were 0.0025 and 0.0001, respectively (the parameters were raised in this latter case to help keep the joint-type space manageable). The average cumulative reward to the team without communication costs factored in is shown in (a), while the average reward with communication costs included is shown in (b). (c) shows the average number of communication acts for the team over the length of the task, while (d) shows the average number of timesteps taken to capture the opponent. All results are averaged over 10000 trials with 95% confidence intervals shown.

Table 5.2: Run-time communication policy results for the *unknown-teammate-position* variation of *Robotic Tag*. All results are averaged over 10000 trials with 95% confidence intervals shown. The average number of clusters is the average of the summed number of clusters maintained by a robot at each timestep, i.e., $\sum_t |C_i^t|$. BaGA-Comm was implemented with a combination of low-probability pruning with a cutoff of 0.00025 and minimum-distance clustering with a threshold of 0.00 except for the case of no communication, in which case those parameters were 0.0025 and 0.0001, respectively. Results for fixed communication policies are included for comparison purposes.

Policy	**Capture Time**	**# Comm. Acts**	**Reward**	**Reward+Comm. Costs**	**Average # Clusters**	**% Success**
BaGA-Comm						
no comm.	45.69 ± 0.67	n.a.	-88.15 ± 1.52	n.a.	1032.81 ± 27.86	77.5%
fixed, $x = 1$	38.67 ± 0.61	23.06 ± 0.21	-67.06 ± 1.29	-90.11 ± 1.40	50.20 ± 0.67	85.6%
fixed, $x = 2$	40.23 ± 0.24	16.57 ± 0.17	-72.71 ± 1.34	-89.28 ± 1.42	62.14 ± 0.74	84.7%
EVD	41.73 ± 0.64	0.89 ± 0.02	-73.65 ± 1.36	-74.53 ± 1.37	742.46 ± 26.91	82.1%
EVD + $\|C_i\| > 20$	41.41 ± 0.64	2.43 ± 0.04	-72.94 ± 1.36	-75.37 ± 1.37	248.02 ± 3.45	82.5%
PD	38.91 ± 0.61	6.45 ± 0.08	-67.51 ± 1.28	-73.96 ± 1.31	171.27 ± 3.42	85.7%
PD + $\|C_i\| > 20$	38.86 ± 0.61	6.57 ± 0.08	-67.41 ± 1.28	-73.98 ± 1.32	152.19 ± 2.49	86.0%
$\|C_i\| > 20$	42.84 ± 0.63	2.38 ± 0.04	-80.32 ± 1.39	-82.70 ± 1.41	287.98 ± 3.66	82.5%
Selfish Heuristic						
MLS, no comm.	82.74 ± 0.64	n.a.	-186.85 ± 1.64	n.a.	n.a.	23.4%
MLS, fixed $x = 1$	39.63 ± 0.52	79.26 ± 1.03	-68.02 ± 1.07	-147.28 ± 2.10	n.a.	93.5%
MLS, fixed $x = 2$	40.61 ± 0.53	41.12 ± 0.53	-72.74 ± 1.17	-113.61 ± 1.70	n.a.	92.4%
MLS, unexpected - no sync	64.16 ± 0.66	7.21 ± 0.12	-131.98 ± 1.66	-139.19 ± 0.12	n.a.	60.1%
MLS, unexpected - forced sync	48.84 ± 0.60	4.40 ± 0.07	-89.81 ± 1.29	-94.21 ± 1.34	n.a.	85.5%
Q_{MDP}, no comm.	96.23 ± 0.34	n.a.	-202.05 ± 0.81	n.a.	n.a.	4.9%
Q_{MDP}, fixed $x = 1$	36.93 ± 0.60	73.93 ± 1.19	-63.44 ± 1.25	-137.37 ± 2.44	n.a.	87.3%
Q_{MDP}, fixed $x = 2$	39.12 ± 0.58	39.73 ± 0.58	-70.85 ± 1.24	-110.59 ± 1.81	n.a.	86.6%
Q_{MDP}, unexpected - no sync	76.35 ± 0.61	14.78 ± 0.18	-159.78 ± 1.41	-174.57 ± 1.54	n.a.	46.3%
Q_{MDP}, unexpected - forced sync	56.28 ± 0.65	8.10 ± 0.14	-110.14 ± 1.41	-118.23 ± 1.46	n.a.	70.6%

was the second type of run-time communication policy tested.

PD results in similar performance to communicating every timestep but at a quarter of the number of communication acts. With communication costs factored in, its average cumulative reward is similar to that of the EVD communication policy. This result is a reflection of how the EVD policy actually trades off communication costs and expected increase in reward whereas the PD policy does not. Again, adding the additional cluster-threshold condition to PD did not greatly affect performance, but it reduced the average number of total clusters maintained by each robot, thereby improving computation time.

While PD is triggered by every situation in which EVD would be triggered (as a policy difference is what leads to an expected difference in performance), it will also be triggered by cases in which a policy difference would occur as the result of communication but with no restrictions on the resulting difference in expected reward. These additional cases are why PD results in about six times the number of communication acts. For example, PD will be triggered if a robot is in the goal cell, regardless of whether the opponent is there or not, or it will be triggered if both robots appear in Cell 24 at the same time. It is postulated that this last event results in a policy change because it is one of the few positions in the grid in which robots sometimes appear simultaneously. While each robot will see its teammate, it is not assumed to be common knowledge (i.e., that robot i knows that its teammate sees it and that its teammate knows that i knows that its teammate sees it and so on), therefore by establishing this information as common knowledge (achieved by sharing cluster numbers), the probability distribution over the set of joint types is changed so much that policy changes would also occur.

The conditions under which EVD and PD lead to communication acts were used to create a run-time communication heuristic for MLS and Q_{MDP}. Essentially, this heuristic captures the idea that in BaGA-Comm robots communicate when something unexpected happens. For example, if the opponent is not in the goal cell but the joint-type space of the team is such that a miscoordination on the *tag* action is likely to occur, then both EVD and PD will communicate. Rather than just have the communication heuristic be specific to unexpected events concerning the goal cell, it was expanded. For example, if a robot following the MLS heuristic or Q_{MDP} heuristic determines that the most likely position for the opponent on the next timestep is cell y but, upon moving into that cell itself, does not detect the opponent, it will communicate. Similarly, a robot moving into the goal cell that does not detect the opponent would broadcast its observation and action history to its teammate (it would not have moved into that cell unless it believed with high probability that the opponent was there). Other unexpected events are when a robot does not detect its teammate in the cell in which it believes its teammate to be in.

This *unexpected* communication heuristic was implemented both with and without a forced synchronization between the robots. If a robot decides to communicate, it tells its teammate its observation and action history. In the forced-sync case, its teammate must then tell its own history back. Both approaches resulted in better performance than if there is no communication available to the team. As anticipated, the forced-sync version of the heuristic worked better for both MLS and Q_{MDP} because many of the unexpected events that trigger a communication decision actually involve a robot needing information (such as its teammate's true position given that the robot's believed position of it must be wrong). For both MLS and Q_{MDP}, the forced-sync communication heuristic resulted in performance similar or better than any of the fixed communication policies once the cost of communication is factored in. Without communication costs factored in, they both performed similarly to the $x = 5$ case, but with fewer than half the number of communication acts. If synchronization is not forced, the communication heuristic does not lead to such good performance. Only the case of $x = 1$ is now outperformed with respect to average reward including communication costs.

5.4 Summary

In this chapter, BaGA is extended to handle robot teams with communication. In order to find optimal run-time communication policies it is necessary to include communication acts in the action set. Unfortunately, doing so increases the number of histories that must be tracked by the BaGA algorithm by a factor exponential in the number of robots. Rather than find such an optimal universal plan, BaGA-Comm generates run-time communication decisions myopically: each robot computes domain policies both with and without communication and, based on their differences, communicates.

Various criteria can be used to determine when to communicate, BaGA-Comm includes decisions based on expected improvements in performance as well as decisions based on computational requirements. These run-time communication policies not only improve the performance of the robot team in domains like *Lady and the Tiger* and *Robotic Tag*, but also reduce computational costs. By using run-time communication decisions rather than fixed communication policies, robots can adapt their behaviour to accommodate different communication costs.

Chapter 6

Real-Time Robot Controllers

This chapter discusses how the BaGA algorithms can be used to implement real-time robot controllers. Realistic robot problems generally require more sophisticated state transition and observation models in order to take into account noisy sensors and actuators. As a result, a large number of observation and action histories must be tracked by the BaGA algorithm. In order to examine the effects of more complex models, a game of robotic tag in the Gates Building at Stanford University is presented. Several versions of the game are considered, which gradually increase in realism.

In these games, a robot team is trying to coordinate and tag an opponent that is moving around an office building. Even if only one robot is required to tag the opponent, the team is penalized for movement, and so robots must still coordinate their searches. The robots only have 100 timesteps in which to tag the opponent in any location in the environment.

6.1 Simple Gates Tag

In the most abstract version of this problem, the A Wing of the Gates Building is converted into a grid world by discretizing the free space into a series of cells that are roughly $1.0m \times 3.5m$. Two robots are hunting for an opponent within this grid world as shown in Figure 6.1(a). The state space for this problem is the cross-product of two robot positions $r_1 = r_2 = \{s_0, \ldots, s_{25}\}$ and the opponent state $op = \{s_0, \ldots, s_{25}, s_{tagged}\}$ and has a total of 18252 states. All three robots start in randomly selected cells and the game is over when $op = s_{tagged}$. Each robot has five actions $A_i = \{$*north, south, east, west, tag*$\}$ and all actions have deterministic outcomes. A shared penalty of -1 is incurred for each motion action. The *tag* action results in a reward of +10 to the team if the robot performing the *tag* action is in the same cell as the opponent ($r_i = op$ and $a_i =$*tag*), otherwise it results in a penalty of -10. The opponent moves with Brownian motion and at each timestep either does not move or moves into one of its adjacent cells with known probability.

Each robot is able to observe the opponent only if they are in the same cell; otherwise it receives a null observation of the opponent leading to an observation set $Z_i = \{s_0, \ldots, s_{25}, \emptyset\}$.[1] For this variation of *Gates Tag*, the robots have no communication and so cannot share observations of the opponent, but they do know the exact position of themselves and their teammate.[2] As a result, a history θ_i for a robot includes only the observations it has made of the opponent. After constructing the policy π^t, any joint histories that have a robot implement an action from this policy that is not consistent with its current position can be removed from consideration at timestep $t + 1$.

As with the *Robotic Tag* problem of Chapter 3, the utility function for BaGA comes from the Q-values for the fully observable version of the problem, which were found using dynamic programming for an infinite-horizon version of the problem with a discount factor of 0.95. The value of taking a set of actions $\{a_1, a_2\}$, given the joint history θ and the resulting belief state b_θ it induces over the position of the opponent in the environment, is:

$$u(a_1, a_2, \theta) = \sum_{s \in s_{op}} b_\theta(s) Q(\{r_1, r_2, s\}, \{a_1, a_2\})$$

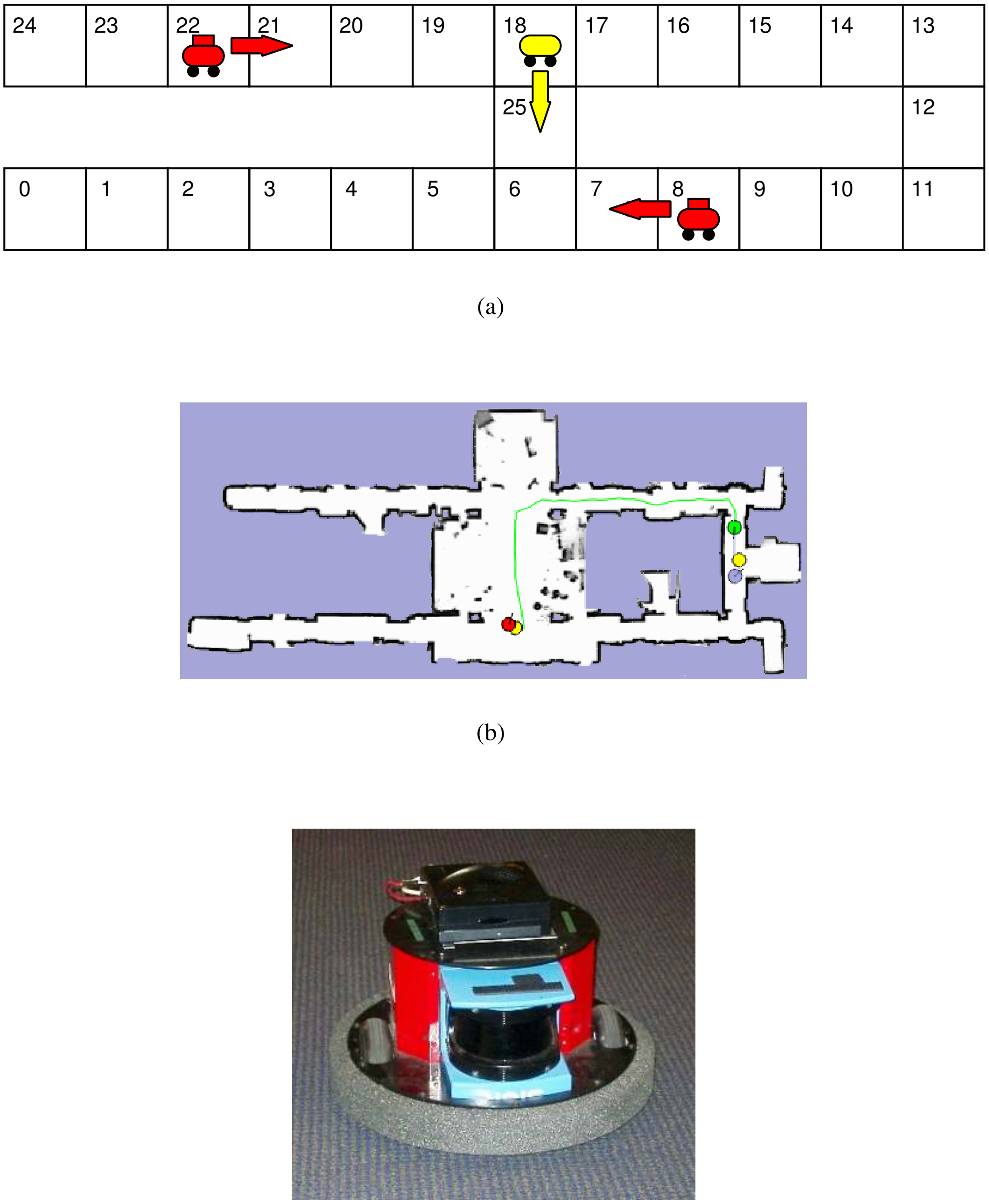
where $Q(\{r_1, r_2, s\}, \{a_1, a_2\})$ is the Q-value of taken the joint action $\{a_1, a_2\}$ in the state given by the positions r_1, r_2 and $op = s$.

With respect to the number of joint histories that must be tracked by the BaGA algorithm at each timestep, *Simple Gates Tag* has an upper bound on the branching factor of three.[3] That is, the number of possible joint histories at timestep $t + 1$ is at most three times that

[1] Technically, Z_i is actually the cross-product of the two robots' positions and the observation set $\{s_0, \ldots, s_{25}, \emptyset\}$ because each robot knows both its own position and that of its teammate: the notation has been abused for simplicity. However, it is important to remember that $|Z_i|$ is actually much greater than 27 when considering upper bounds on $|\Theta_i^t|$ such as those discussed in Sections 3.3.1 and 4.3.1.

[2] In all of the domains examined in this chapter, the assumption is made that robots can always identify their current cell with certainty. This assumption is valid for these real-world problems because the localization algorithm used in the high-fidelity simulation and on the real robots is very accurate (Montemerlo et al., 2002). In all experiments run for this chapter, the mapping between a robot's localized position in the continuous world and a discrete grid cell was always correct.

[3] Due to constraints on observations with respect to the positions of each robot (for example, if both robots are in the same cell, then they must have the same observation of the opponent or, if they are in different cells, they cannot both see the opponent), the branching factor is not four.



(a)

(b)

(c)

Figure 6.1: The Gates Building environment for robotic tag. The discrete environment used for obtaining statistical results is shown in (a). The opponent can be caught by either robot in any cell. A map of a portion of the physical Gates Building that was used for simulation and physical robot runs is shown in (b) and one of the Botrics Obot d100 robots used for the real-robot runs is shown in (c).

Table 6.1: Results for *Simple Gates Tag* in the grid world environment. BaGA was implemented with a low-probability pruning with a cutoff of 0.001. Average cumulative reward and the average number of iterations to capture the opponent are shown. % success is the percentage of trials in which the opponent was successfully tagged within 100 timestep. Results are also included for the performance of the fully observable policy under full observability. All results are averaged over 10000 trials with 95% confidence intervals shown.

Algorithm	**Average Reward**	**Average Iterations**	**% Success**
Fully Observable	1.18 ± 0.11	5.41 ± 0.06	100.00%
BaGA	-18.22 ± 0.55	15.10 ± 0.27	99.86%
MLS	-21.04 ± 0.56	16.52 ± 0.28	99.95%
Q_{MDP}	-38.56 ± 1.11	25.01 ± 0.54	95.40%

at timestep t. For this reason, the BaGA algorithm can be implemented with just low-probability pruning with a cutoff of 0.001. To farther cut down on the number of joint types tracked, any joint type that leads to a policy in which a robot implements the *tag* action while in the same cell as the opponent at timestep $t - 1$ was cut from any future possible types. Actions are deterministic and observations never noisy and so if a robot did execute the *tag* action in this situation, then the problem would have ended. As the problem has continued on to another timestep, this event could not have happened. Because it is mutual knowledge that the task is not complete, this common knowledge can be used to update the set of possible joint types.

In addition to the BaGA controller, the MLS and Q_{MDP} heuristics were also implemented for this environment in an identical fashion as how they were done for *Robotic Tag* in Section 3.4.3. Results for the three controllers can be found in Table 6.1, as can the results for a fully observable version of the problem using the policy given by taking the joint action with highest Q-value for each state. The random nature of the opponent means that all the controllers are almost always able to complete the task within 100 timesteps because, even if the robots were to never move, the opponent would eventually stumble into them. The performance differences, therefore, come from how the controllers make the robots explore the environment.

In BaGA, the robots split the environment between them, with robots exploring different ends of the hallway. Furthermore, while one robot moves around the loop in the right-hand side of the environment, the other robot will move back and forth in front of the intersection to that loop. If the opponent is in the loop, then either it will be discovered by the robot moving around the loop or by the robot ‘guarding’ the entrance to the loop. Because only one robot is necessary to tag the opponent, splitting up the environment in this fashion is efficient.

With the MLS heuristic, the robots have a tendency to follow each other around. Provided that the two robots generate a similar belief state, they will both select similar locations as the most likely position for the opponent. By then treating the problem as fully observable, it is logical for both robots to move towards the same location because the reward function is such that the team wants to minimize opponent capture time. However, as the problem is not really fully observable, this scheme is not an efficient use of resources leading to a slightly longer capture time than BaGA.

The Q_{MDP} heuristic takes longer on average to capture the opponent than other approaches because the robots have a tendency to get ‘trapped’ in cells near the middle intersection. During a trial, the robots will move towards this intersection and then stop moving because a multi-modal belief state leads no single action to have highest expected reward. A similar misbehaviour arose with the Q_{MDP} heuristic in the *Robotic Tag* problem of Section 3.4.3. The reason, however, that Q_{MDP} manages to successfully capture the opponent most of the time in this problem is because the opponent is not moving away from the team but rather moving randomly. In cases where the robots do not get trapped near the middle of the environment, Q_{MDP} does result in robots splitting the environment between them and adopting similar paths to those taken by BaGA. This behaviour is expected because BaGA uses Q_{MDP} as its utility function for actions. Unlike Q_{MDP}, however, BaGA rarely gets trapped because the one step of game-theoretic reasoning allows robots to avoid these situations.

6.1.1 High-Fidelity Simulation and Physical Robot Results

After testing the controller in the discretized version of the environment, policies were then executed in both a high-fidelity simulator and on physical robots. The mapping between the grid world and the real Gates Building and two Botrics Obot d100 robots was done using the Carmen software package for low-level control (Montemerlo et al., 2002). 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. Figure 6.1 (b) shows a screen shot from the simulator and Figure 6.1 (c) shows one of the robots used.

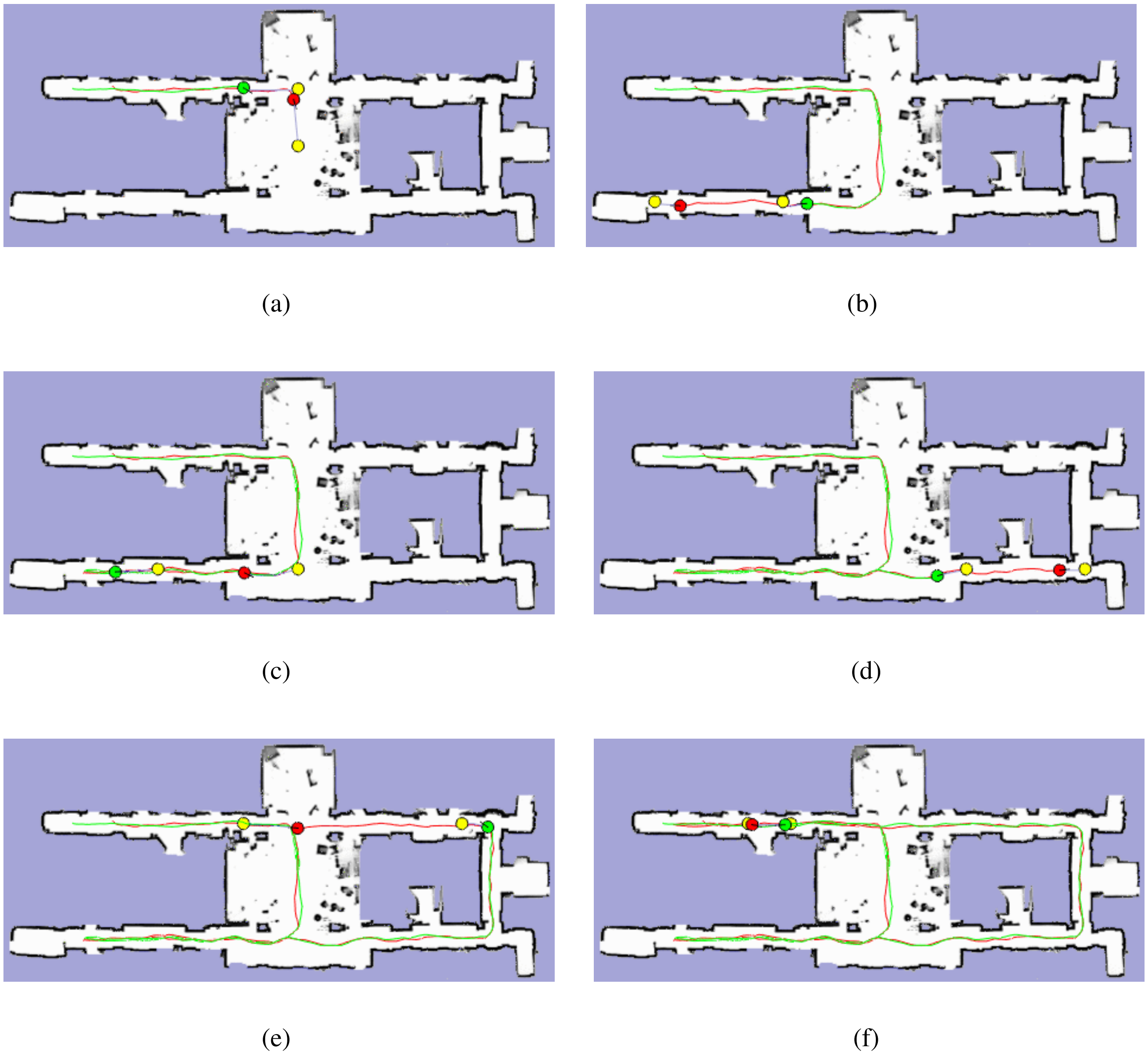
Localized positions of the robots within the map of the Gates Building are used to calculate their specific grid-cell positions. Robots navigate by converting the actions selected by BaGA into goal locations. In order to ensure robots always plan and execute actions in lockstep (because in the grid-world model robots move at the same time), robots will not execute their next actions until a message has been received by a central controller indicating that all robots had reached their last goal location. For runs in the real environment, no opponent was used in order to observe how the team coordinated to search the area. The robots, therefore, always received a no-opponent-present observation. As a result, the robots would continue to hunt for an opponent until stopped by a human operator.

Figures 6.2, 6.3 and 6.4 show the trajectories taken by robots in a simulation of this environment as they attempt to capture and tag a (non-existent) opponent while controlled by three different controllers. The reward function for this problem has not been designed to encourage the robots to efficiently cover the environment in the search for the opponent, but rather to minimize opponent capture time. However, when there is no opponent actually present in the environment, one can get a feel for how well the robot team actually searches for the opponent as opposed to taking advantage of the random nature of the opponent. It should be noted that, given the sensor limitations imposed on the robots for this problem and the structure of the environment, it is impossible for two robots to cover the environment in such a way as to guarantee capture of an opponent (specifically, moving into the middle corridor allows an opponent to 'sneak' by the team).

In the simulation runs, both robots started at the far right of the top corridor (Cells 22 and 23). With these starting conditions, the two robots will initially have very similar belief states about the opponent's position and this condition will persist so long as they remain physically close. With the MLS controller, shown in Figure 6.2, the two robots follow each other across the top corridor, down the middle corridor and into the bottom corridor. The robots almost go entirely into the far left side of the corridor before turning around to traverse the loop on the right-hand side of the environment. If they have still not found the opponent, they then move back into the upper corridor and then start to retrace their steps.

The robots do not go entirely into the ends of the corridor with MLS because generally, by the time they get into Cells 23 and 1, the probability that the opponent is in Cell 24 or 0 is low enough that some other cell is picked as the most likely position causing the robots to turn around. This type of behaviour is also exhibited by Q_{MDP} and BaGA; however, as time progresses and the opponent has still not been found, the controllers are more likely to move into the end cells.

The trajectory for Q_{MDP} under these starting conditions, shown in Figure 6.3, demonstrates how this controller causes the robots to stop moving. After following each other out of the

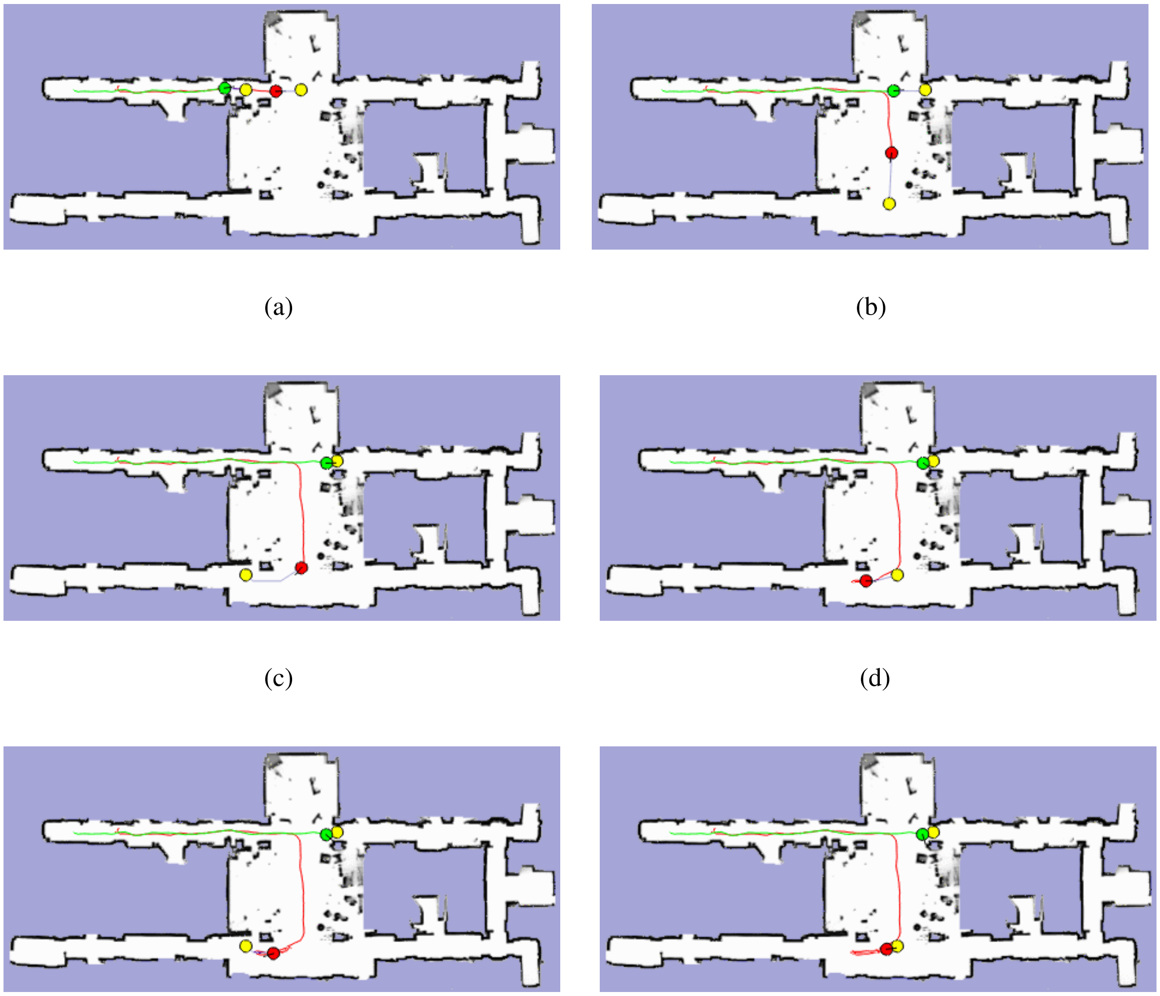


(a) (b)

(c) (d)

(e) (f)

Figure 6.2: Example of MLS-controlled robot trajectories for *Simple Gates Tag*. The MLS heuristic results in paths where the robots follow each other and so do not cover the space well. In this, and all following figures, the current goal location of each robot is shown by the pale circles. Robots themselves are represented by darker circles with a line indicating heading.



(a)

(b)

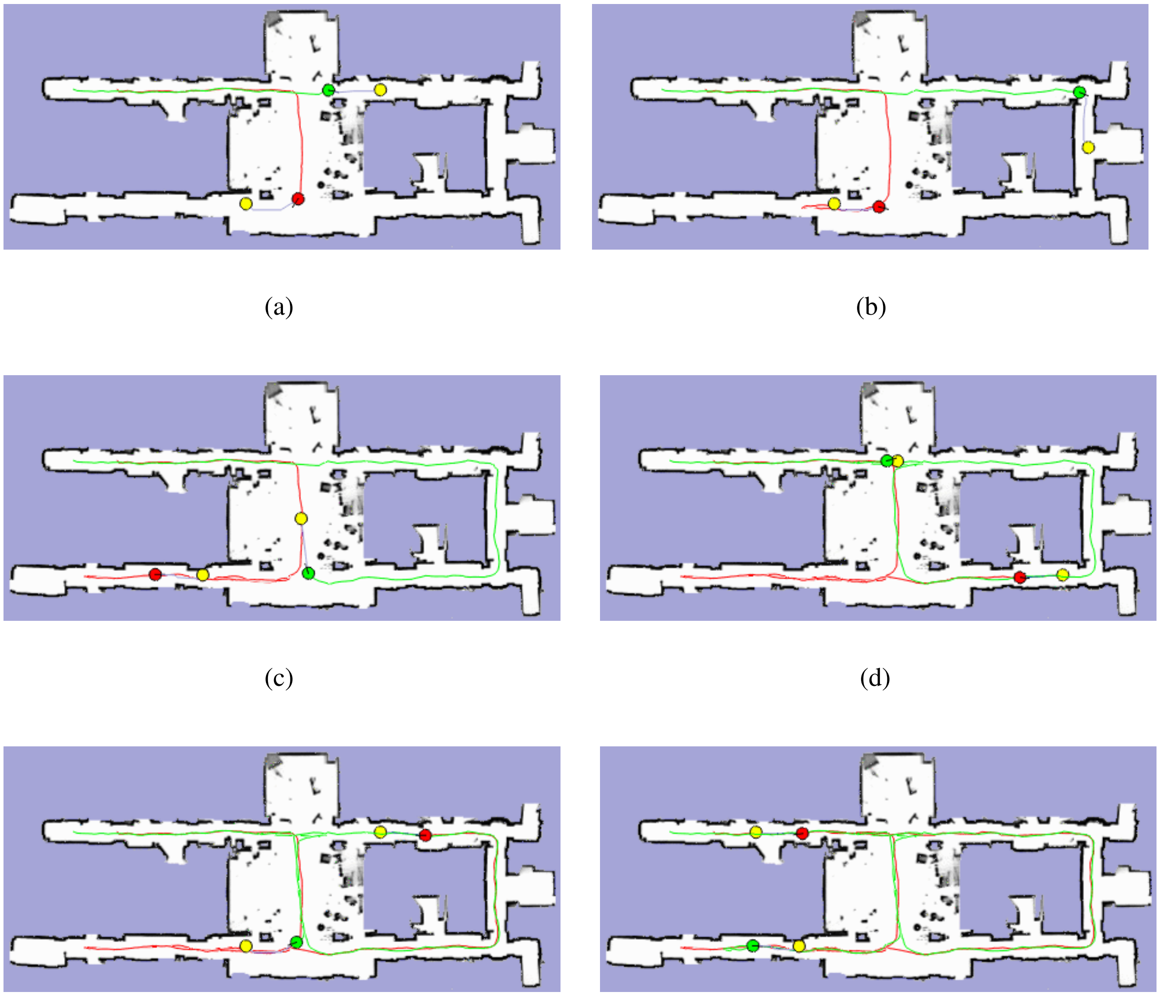
(c)

(d)

(e)

(f)

Figure 6.3: Example of Q_{MDP}-controlled robot trajectories for *Simple Gates Tag*. With the Q_{MDP} heuristic, the robots start to cover the space well but, after a few moves, the robots get trapped and do not make any more progress.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 6.4: Example of a BaGA-controlled robot trajectories for *Simple Gates Tag*. With the BaGA algorithm, the robots do a better job of covering the space than they did when controlled with MLS or Q_{MDP}.

upper corridor, one robot moves down the middle corridor while the other one stays near the top intersection. Once its teammate reaches the bottom of the middle corridor, the robots effectively stop moving (with some oscillation in the position of the bottom robot between Cells 6 and 5). With an opponent that is moving randomly, these are relatively good positions to be in as, with high probability, the opponent will eventually cross one of these intersections.

BaGA starts off similarly to Q_{MDP}, however, the robot that stays at the top continues to move into the loop while its teammate oscillates between Cells 5 and 6 at the bottom of the middle corridor (Figures 6.4 (a) and (b)). Once its teammate has almost completely traversed the loop, this robot then moves into the bottom corridor. The other robot finishes traversing the loop before moving back up the middle corridor towards the upper left corridor (Figure 6.4 (c)). If the opponent has still not been found, the robot in the bottom corridor turns around and starts to traverse the loop counter-clockwise, while its teammate clears the upper corridor and then moves back down to 'guard' the lower intersection of the middle corridor (Figure 6.4 (d) and (e)). It will finally move into the bottom corridor when its teammate finishes traversing the loop and moves into the upper corridor (Figure 6.4 (f)).

Runs on the physical robots generated similar paths for the same initial conditions. A slight modification to the interleaving of planning and execution was made with respect to the physical robots in order to improve the running time of BaGA. After the first timestep, robots construct the set of possible joint types and solve the corresponding Bayesian game for timestep $t + 1$ while executing the action found for timestep t. If for some reason the action fails to execute properly (e.g., the low-level controller cannot calculate a path from one cell to another so the robot does not move), the robots reconstruct the set of possible joint types, taking their true current positions into account, and then re-plan. This approach allows a robot to use a policy that was constructed assuming deterministic actions in a world where actions fail a small fraction of the time. It only worked, however, because the robots always knew their teammates' true positions. In order to properly model any non-deterministic effects of actions in real robots, the model must be enriched to include noisy actions. This and other changes lead to the *Realistic Gates Tag* problem, described next.

6.2 Realistic Gates Tag A and B

These two variants of *Realistic Gates Tag* capture the fact that each physical robot used for experiments has only one laser range finder and so while each robot can see forward (and to some extent to the sides), it cannot see backwards. This addition to the model results in a much larger state and observation space: each robot's state now includes both its cell location and a discrete heading (orientation). Along a corridor, a robot can face in either direction, while at an intersection it can have a heading in all four cardinal directions. The opponent is still modelled as in *Simple Gates Tag*. This additional state variable for each robot increases the size of the problem from the 18252 world states of *Simple Gates Tag* to 97200 states.

Robots can no longer move in the four cardinal directions but instead have the action set $A_i = \{$*do-nothing, move-forward, turn-clockwise, turn-counterclockwise, tag*$\}$. As expected, the *do-nothing* action results in no change in the robot's state, while *move-forward* moves the robot forward one cell in the direction it is facing unless there is a wall in the way. The *turn-clockwise* and *turn-counterclockwise* actions cause the robot to turn in place and change its heading. Along a corridor, these two actions have the same effect, while at an intersection they allow the robot to select the direction it rotates. If a robot wishes to turn 180^o at an intersection, it must select either *turn-clockwise* or *turn-counterclockwise* twice in a row. The *tag* action and the opponent policy are as in *Simple Gates Tag*. Unlike *Simple Gates Tag*, however, all actions except for *tag* have a known, non-zero probability of failing.

To take into account the effects of heading on observations, a robot can now sense its current cell location and the cell immediately in front of it. This model corresponds to robots that have a sensing range of about $5m$. As with *Simple Gates,* robots still have perfect knowledge of their own cells.

In the first variation of this problem, *Realistic Gates Tag* A, observations are noisy. The robots still know the state of their teammate (both position and heading), but they may not sense the opponent even if it is within their visible range. Positive detections of an opponent, however, are always correct. This type of observation noise was chosen because it models the fact that the robot's lasers only have a 180^o field of view and so, if the opponent moves into a cell behind a robot, the robot may not see it. If the opponent is detected by the laser, however, it is really there. This observation model, in combination with noisy actions, results in an upper bound on the branching factor of joint histories of seven. That is, at timestep $t+1$ there is at most seven times the number of possible joint types that there were at timestep t. This number is more than twice the branching factor of *Simple Gates Tag*.

The second variation of this problem, *Realistic Gates Tag* B, looks at the effects of not knowing the location of one's teammate. The robots' sensors are not noisy in this version (helps keep the problem more manageable), but the robots do not know the state of their teammate, beyond its initial state, unless they sense it. The assumption is made that sensing the teammate provides enough information to infer both its heading and its cell location. This combination results in a branching factor of up to twenty, making *Realistic Gates Tag* B substantially larger than variation A.

The fully observable policy of Section 6.1 was recalculated to take into account the effects of noisy actions. The Q-values generated by this policy were used for the utility function of BaGA as well as for the MLS and Q_{MDP} controllers. In the case of variation B, these two heuristic controllers require the robots to maintain an estimated position of their teammate. This estimate is updated using the joint action selected by the controller for the robot at each timestep (i.e., the robot assumes its teammate will implement the same joint action as it) and any positive observations of the teammate.

Due to the larger size of these problems, BaGA-Cluster was used to reduce the number of histories tracked at each timestep and therefore keep the controllers operating in real time or better. For Variation A, low-probability pruning, low-probability clustering and minimum-distance clustering were all implemented. The threshold parameter used for each condition was selected such that similar performance was generated for all three cases. The resulting parameters, however, generated a very different number of histories being maintained for each robot over the problem. As shown in Table 6.2, minimum-distance clustering with a maximum-allowable expected-loss threshold of 0.05 was the most computationally efficient of the three conditions, as each robot maintained, on average, about 50 histories over the entire problem. In comparison, low-probability pruning (with a threshold of 0.00025) had to maintain about 167 types per robot to achieve the same performance. All three BaGA-Cluster conditions outperformed the MLS and Q_{MDP} conditions given random starting locations for the robots and opponent.

BaGA-Cluster and Q_{MDP} do better in *Realistic Gates Tag* A than they did in *Simple Gates Tag*. In Variant A, the robots can now see the opponent if it is in the cell in front of them and not just their own cell. This addition helps reduce the number of configurations in which robots get 'trapped' due to the multi-modal nature of the belief state, which in turn impacts overall performance. As a result, BaGA-Cluster caught the opponent in about two fewer steps on average while Q_{MDP} improved dramatically by almost 10 steps. MLS, however, performed about the same. It has less to gain from the increased observation capabilities than BaGA-Cluster and Q_{MDP} because, in both types of tag, the MLS-controlled robots follow each other around rather than taking different paths from each other.

Realistic Gates Tag B is a much larger problem with respect to the number of histories that must be tracked for each robot. As a result, BaGA-Cluster was implemented with a combination of low-probability pruning with a cutoff of 0.0005 and minimum-distance clustering with a threshold of 0.001 in order to keep the number of maintained histories manageable. Furthermore, because the resulting controller runs in real time (and not faster as in Variation A), only one starting condition was used: one robot starts in Cell 6 looking down and the other in Cell 18 facing right with the opponent in Cell 12.[4] In about 3% of the trials, however, BaGA-Cluster still ran out of memory before the task was completed. The results shown in Table 6.2 for this condition are therefore averaged over the remaining successful trials. This starting condition is very difficult for the MLS and Q_{MDP} heuristics which is reflected in their poor performance in comparison to BaGA-Cluster. In the next section, example trajectories are given that show how this starting condition exposes the weaknesses in the two heuristics.

With both *Realistic Gates Tag* A and B, minimum-distance clustering had the effect of grouping together observation and action histories that were similar with respect to reward. The results of this clustering is similar to how a human would manually group together histories. For example, a history in which a robot in Cell 10 facing west sees an opponent in Cell 9 is clustered together with a history in which a robot in Cell 22 facing east sees the opponent in Cell 21. In both cases, the robot must move forward to capture the opponent. They both also have a similar distance to an intersection point and therefore similar future actions should the opponent move away from the robot. By increasing the maximum-allowable expected-loss threshold, histories with robot positions that are only similar in the short term will also be grouped together. For example, a robot in Cell 19 facing east seeing an opponent in Cell 18 will only generate similar actions to a history in which a robot is in Cell 7 facing west and seeing an opponent in Cell 6 for immediate timesteps.

Communication was also added to *Realistic Gates Tag* B at a cost of 1.0 per communication act (i.e., the same cost as a move action) in order to improve the computational requirements of this controller. In addition to fixed communication policies of communi-

[4] The number of trials required to generate statistically significant results for randomized starting conditions is much greater than that required if only one starting condition is used. Because this controller runs at real time, it was not feasible to run the required number of trials.

Table 6.2: *Realistic Gates Tag* results with 95% confidence intervals shown. Results are averaged over 10000 trials except for minimum-distance clustering in *Realistic Gates Tag* B, which is over 1000 trials. Average iterations is the number of steps it takes to capture the opponent. Total number of types per robot is the average number of individual types or clusters maintained for each robot. The number of histories not retained is the average number of timesteps either robot's true history was not in the maintained set. The initial robot positions for *Realistic Gates Tag* B has one robot starting in cell 6 facing down and the other in cell 18 facing right. The opponent starts in cell 12. BaGA-Cluster used alternating maximization with 200 random restarts to find policies.

Algorithm	**Reward**	**Iterations**	**# Types Per Robot**	**# Histories Not Retained**
Realistic Gates Tag A: know teammate state, noisy observations, random starting positions				
Fully Observable	-2.51 ± 0.15	6.91 ± 0.08	n.a.	n.a.
BaGA, low prob. pruning, cutoff 0.00025	-15.19 ± 0.43	13.09 ± 0.21	166.73 ± 4.17	0.31 ± 0.05
BaGA, low prob. clustering, threshold 0.0025	-15.48 ± 0.43	13.24 ± 0.22	118.68 ± 2.71	0.32 ± 0.03
BaGA-Cluster, min.dist. clustering, max. loss 0.05	-15.20 ± 0.43	13.10 ± 0.21	50.45 ± 0.87	1.54 ± 0.08
MLS	-21.13 ± 0.51	16.02 ± 0.25	n.a.	n.a.
Q_{MDP}	-19.91 ± 0.58	15.45 ± 0.29	n.a.	n.a.
***Realistic Gates Tag* B: do not know teammate state , noise-free observations, fixed starting positions**				
Fully Observable	-3.68 ± 0.08	7.45 ± 0.04	n.a.	n.a.
BaGA-Cluster, low prob. pruning 0.0005,min.dist. clustering, max. loss 0.001,	-12.84 ± 1.46	11.57 ± 0.51	463.40 ± 78.41	5.13 ± 0.79
MLS	-58.92 ± 0.52	32.30 ± 0.21	n.a.	n.a.
Q_{MDP}	-72.82 ± 1.00	41.44 ± 0.48	n.a.	n.a.

Table 6.3: *Realistic Gates Tag* B communication results. Results are averaged over 10000 trials with 95% confidence intervals shown. Reward is the average reward with communication costs of 1.0 per communication act factored in. Average iterations is the number of steps it takes to capture the opponent. Total number of clusters per robot is the average number of clusters maintained for each robot. BaGA-Comm used alternating maximization with 200 random restarts to find policies and a combination of low-probability pruning with a threshold of 0.0005 and minimum-distance clustering with a maximum-allowable expected-loss threshold of 0.001.

Communication Decision	**Algorithm**	**Reward**	**Iterations**	**# Comm. Acts**	**# Clusters Per Robot**
Fixed starting conditions, robots in cells 6 and 18, opponent in cell 12					
Communicate every timestep	BaGA-Comm	-35.11 ± 0.52	11.53 ± 0.12	23.02 ± 0.25	43.06 ± 0.45
	MLS	-93.24 ± 0.59	26.30 ± 0.16	52.31 ± 0.30	n.a.
	Q_{MDP}	-35.08 ± 0.49	11.54 ± 0.12	23.09 ± 0.25	n.a.
Communicate every third timestep	BaGA-Comm	-19.89 ± 0.30	11.33 ± 0.11	8.22 ± 0.08	97.53 ± 0.95
	MLS	-60.88 ± 0.43	26.70 ± 0.16	18.46 ± 0.11	n.a.
	Q_{MDP}	-21.93 ± 0.36	12.10 ± 0.14	8.74 ± 0.09	n.a.
$\|C_i^t\|$>20	BaGA-Comm	-15.09 ± 0.25	11.24 ± 0.11	3.59 ± 0.03	119.95 ± 1.33
PD	BaGA-Comm	-18.72 ± 0.44	11.30 ± 0.11	6.97 ± 0.08	103.62 ± 1.47
bad match	BaGA-Comm	-13.70 ± 0.26	11.75 ± 0.12	1.23 ± 0.03	301.40 ± 4.37
Random starting conditions					
Communicate every timestep	BaGA-Comm	-35.49 ± 0.79	11.65 ± 0.19	23.20 ± 0.38	46.50 ± 0.72
	MLS	-42.73 ± 0.90	13.51 ± 0.22	27.03 ± 0.45	n.a.
	Q_{MDP}	-35.61 ± 0.72	11.68 ± 0.18	23.36 ± 0.36	n.a.
Communicate every third timestep	BaGA-Comm	-19.19 ± 0.50	11.06 ± 0.19	8.07 ± 0.12	101.35 ± 1.75
	MLS	-26.13 ± 0.62	13.64 ± 0.23	9.79 ± 0.15	n.a.
	Q_{MDP}	-20.80 ± 0.49	11.66 ± 0.18	8.47 ± 0.12	n.a.
$\|C_i^t\|$>20	BaGA-Comm	-15.94 ± 0.44	11.43 ± 0.19	4.03 ± 0.07	142.49 ± 2.48
PD	BaGA-Comm	-20.37 ± 0.89	11.46 ± 0.19	7.61 ± 0.15	138.19 ± 2.76
bad match	BaGA-Comm	-14.47 ± 0.39	11.91 ± 0.18	2.12 ± 0.05	347.87 ± 7.22

cating every timestep or every third timestep, three run-time communication decisions were also implemented. The first was the cluster-threshold condition in which a robot communicates if it has more than 20 history clusters on the current timestep ($|C_i^t| > 20$). The second is the policy-difference (PD) condition in which a robot communicates if doing so would result in a different policy for either itself or its teammate. Finally, a third communication decision was implemented in which a robot initiates a forced synchronization if there is no good match between its true history and one of its retained histories. This 'bad match' condition is triggered if the difference between the reward profiles of the true history and its *best* match amongst the history clusters is larger than a specific threshold. Unlike the previous communication decisions, if the bad match condition is triggered, a robot does not tell its teammate the cluster to which its true history belongs. Rather it tells its teammate its true history and, in turn, its teammate tells it the teammate's true history. This sharing results in the robots having identical information about the current situation (i.e., one joint type). While it is a more expensive operation, each robot still only incurs a cost of 1.0 each if such a forced synchronization occurs. This communication condition was considered because it can help compensate for the aggressive low-probability threshold used for BaGA-Comm: if a robot's true history has effectively been pruned from consideration then it can 're-initialize' the set of possible individual and joint types. Experimental results are not included for expected value difference (EVD) for this problem because it resulted in communication so rarely (i.e., communication does not have a big effect on performance in this domain so EVD was rarely triggered), that the trials took too long to run. EVD was run in combination with the communicate if $|C_i^t| > 20$ condition for the fixed starting conditions; however, this latter condition triggered all of the communication acts and the outcome was essentially identical to simply applying communicate if $|C_i^t| > 20$ by itself.

While for the initial starting configuration, communication of any kind does not result in BaGA-Comm capturing the opponent any sooner, as seen in Table 6.3, it does result in many fewer clusters being tracked for each robot. Without communication, each robot must maintain about 450 clusters over the course of the problem. When communication is added, this number drops to as low as 43 for the communicate-every-time-step condition. It is not surprising, however, that adding communication does not affect the time taken to capture the opponent. Unlike the *Robot Tag* problem of Section 5.3.2, only one robot is required to tag the opponent and the opponent is moving randomly. Therefore, knowing the cluster to which its teammate's true history belongs has less of an impact on a robot's decisions than it did in the *Robotic Tag* case.

The results shown in Table 6.3 factor the cost of communication into the average reward achieved by each algorithm for the team. The bad-match communication decision, while retaining the greatest number of histories and therefore being the most computationally expensive, also results in the least number of communication acts and therefore has the highest average reward for any of the BaGA-Comm communication conditions. Out of all the conditions, $|C_i^t| > 20$ is able to strike a good balance between computational requirements and number of communication acts.

MLS and Q_{MDP} are more affected than BaGA-Comm by the addition of communication every timestep or every third timestep. Without communication, it takes MLS about 32.3 iterations to catch the opponent but with a fixed communication policy this performance improves to about 26 iterations. Q_{MDP} shows an even more drastic improvement from an average of 41.4 timesteps to about 12. For both of these heuristics, communication directly affects the belief state built up by each robot and therefore has a bigger impact on action selection than it does for BaGA-Comm. For example, with this specific starting condition, Q_{MDP} without any communication suffers from the oscillating problem previously described as a weakness of Q_{MDP} in general. By sharing observations from the very beginning, robots are able to avoid the belief state that leads to this problem and make progress towards the goal.

The resulting speed-up in planning generated by adding communication allows BaGA to be run for random starting conditions and not just the single fixed starting conditions tested in the no-communication case. As with the fixed starting conditions, BaGA-Comm was run with low-probability pruning and minimum-distance clustering with parameters of 0.0005 and 0.001, respectively. For the fixed communication decisions, BaGA-Comm performs as well as, or better than Q_{MDP}, and much better than MLS. This finding makes sense because BaGA-Comm is using Q_{MDP} as its utility function for approximating the expected future value of actions. As with the fixed starting conditions, the run-time communication decision of $|C_i^t| > 20$ represents a good compromise between computational requirements and overall performance. While it cannot be verified, it is believed that, as with the fixed starting conditions, communication does not have a great impact on the overall time taken to capture the opponent for this domain, but instead just reduces the number of clusters tracked by the algorithm. This reduction directly translates to a reduction in algorithm's computational overhead, allowing the controllers to run faster.

6.2.1 High-Fidelity Simulation and Physical Robot Results

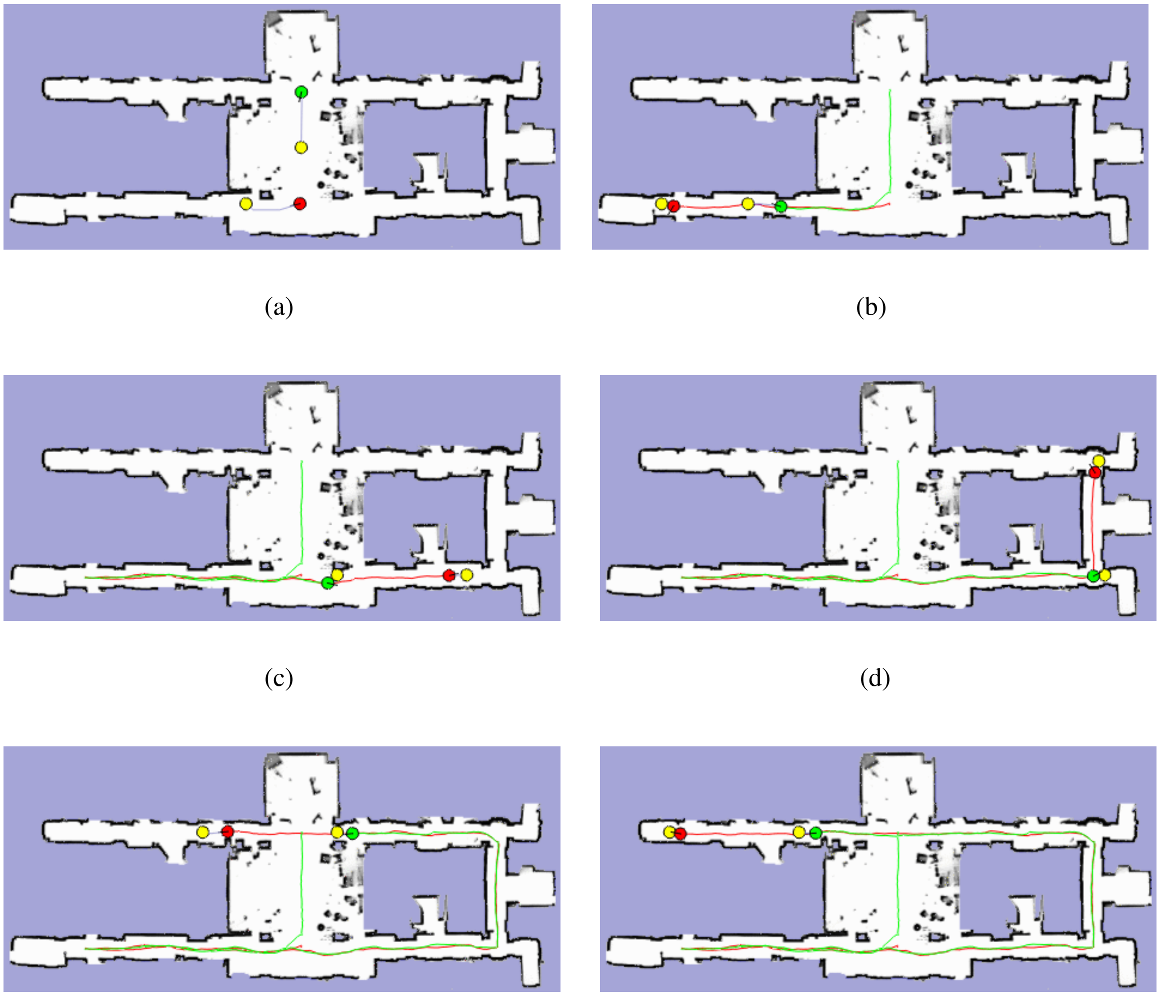
As with *Simple Gates Tag*, the BaGA controller was executed in both a high-fidelity simulator and on physical robots using the Carmen software package for low-level control. As in *Simple Gates Tag*, an additional layer was used to map between the grid world and continuous world. The heading of each robot, as determined by its localized position within the map of the Gates Building, was converted into one of the allowable cardinal directions as given by its corresponding cell in the grid world. While actions were modelled as noisy in the grid-world version of this problem, no artificial noise was added into actions in the simulator or on the real robots. For the specific starting conditions used for *Realistic Gates Tag*, actions rarely failed in the simulator and never on the physical robots. Therefore, while the robots still tracked histories in which actions failed, robots rarely had true histories for which this occurred. This does not negate the validity of modelling non-deterministic actions: previous experience with *Simple Gates* and the physical robots indicate that modelling this noise is important in general.

Overall, the BaGA controllers for *Realistic Gates Tag* A and B show some similar behaviour to those of *Simple Gates Tag*. For example, one robot will ‘guard’ the intersection while the other one traverses the loop. Figures 6.5, 6.6 and 6.7 show the trajectories taken by the robot team in simulation for the fixed initial starting conditions of *Realistic Gates Tag* B with no communication. The opponent is not present in this specific set of runs as it allows one to understand how the team searches the space for the opponent.

This starting condition is very difficult for the MLS and Q_{MDP} heuristics. With MLS, the two robots have somewhat similar starting belief states and so follow each other around. As a result, they continue to have similar belief states and continue to follow each other. As seen in Figure 6.5, the two robots move into the bottom left corridor before retracing their steps to the intersection and moving around the loop. Finally, they move into the upper left corridor. Unless the opponent starts in the bottom left corridor, this path is a fairly inefficient way to move around the space and gives the opponent lots of ways in which to move around the environment without encountering a robot.

With the Q_{MDP} heuristic, shown in Figure 6.6, the two robots turn to face the center of the environment and then never move again. They each now have a multi-modal belief state that results in the *do-nothing* action having the best expected reward. In part, this starting condition was chosen because it so clearly shows this flaw of the Q_{MDP} heuristic.

While BaGA-Cluster uses Q_{MDP} as a utility function, the additional step of game-theoretic reasoning allows the robots to overcome this ‘trap’ as shown in Figure 6.7. Instead, the



(a)

(b)

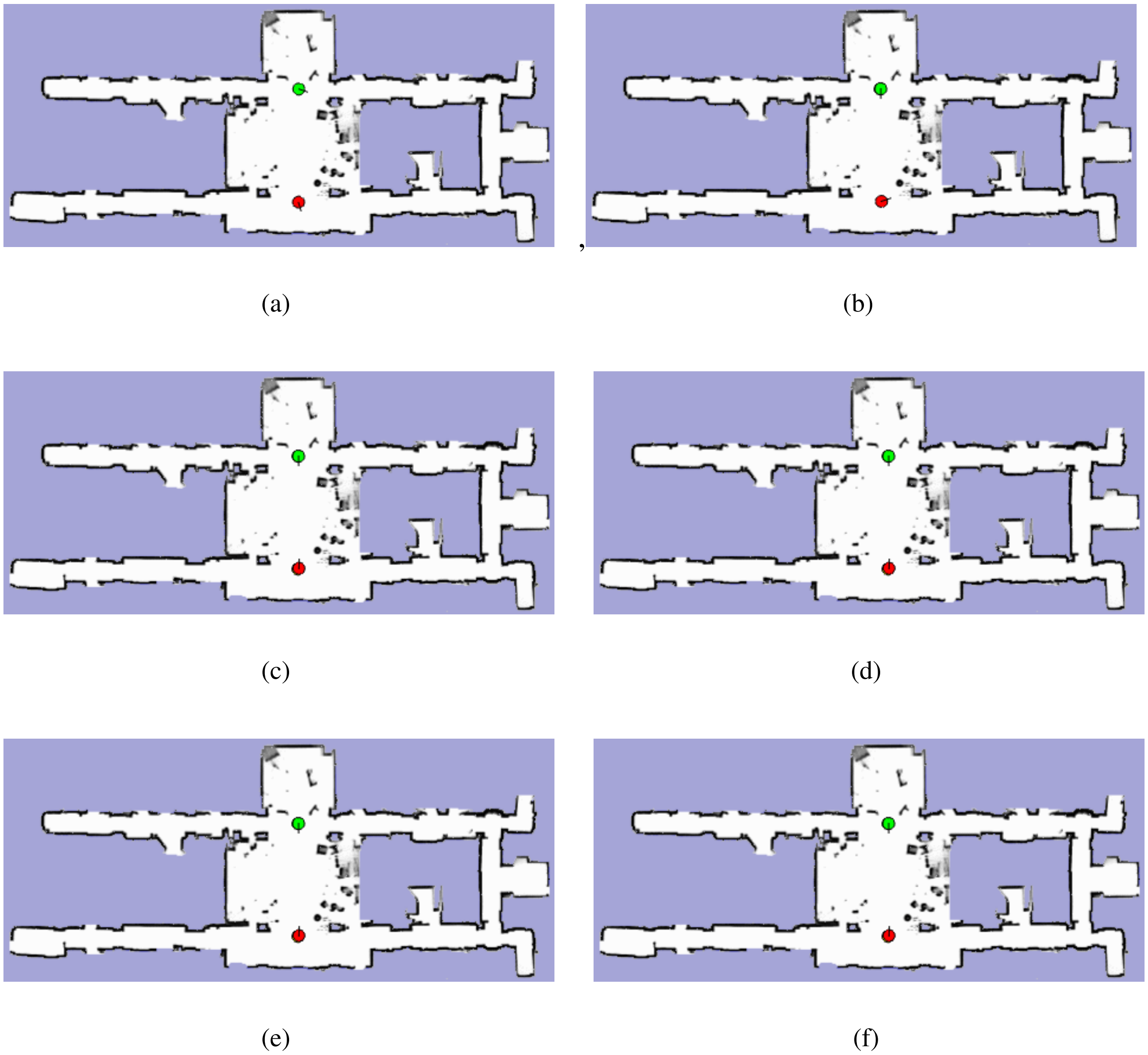
(c)

(d)

(e)

(f)

Figure 6.5: Example of MLS-controlled robot trajectories for *Realistic Gates Tag* B (no communication). The MLS heuristic results in paths where the robots follow each other and so do not cover the space well.



(a)

(b)

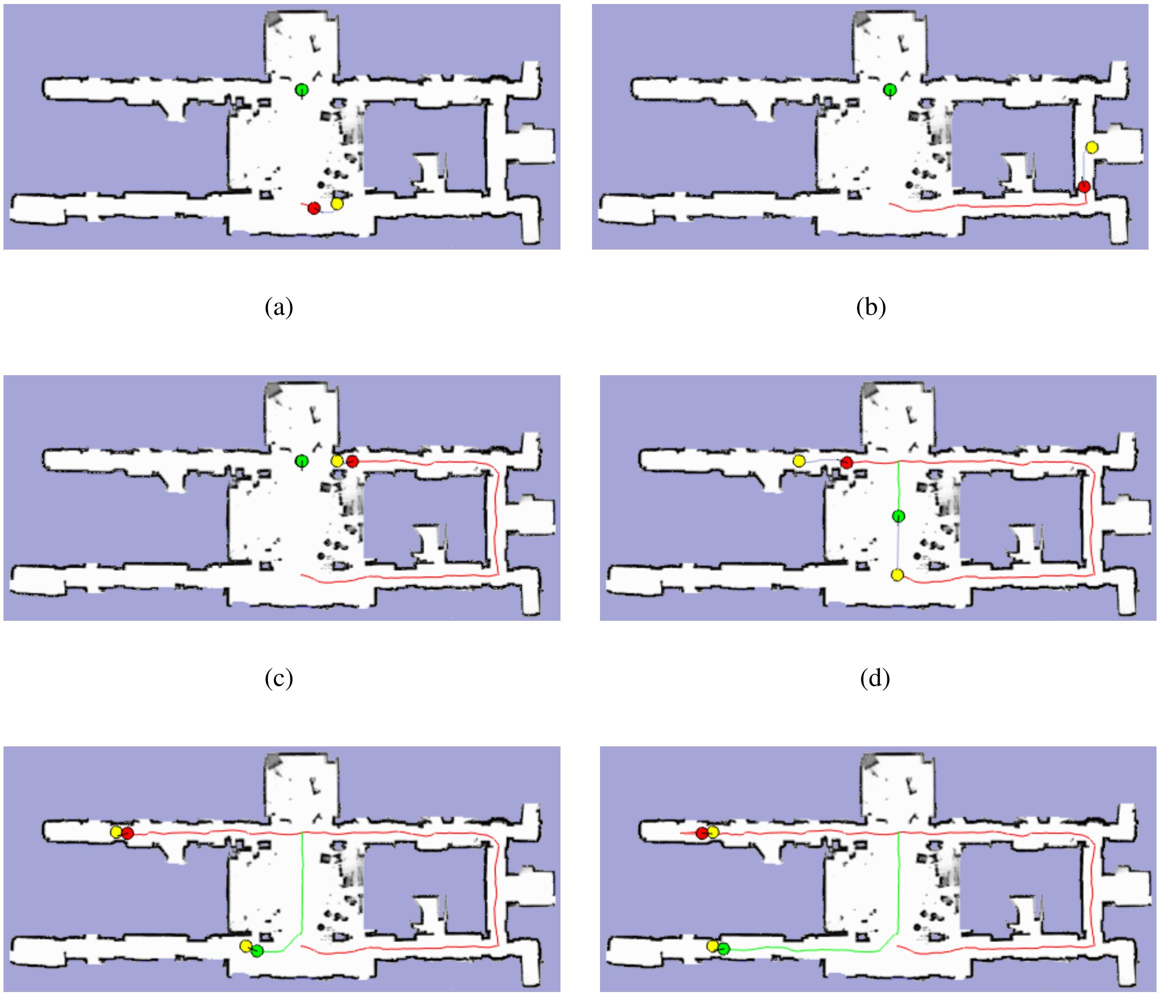
(c)

(d)

(e)

(f)

Figure 6.6: Example of Q_{MDP}-controlled robot trajectories for *Realistic Gates Tag* B (no communication). With the Q_{MDP} heuristic the robots stop moving after they turn to face each other and do not make any more progress.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 6.7: Example of a BaGA-controlled robot trajectories for *Realistic Gates Tag* B (no communication). With the BaGA algorithm, the robots coordinate to do a better job of covering the space than they did when controlled with MLS or Q_{MDP}.

top robot turns to face the center of the environment and ‘guards’ that intersection, while the other robot moves around the loop in a counter-clockwise direction (Figures 6.7 (a) through (c)). Once it reaches the top intersection, it continues into the upper left corridor while its teammate moves down and into the lower left corridor (Figure 6.7 (d)). It is a more effective way to cover the environment than the trajectories generated by MLS. Under BaGA-Cluster control, the robots still do not perfectly cover the environment (it is impossible given the sensor range and the number of robots), but they seem to leave fewer openings for a randomly moving opponent to escape detection by splitting the work between them.

Runs on the physical robots generated similar paths for the same starting conditions. Runs were made both with and without the presence of the opponent. If the opponent starts in Cell 12, then BaGA-Cluster will catch it the fastest because it sends the bottom robot around the loop while ‘guarding’ the top intersection with the other one. The opponent can therefore not pass by either robot without being captured. With MLS, depending on how the opponent moves, the time to capture can vary but is, on average, longer than with BaGA-Cluster because the robots are less efficient about searching the space. With Q_{MDP}, the robots must wait for the opponent to randomly move into one of the cells occupied by the robots because they never move on their own.

6.3 Team Gates Tag

In this version of *Gates Tag*, the robot team is expanded to three robots to show how BaGA scales to a 3-robot controller. Each robot has the same local state, action and observation space as in *Realistic Gates Tag*, but both actions and observations are now considered to be deterministic rather than noisy (unlike the case of *Simple Gates*, none of the simulator or physical robot runs examined in this section resulted in cases where this assumption was violated). The addition of a third robot increases the size of the problem from 93600 states in *Realistic Gates Tag* to 5832000 states. As with *Realistic Gates Tag* B, the robots do not know the position of their teammates with certainty but instead receive observations of their teammates.

In *Team Gates Tag*, the opponent no longer moves with Brownian motion but instead follows a known, stochastic policy in which it tries to maximize its distance to the robot team. This change was made because, in *Realistic Gates Tag*, the Q_{MDP} controlled-robots were still able to successfully tag the without moving simply because the opponent would eventually wander past one of them. Furthermore, tagging the opponent now requires at least two of the robots to coordinate on the *tag* action. Similarly to *Robotic Tag* (Sections 3.4.3 and 5.3.2), one of the tagging robots must be in the same cell as the opponent with at least one other teammate in the same or adjacent cell. Robots must now coordinate on both the *tag* action as well as on finding the opponent and 'herding' it to a location in which it can be tagged by multiple team members. Unlike *Robotic Tag*, the opponent can be captured in any cell.

As with *Realistic Gates Tag* B*,* BaGA-Comm was used with both low-probability pruning and minimum-distance clustering to keep the number of histories for each robot manageable. Communication, with a cost of 1.0 per act, was also permitted between team members to ensure real-time controllers. Because actions are deterministic, *Team Gates Tag* has a branching factor on the number of joint types with an upper bound of seven rather than the twenty of *Realistic Gates Tag B*.[5] However, the increased number of robots, possible joint actions and average length of time to capture the opponent makes this problem more computationally expensive to solve.

The utility function for BaGA-Comm comes from the Q-Values for a fully observable version of the problem. These were found using dynamic programming on an infinite-horizon version of the problem with a discount factor of 0.95. In order to find this policy, the assumption was made that, in the fully observable problem, the robots are effectively interchangeable. That is, it is assumed that if the global state $\{r_1 = (12, north), r_2 = (15, west), r_3 = (21, east), op = 25\}$ results in an optimal joint action of {*move-forward, do-nothing, turn-clockwise*}, then the global state $\{r_1 = (21, east), r_2 = (12, north), r_3 = (15, west), op = 25\}$ will result in an optimal joint action of {*turn-clockwise, move-forward, do-nothing*}. This assumption reduces the state space over which a fully observable policy must be calculated from 5832000 to 1021140.

Unlike *Simple Gates Tag* and *Realistic Gates Tag*, MLS rather than Q_{MDP} was used as the heuristic for evaluating the utility of actions in BaGA-Comm. The value of taking a set of actions $\{a_1, a_2, a_3\}$, given the joint history θ and the resulting belief state b_θ it induces over the position of the opponent in the environment, is now:

$$u(a_1, a_2, a_3, \theta) = Q(\{r_1, r_2, r_3, s_{\max}\}, \{a_1, a_2, a_3\})$$

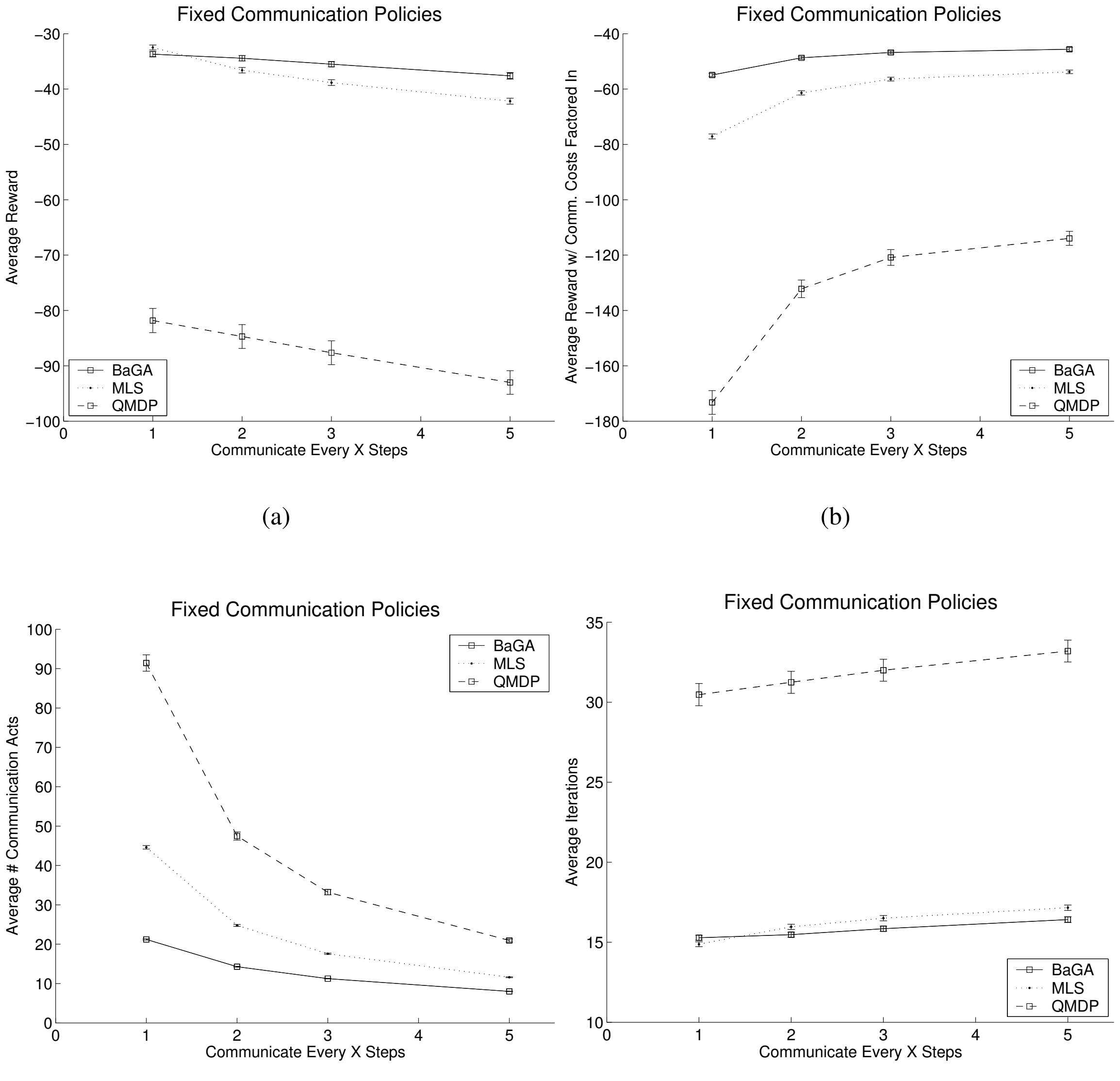
[5] Because each joint type highly constrains what observations of opponents and teammates are possible at $t + 1$, the majority of the 27^3 possible one-step extensions will not occur. Thus, the branching factor is only seven.

where $s_{\max}$ is the most likely state in s_{op} given the belief b_θ, and its associated Q-value is $Q(\{r_1, r_2, r_3, s_{\max}\},\{a_1, a_2,a_3\})$. During experimental runs, it was found that for this problem the MLS heuristic greatly outperformed the Q_{MDP} heuristic. Because the overall performance of BaGA depends on the strength of the heuristic used for evaluating utility, the decision was made to use MLS. It is important to note, however, that if BaGA-Comm uses Q_{MDP} to evaluate utility, it is still able to improve on the performance of the Q_{MDP} heuristic applied without any game-theoretic reasoning.

Figure 6.8 shows the performance of BaGA-Comm, MLS and Q_{MDP} for fixed communication policies. BaGA-Comm was run with a combination of low-probability pruning with a cutoff of 0.000005 and minimum-distance clustering with a maximum-allowable expected-loss threshold of zero, except for the case of $x = 5$ in which case the parameters were 0.0005 and 0.001, respectively. Unlike *Realistic Gates Tag* B, communication has a positive effect on performance. Because the opponent's policy is dependent on the positions of the robot team and because at least two robots must coordinate to tag the opponent, it is now necessary to cooperate on more than just covering the environment. Any communicated information can only improve that cooperation.

For all but the case of full communication, BaGA-Comm was able to catch the opponent in the fewest number of iterations. If the cost of communication is factored into the average reward achieved by the team, then BaGA-Comm always performs better than the two heuristics. Because it is unnecessary for a robot to transmit its type if it has only one, fewer communication acts were required by BaGA-Comm than MLS, even when they took similar numbers of iterations to tag the opponent. With respect to reward and number of iterations, BaGA-Comm's performance varied slightly less than MLS as the number of communication acts decreased. This stability is because BaGA-Comm-controlled robots are better able to reason about the observations of teammates and their locations than those controlled by MLS. In MLS and Q_{MDP}, performance is tied to how well the robots are able to coordinate their belief states about the opponent's position, which in turn is positively affected by communication.

The effects of run-time communication decisions are shown in Table 6.4. Policy difference (PD) and expected value difference (EVD) were run in combination with the $|C_i| > k$ decision, which places an upper bound on the computational requirements of the algorithm, as well as the bad-match decision in which the robot team synchronizes on the true joint history if any one member's true history does not have a good match with any of its possible types. As expected, limiting computation through the use of the $|C_i| > k$ communication decision alone does not have as positive an impact on performance as it does in combi-



(a)

(b)

(c)

(d)

Figure 6.8: Fixed communication policy results for *Team Gates Tag*. Each fixed communication policy is to communicate every *x* steps. BaGA-Comm was implemented with a combination of low-probability pruning with a cutoff of 0.000005 and minimum-distance clustering with a threshold of zero (i.e., two clusters are only merged if they have identical reward profiles), except for the case of $x = 5$ in which those parameters were 0.0005 and 0.001, respectively. The average cumulative reward to the team without communication costs factored in is shown in (a) while the average reward with communication costs included in shown in (b). (c) shows the average number of communication acts for the team over the length of the task while (d) shows the average number of timesteps taken to capture the opponent. All results are averaged over 10000 trials with 95% confidence intervals shown.

Table 6.4: Run-time communication policy results for *Team Gates Tag*. Results are averaged over 10000 trials with 95% confidence intervals shown. The average number of clusters is the average of the summed number of clusters maintained by a robot at each timestep, i.e., $\sum_t |C_i^t|$. BaGA-Comm was implemented with a combination of low-probability pruning with a cutoff of 0.000005 and minimum-distance clustering with a threshold of zero. Results for two of the fixed communication policies are included for comparison purposes.

Communication Decision	**Capture Time**	**# Comm. Acts**	**Reward**	**Reward+Comm. Costs**	**Average # Clusters**
EVD + $\lvert C_i\rvert > 20$ + bad match	14.81 ± 0.15	4.66 ± 0.04	-32.40 ± 0.45	-37.06 ± 0.48	94.30 ± 0.86
EVD + $\lvert C_i\rvert > 10$ + bad match	14.89 ± 0.15	5.80 ± 0.05	-32.64 ± 0.45	-38.44 ± 0.49	62.66 ± 0.53
PD +$\lvert C_i\rvert$> 50 + bad match	15.00 ± 0.15	11.65 ± 0.10	-32.88 ± 0.45	-44.54 ± 0.52	64.37 ± 0.79
PD + $\lvert C_i\rvert$> 20	15.00 ± 0.16	11.88 ± 0.11	-32.95 ± 0.48	-44.83 ± 0.56	52.80 ± 0.52
Fixed, $x = 1$	15.28 ± 0.15	21.21 ± 0.16	-33.70 ± 0.46	-54.91 ± 0.58	24.50 ± 0.20
Fixed, $x = 2$	15.51 ± 0.15	14.41 ± 0.34	-34.43 ± 0.50	-48.71 ± 0.59	32.71 ± 0.28
$\lvert C_i\rvert > 8$	15.88 ± 0.16	6.47 ± 0.05	-35.83 ± 0.51	-42.30 ± 0.55	64.18 ± 0.51
$\lvert C_i\rvert > 12$ + bad match	16.16 ± 0.15	5.53 ± 0.04	-36.49 ± 0.45	-42.03 ± 0.50	79.67 ± 0.63

nation with PD or EVD. PD, however, requires more communication acts to achieve its performance and so once communication costs are factored in, PD and $|C_i| > k$ perform more comparably to each other with respect to reward. With respect to the length of time it takes to capture the opponent, PD and EVD take about one timestep fewer to accomplish a successful tag than $|C_i| > k$ alone.

EVD in combination with $|C_i| > 10$ strikes the best balance between performance and computational costs. While it only averages about one more communication action per run than EVD with $|C_i| > 20$, each robot has about 30 fewer clusters on average, which lets the controller run faster. EVD with $|C_i| > 10$ generates a similar number of clusters and communication actions to $|C_i| > 8$; however, it has a much better performance with respect to reward and capture time.

In comparison to fixed communication policies, the PD and EVD conditions perform similarly or better to the fixed communication cases of $x = 1$ and $x = 2$ (both with respect the number of iterations taken to capture the opponent and reward); however, they require fewer communication acts to do so. With EVD and PD, the robots are actually evaluating when communication is useful as opposed to following a fixed schedule. The $|C_i| > k$ conditions, which serves to bound computational costs, perform similarly to the $x = 3$ fixed communication case, but track a higher number of possible types for each robot.

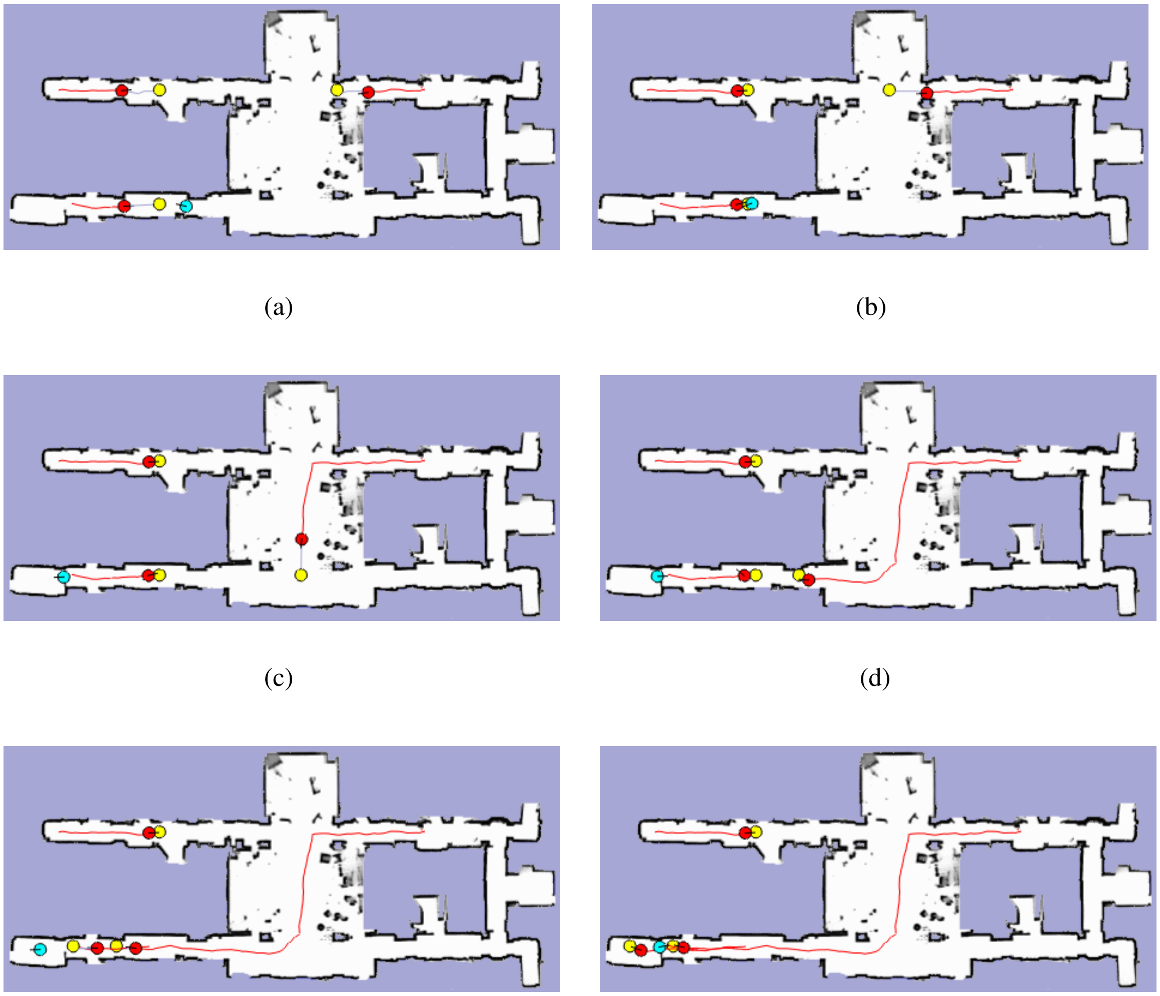
6.3.1 High-Fidelity Simulation

As with the 2-robot examples, the BaGA controller for *Team Gates Tag* was executed in a high-fidelity simulator. The mapping between the discrete and continuous worlds is the same as for *Realistic Gates Tag* B. BaGA-Comm with both the EVD and PD run-time communication decisions was used to control the simulated robots. For the two example trajectories shown in Figures 6.9 and 6.10, $|C_i| > 40$ was also included as a communication decision; however, in these cases it was never triggered. BaGA-Comm was implemented with low-probability pruning with a cutoff of 0.000005 and minimum-distance clustering with a maximum-allowable expected-loss threshold of zero.

Figure 6.9 illustrates how in this problem one robot will 'trap' the opponent in a corner until one of its teammates arrives to coordinate on a *tag* action. For this simulation run, the robots were controlled using BaGA-Comm with the EVD communication decision. A member of the robot team starts in the far left of each of the left-hand corridors, while a third member of the team is located along the top corridor, just to the right of the middle intersection. All three robots start to move towards the middle corridor. When the bottom robot senses the opponent (Figure 6.9 (b)), it lets the opponent pass it and takes up a fixed position. Because the opponent has a policy that tries to maximize its distance to the robot team, this fixed position effectively traps the opponent at the end of the bottom left corridor. At the same time, sensing the opponent triggers the EVD condition for this robot and it broadcasts its type to the rest of the team.

After the 'trapping' robot broadcasts its type, its closest teammate makes its way to the bottom left corridor to help it tag the opponent. The 'trapping' robot maintains its fixed position until its teammate draws near, at which point it turns (Figure 6.9 (d)) and moves into the end of the corridor and the two robots coordinate to capture the opponent without any communication. The third robot remains in its position in the top left corridor because it is unnecessary to accomplish a successful *tag*. By not moving, it also maintains a constant effect on the opponent's policy. For this entire run, only one communication act was performed by any robot and that was to indicate that the opponent had been seen. For these starting conditions, BaGA-Comm with a fixed communication policy generates similar robot paths but with many more communication acts.

An example of BaGA-Comm using the PD condition is shown in Figure 6.10. In this case, all three robots start in the upper left corridor of the environment, while the opponent is on the far right side of the right-hand loop. When the first two robots reach the intersection with the middle corridor, one continues straight while the other one starts to turn down. Just



(a)

(b)

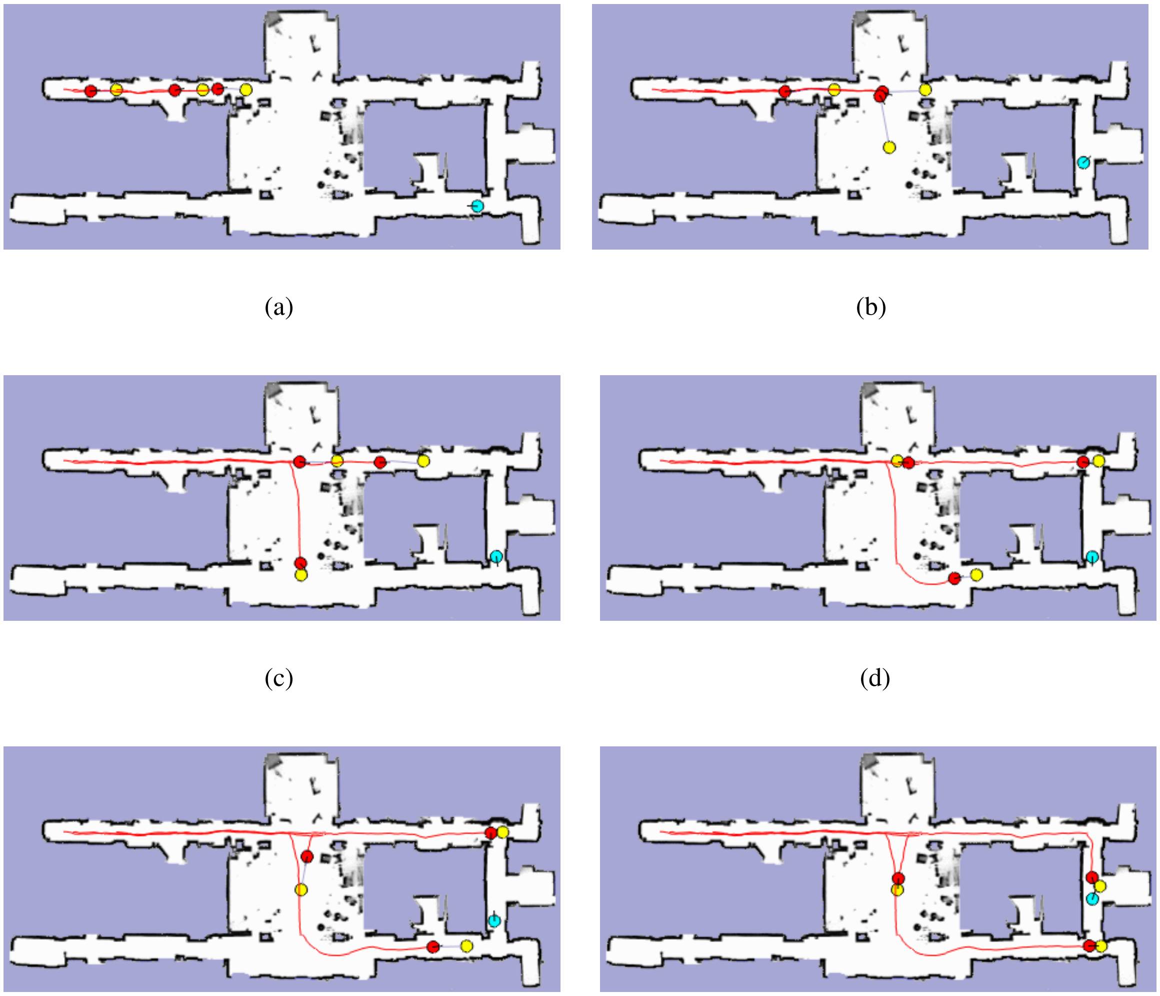
(c)

(d)

(e)

(f)

Figure 6.9: Example of BaGA-Comm robot trajectories for *Team Gates Tag*. These trajectories demonstrate how the robots coordinate to capture the opponent using the EVD communication decision. Members of the team start in three different parts of the environment and are dark while the opponent starts in the far left of the bottom left corridor and is light. The only communication act in this run takes place when the team member on the lower left observes the opponent in (b). That team member then traps the opponent in the end of the far left corridor until a teammate arrives in (d), at which point they coordinate on the tag action without the need for any additional communication.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 6.10: Example of BaGA-Comm robot trajectories for *Team Gates Tag*. These trajectories demonstrate how the robots coordinate to capture the opponent using the PD communication decision. The members of the robot team start in the top right corridor while the opponent starts in the middle of the far right portion of the loop. Key communication acts occur in (b), when neither of the two robots at the intersection see the opponent; in (e), when the team member of the top right sees the opponent; and in (f), when the team member is in the same cell as the opponent. This communication act allows the bottom right teammate to coordinate with the robot on the *tag* action even though that teammate cannot observe the opponent itself.

before these robots move out of the intersection, the PD condition is triggered for both of them (Figure 6.10 (b)). As no opponent has been sighted, the two robots continue on their way, approaching the right-hand loop from both sides. Meanwhile, the third robot stays close to the middle corridor, placing it in a good position to support its bottom teammate for an opponent capture in the bottom left corridor should the opponent prove not to be in the loop.

When the robot in the top right side of the loop senses the opponent, it broadcasts its type to the rest of the team (Figure 6.10 (e)). It again broadcasts its type when it is located in the same cell as the opponent (Figure 6.10 (f)). This information allows its teammate, located at the bottom right of the loop, to coordinate on the *tag* action with it, even though that teammate cannot sense the opponent itself (although it is in an adjacent location, it is facing the wrong way to observe the opponent). While communication acts were also triggered by PD in some other locations, overall the BaGA-Comm controller behaved similarly to how it would with full communication but with fewer communication acts.

6.4 Real-World Operation

Robust control of a team of robots in the physical world requires an algorithm to meet several criteria that do not arise in abstract problems. In order to be appropriate for real-world use, a multi-robot software framework must demonstrate inter-robot coordination, computational feasibility and scalability, robustness to uncertainty and robustness to failures and malfunctions (Dias & Stentz, 2003; Dias et al., 2004). This section discusses how POSGs and BaGA are able to meet these criteria despite the fact that these frameworks are based on relatively abstract notions of control and optimality.

6.4.1 Inter-Robot Coordination

By using POSGs to model problems, the resulting robot controllers use game- and decision-theoretic principles to achieve coordination in tightly-coupled problems. Robots do not individually select the best joint action for the team at each timestep, but rather find a set of complementary policies that dictate what action each member of the team will pick for each of its possible observation histories. While each robot does not know the exact observations made by teammates, it does know exactly what they will do in every possible case and therefore has an accurate probability distribution over their action choices, even in the absence of communication. These best action choices are found simultaneously with those of all other robots. In this thesis, coordination has been defined as finding a (Pareto-optimal) Nash equilibrium because, in the resulting policies, the robots pick actions conditioned on both their observations of the world and their (implicit) beliefs of what their teammates will do. Therefore, it is believed that Nash equilibrium policies for POSGs with common payoffs result in robot controllers that provide full inter-robot coordination.

While alternating maximization can be used on problems of limited size to find locally optimal Nash equilibria for POSGs (and therefore guarantee coordination), it is not feasible in real-world problems. Instead, BaGA can be used at the cost of another layer of approximation. This algorithm finds one-step lookahead policies by doing one full step of game-theoretic reasoning over expected action payoffs that are estimated using a heuristic. The resulting policies attempt to maximize performance of the team (i.e., the level of coordination) with respect to this heuristic rather than the true expected future value of actions. However, because each robot is using the same heuristic function, these policies are still in a Nash equilibrium with respect to the robots' viewpoints and therefore, with respect to game theory, result in controllers that are in full coordination.

It can be argued that Nash equilibria are not necessarily the best solution to a game (and therefore to the question of coordination) because, while a Nash equilibrium is stable in that no one player would wish to change its policy, it is possible that each player could gain greater reward if multiple players were to simultaneously switch their policies. As discussed in Section 2.1.6, correlated equilibria allow for agents to come up with correlated probability distributions over their sets of actions that generate higher payoffs than the independent probability distributions found in Nash equilibria. This correlation requires some sort of signal between players on which to condition action selection and, while it can be hard to motivate the use of such a signal in arbitrary games, it is easy to do so in common-payoff games where players are working towards the same goals. However, in a common-payoff game, the Pareto-optimal Nash equilibrium solution corresponds to the equilibrium in which all robots receive their highest payoff and so, unless it simplifies the search for this equilibrium, it is not necessary to move to a different type of solution. Therefore, it is believed that the Nash equilibrium solution concept used in this thesis is an appropriate way to define optimal coordination between team members. While guarantees cannot be made about playing a Pareto-optimal Nash equilibrium, a large number of random-restarts was used in both alternating maximization and BaGA to ensure that, under the same conditions, an identical high-quality solution is found.

6.4.2 Computational Feasibility

POSGs with common payoffs are computationally intractable as shown by their NEXP-completeness (Bernstein et al., 2002); however, the BaGA algorithm has been used to generate real-time controllers for robot teams. This computational feasibility does come at the cost of trading bounded-error in solution quality for the computational savings gained through the use of heuristics (bounds are only on one-step regret). Further computational savings can be made by allowing robots to transmit their observation histories (clusters) should that number exceed some threshold. Experimentally, this type of communication had a significant impact in the time taken by the controllers to find an appropriate action in the *Realistic* and *Team Gates Tag* problems, but without introducing a large number of additional communication acts. While the majority of problem domains in this thesis assumed no communication is available to the team, in real-world problems this constraint can be relaxed so that the team tries to minimize bandwidth use in order not to overwhelm the system.

If computational time permits, BaGA can be extended to a multi-step lookahead rather than just its current one-step lookahead. Indeed, an any-time algorithm can be constructed that generates higher quality solutions as more time is made available to the team for computation. As discussed in Section 3.3.3, in an n-step lookahead robots find locally optimal policies of length n. These policies can be thought of as policy trees with their expected value being the expected sum of the immediate rewards over the n steps plus the expected future value of the last action given the heuristic function used. The policy π now tells robots which policy tree to follow for each of their clusters of histories and so, at run time, robots would implement the action given by the root of the appropriate policy tree. At timestep $t + 1$, robots find a new policy of length n and use it to pick their next action.

An anytime version of BaGA would not re-use computation in the traditional sense because it cannot reuse calculations made during the construction of a $(n - 1)$-step policy in the calculating of an n-step policy. Instead, some of that computation could be reused at future timesteps by giving robots the option of continuing to use a policy calculated in a previous timestep rather than compute a new policy. In this anytime algorithm, each robot finds a series of n-step lookahead solutions for the first timestep. Depending on how much time is available, robots will solve for increasing values of n, starting with one (i.e., the current BaGA algorithm), until no more computation time is available. A robot now has a policy of length n that dictates a policy tree to be used for each of its clusters and implements the action given by the root of the appropriate policy tree.

If there is enough time (and assuming sufficient space), the robots find the full T-step policy on the first timestep and no further computation is necessary. Otherwise, at timestep $t + 1$, a robot will again construct its n-step lookahead policies for increasing values of n until time runs out. It, however, does not necessarily have to start with $n = 1$ because an m-step policy at timestep t is equivalent to a $(m - 1)$-step policy at timestep $t + 1$. For example, if a 2-step policy σ_i was found at timestep t, a robot knows that the action given by $\sigma_i(h_i^t \cup o_i^t)$ would be equivalent to the action that would be found for a 1-step policy conditioned on histories of length $t + 1$. Therefore, a robot can start constructing n-step policies with $n = m$ where m is the maximum value of n reached at the previous timestep. In the worst case, only a 1-step policy can be found at each timestep and no computation reuse can be made; however if more computation time is available robots can increase the quality of their solutions. This approach allows for a nice tradeoff between the approximations made by BaGA and the computational overhead of a longer, more accurate policy. It is, however, crucial to the algorithm that all robots come up with policies of the same depth at each timestep or else it cannot be guaranteed that they remain coordinated. The assumption needs to be made that robots either have similar computational resources to each other (and therefore would be able to find policy trees of depth m in the same time) or that some function exists for relating time to policy-tree depth and robots use this to pick an appropriate depth for which to solve (a similar approach is taken by Noh & Gmytrasiewicz (2005) and is discussed in Section 7.3.1).

6.4.3 Scalability

In order to be applied to large real-world robot teams, an algorithm must demonstrate good scalability as the size of the team increases. Obviously, scalability is a concern for BaGA because it has a strong exponential dependence on the number of robots in the team. As discussed in previous chapters, BaGA-Cluster has a computational complexity of $O\big(|S|^2(|A_*||C_*|)^n\big)$ with $|C_*| \leq |Z_*|^t$. Therefore, solving each timestep of a problem with BaGA will always be exponential in the number of robots and, if no clustering can be done, in the time horizon (the use of the sequence form prevents it from being doubly exponential in time).

While BaGA does not scale as well as heuristic approaches, it has been shown in this thesis that it significantly outperforms common heuristic methods. Furthermore, it scales better than the use of the full POSG: BaGA is able to take advantage of problem dynamics to reduce the number of histories that must be maintained for each robot and therefore reduce the effects of the time horizon on computation. For example, in the *Multiple Access Broadcast Channel* problem, BaGA-Cluster was able to generate clusters for each agent that were independent of the timestep, i.e., $|C_i^t| = |C_i^k| \; \forall t, k$.

6.4.4 Uncertainty

POSGs provide a decision- and game-theoretic way to reason about uncertainty in world state and stochastic motion (i.e., noisy actuators) as well as the actions of teammates. Uncertainty in world state can be caused by parts of the environment not being accessible to the team or individual members of the team (e.g., wrong type of sensors, sensor aliasing), as well as through sensor error. No limitations are placed on the type of uncertainty in observations or motion so long as it can be modelled through a probabilistic function (i.e., a model is required). Furthermore, the POSG model places no assumptions on the type of partial observability to be modelled: members of the team do not have to receive the same observations of world state nor do they have to share them with each other. These observations, however, will be correlated due to the fact that there is some probabilistic function governing them which is conditioned on the true world state.

In BaGA, robots are able to condition their action selection on individual observation histories despite the fact that their teammates do not know them with certainty. They are able to do so in a way that takes into account what observation histories their teammates might have seen without having to explicitly represent the associated infinite belief hierarchy. Therefore, BaGA-controlled robots are not only robust to incomplete information about the environment, sensor error and actuator error, they are also robust to uncertainty in the observations, and therefore specific action choices, of their teammates.

6.4.5 Robustness To Failure

There are three major types of failure in a robotic system: communication failure; a partial robot malfunction in which the robot is still operable but in a reduced capacity or with actual errors in how it executes actions; and complete robot failure in which the robot ceases to operate (Dias et al., 2004). There is no reason in theory why a POSG model cannot include these failures as part of its world state although, in practice, size of the problem plays a limiting factor. While none of the problems in this thesis have modelled these types of failures, this section discusses how to add them to BaGA.

In POSGs and BaGA, communication is not required for robots to condition their action choices on their own observation histories; however, if communication is present, performance can be improved both with respect to quality and computation. In this thesis, the assumption has been made that communication is global and essentially failure free. But, as briefly mentioned in Section 5.1.2, a communication protocol can be introduced with robots reverting to their no-communication policy if a failure is indicated. Theoretically, if communication is not guaranteed (i.e., failure free), then common knowledge about an event cannot be achieved through any finite series of acknowledgements about the receipt of information (Fagin et al., 1999). Instead, approximate common knowledge can be achieved through the use of robust communication languages and protocols such as those described in Stone & Veloso (1998), Barbuceanu & Fox (1995) and Emery et al. (2002). These sorts of finite protocols result in both the sending and receiving robot being mutually aware that either there was successful communication or, if there was a problem, in the appropriate robots knowing that a failure occurred.

The questions of partial robot malfunction and total robot failure can be addressed by adding components representing failure into the state of each robot. For example, the state of a robot can include a bit representing whether or not it has a malfunctioning motor, communication system, and so on. Histories would now include information about these added components and the standard BaGA algorithm applied. While communication provides the most reliable source of information about the failure of others, BaGA still works even if it is not present: both communication and observations of teammates (e.g., gathered using opponent or teammate modelling (Kaminka & Tambe, 2000; Browning et al., 2002)), can be used as a source of common knowledge for updating the prior probability over joint types. Additional parameters, such as a function that relates elapsed time to the probability of certain failures, can also be incorporated.

Clearly augmenting state to include failure increases both the size of the state and observation spaces for each robot. As a result, the number of possible types for each robot will increase because each of its types must now include observations of the status of its teammates. In general, those types that include a malfunctioning status will tend to have low probability and so using minimum-distance clustering will be most effective at representing the true distribution over histories while keeping these low-probability outliers in reward space.

6.5 Summary

In this chapter, it is shown that BaGA is not limited to controlling robots in simple grid-world problems, but can also be used to control physical robots in real time. BaGA, BaGA-Cluster and BaGA-Comm were used to generate real-time controllers for robots in a series of tag problems that gradually increased in realism and size. These controllers were implemented in both a high-fidelity simulator and on physical robots. In all cases, the BaGA-controlled robots outperformed those controlled by two common heuristics for decision-theoretic problems.

Robust control of a team of robots in the real world requires that a framework demonstrates inter-robot coordination, computational feasibility and scalability, robustness to uncertainty and robustness to failures and malfunctions. Although POSGs and BaGA are relatively abstract notions of control, they still meet these requirements both in theory and, as shown in this chapter, in practice.

Chapter 7

Related Work

This chapter discusses various approaches taken to solving the problem of decentralized control of a robot team. The majority of the chapter is devoted game- and decision-theoretic frameworks and how they, and their solution techniques, relate to POSGs and the BaGA algorithm. This discussion includes a comparison of BaGA-Comm to existing algorithms for generating run-time communication policies for POSGs with common payoffs.

Several other major multi-robot frameworks are also introduced and analyzed with respect to how robots take teammates into account during decision making. While theories of teamwork, behaviour-based approaches and market-based approaches do not make use of game theory in the same way that POSGs do, such an analysis is still useful because it can help identify what parts of problems require game theory to model teammates and what parts do not.

7.1 Related Frameworks

In this section, a variety of frameworks that are related to the POSG model and its sub-classes are discussed. This discussion is fairly in-depth and is designed to show that although some of these frameworks were originally formulated with respect to their relationships to MDPs and POMDPs, they are all sub-classes of POSGs. The order of presentation was selected to move roughly from those POSG sub-classes that are the most restrictive to those that are the most general. For example, the first framework, MMDPs (Boutilier, 1999), assumes full individual observability of the problem while I-POMDPs (Gmytrasiewicz & Doshi, 2004) make no assumptions on collective observability and are

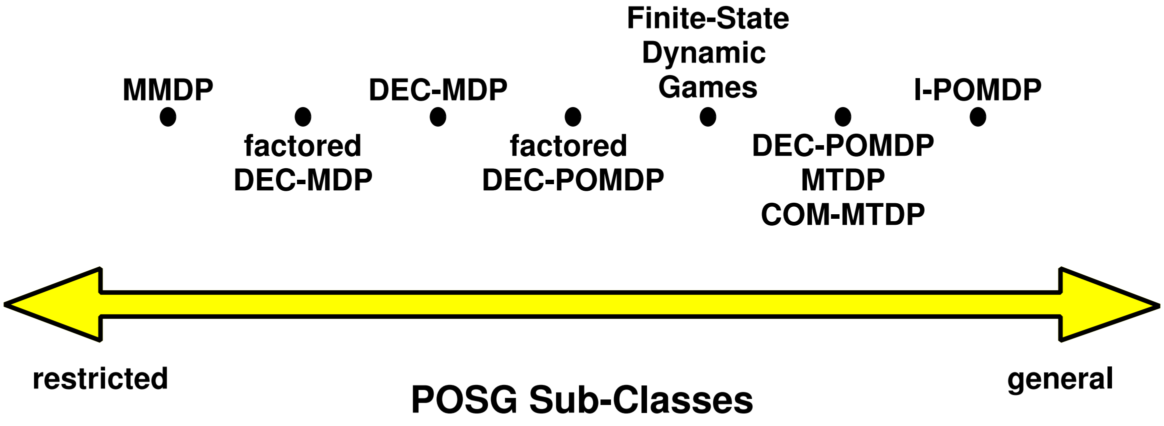


Figure 7.1: POSG sub-classes.

able to handle agents with arbitrary objective functions. An approximate continuum of where these models fall with respect to POSG sub-classes is shown in Figure 7.1.

7.1.1 Multi-Agent Markov Decision Processes

The multi-agent Markov decision process (MMDP) is an extension of the single-agent MDP to the multi-agent case (Boutilier, 1996). Agents are assumed to share the same reward function R and to have full observability of world state. Optimal joint-action policies can be found in such a system by solving the corresponding MDP in joint-action and -state space. MMDPs are perhaps the most restrictive sub-class of POSGs.

While optimal joint-action policies can be found for MMDPs in either a centralized or decentralized fashion (with each agent solving the same copy of the MMDP), execution of these policies can still result in miscoordination if there are multiple optimal joint actions. This problem is related to the problem of choosing a single Nash equilibrium to play from the set of all Nash equilibria: if there are multiple optimal joint action choices and each agent selects a different one to play, the resulting joint action can be arbitrarily bad. Boutilier (1999) looks at incorporating coordination mechanisms into the MMDP in order to solve this problem. These coordination mechanisms are protocols that restrict the focus of agents to a subset of the optimal joint actions. States in the expanded MMDP now include system state and the state of a coordination mechanism. In BaGA and alternating maximization, locker-room agreements about the random number generators are used as a coordination mechanism and, should multiple actions have the same values, robots will always select the action that comes first in the action set.

In a MMDP, structure can be exploited to find optimal or near-optimal solutions in a more efficient way than evaluating every possible joint action. In these approaches, the joint value function is broken up into a set of local value functions for each agent that are dependent on only a subset of the action and state space of the entire team (Guestrin et al., 2001, 2002; Guestrin & Gordon, 2002). Depending on the problem, the combination of these local value functions are either exact or approximate. Using its local value function, an agent then decides which parts need to be optimized jointly and which do not (e.g., through a coordination graph). Joint optimization does not always require communication between agents (Kok et al., 2005).

Alternatively, MMDPs can be thought of as stochastic games with common payoffs. In these games, agents find individual actions to play at each state by solving a normal-form game representation of that state. For a game with common payoffs, there is a Pareto-optimal Nash equilibrium for each state that corresponds to the optimal joint action of the MMDP. Stochastic games are frequently used as a way to model agents learning policies in the presence of other agents, be they teammates or opponents, in model-free settings. Examples include Nash Q-learning (Hu & Wellman, 2003), Friend-or-Foe Q-learning (Littman, 2001), Correlated Q-learning (Greenwald & Hall, 2003) and Joint Action Learning (Claus & Boutilier, 1998).

7.1.2 Factored Decentralized MDPs and POMDPs

Factored DEC-MDPs (Becker et al., 2003) are POSGs in which the global state is the cross product of the local state of each agent, and the transition function for each agent's local state is independent of the local state and actions of the other agents. Furthermore, each team member has full observability of that local state leading to collective full observability. Unlike the transition function, the reward function is not decomposable and remains dependent on the global state and the joint action of the agents. This type of problem is still NEXP-complete (Bernstein et al., 2002); however, it is possible to exploit the structure that results from transition independence.

In a 2-agent factored DEC-MDP, each agent creates an augmented MDP that is its underlying local MDP but with a modified reward function. This new reward function is based on the original reward function, including any joint-reward structure, and a policy for the other agent. In other words, it is conditioned on a policy for the other agent and so multiple reward functions may exist. As a result, each agent actually needs to define a set of augmented MDPs that are parameterized based on the modified reward function, find the set of optimal policies (the optimal coverage set) for the set of augmented MDPs, and then do an exhaustive search in the joint optimal coverage set to find the optimal joint policy. While by definition there are an infinite number of augmented MDPs (although only a finite number correspond to augmented MDPs that represent a valid policy of the other agent), the optimal coverage set is kept small because many augmented MDPs share an optimal policy. The actual determination of the optimal coverage set for each agent is done using a geometric representation. The search for the optimal joint policy is therefore reduced from a search through all possible joint policies to a search through only those joint policies within the joint optimal coverage set. At a loss of global optimality, an alternating-maximization style algorithm could be applied to a set of augmented MDPs to find a locally optimal joint policy.

One could use a factored DEC-MDP as a way of approximating a POSG that does not have transition independence. The resulting solution could then be used as a heuristic for evaluating the future value of actions in the BaGA approximation. Unlike the Q_{MDP} heuristic, which assumes that the agents will know the global state with certainty on the following timestep, this heuristic would assume that the agents only know their own state with certainty (e.g., in the *Gates Tag* problems, each robot's local state could be defined to be its own position and the position of the opponent).

The factored DEC-MDP model can be extended to include partial observability of each agent's local state. This factored DEC-POMDP model, however, still assumes that the global state space is the cross-product of each agent's local state and that both action transition functions and observation emission probabilities are independent of other agents' states and actions (Goldman & Zilberstein, 2003). In other words, agents do not have any observations of the local state of others.

7.1.3 Finite-State Dynamic Games

In finite-state dynamic games with asymmetric information (Fershtman & Pakes, 2004), each agent is characterized by a vector of state variables that evolve over time according to the player's actions (i.e., there is transition independence of local state). Some state variables are publicly observable while others are private, and agents can have arbitrary reward functions. Agents select actions at each timestep and their payoffs are dependent on the state variables of all agents and the joint action. This framework is more general than factored DEC-POMDPs because, while local state is still factored, agents now have partial observability of the local state of other agents in addition to full observability of their own state. It is commonly used for problems from the Industrial Organization literature, which are known as capital accumulation games.

Fershtman & Pakes (2004) present a simple algorithm for calculating the Markov Perfect equilibria of these games. A Markov strategy is a strategy in which actions are conditioned only on payoff-related variables rather than just any state variable. A perfect equilibrium is the same as a sub-game perfect equilibrium: even for parts of the game tree that can never be reached due to earlier decisions, the strategies for each agent define actions that are in equilibrium. Their algorithm for finding Markov perfect equilibria calculates simple sufficient statistics such that, if agents maximize their action choice with respect to them, then this action choice is the same as if the Markov Perfect equilibrium strategy was played. In this model, these statistics turn out to be somewhat equivalent to the utility function used in Bayesian games: they are the expected discounted payoff given the possible outcomes of an agent's action choice, and are computed conditioned on the private information of all the other agents (i.e., the possible state of the game as given by each agent's state vector and any other common knowledge). Like Bayesian games, if the distribution over possible state vectors is known, then the statistics can be calculated; however, if it is unknown, then the authors show it can be approximated by a reinforcement-learning algorithm.

This approach is similar to that taken by BaGA: if the expected future value of actions was known with certainty, then the actions selected by the algorithm would be the same as the actions given by the optimal solution. In BaGA, a heuristic is used to approximate that value function, while in this work reinforcement learning is used.

7.1.4 Decentralized POMDPs

DEC-POMDPs are a generalization of POMDPs to the distributed agent case (Bernstein et al., 2002). In its formalization, a DEC-POMDP is equivalent to a POSG with common payoffs. If the global state can be uniquely determined from the observations of all agents together, then the problem is of the special case of decentralized MDPs (DEC-MDPS). Unlike the factored versions of these problems discussed in Section 7.1.2, the state space is not limited to the cross-product of the local states of agents and the transition and observation probabilities are not assumed to be factored. Communication does not explicitly appear within the DEC-POMDP model.

Perhaps the biggest contribution of the work on the DEC-POMDP model is the computational complexity analysis performed by Bernstein et al. (2002), which shows that DEC-POMDPs are NEXP-complete and therefore require solution techniques other than searching for a conversion to a POMDP.

7.1.5 Communicative Multi-Agent Team Decision Problem

The Multi-agent Team Decision Problem (MTDP) is essentially the same model as a POSG with common payoffs (Pynadath & Tambe, 2002) as it applies only to problems in which all members of the team share a reward function. Like a POSG, actions are conditioned on sequences of observations (despite the somewhat confusing terminology of belief state used to refer to these observation histories), and the model parameters include an explicit definition of the set of all possible joint sequences of any length. Like POSGs, MTDPs place no restriction on the type of observability available to the team: in general, collective partial observability is assumed.

The MTDP model is extended to the Communicative MTDP (COM-MTDP) in order to explicitly model communication actions. The COM-MTDP model includes a parameter to represent the set of all possible communication messages (the cross-product of the individual agent's communication message sets) and the reward function R is now assumed to be factored into two independent parts: one coming from the domain-level actions and the other from communication. The belief states of the agents are now sets of sequences of observations and received communication messages. Pynadath & Tambe (2002) break the decision-making process into two rounds: first, the agents use their pre-communication belief states to select a communication action and then use an updated post-communication belief state to select a domain-level action. As a result, COM-MTDP is designed for *and-communication*. The authors also assume that communication is instantaneous and noise-free, which is similar to the assumptions made by BaGA-Comm.

As it is computationally intractable to find optimal domain-level and communication policies in the COM-MTDP framework (aside from the case of free communication), the authors are only able to present a theoretical way of constructing optimal communication policies. They therefore use the framework as a tool for the analysis of the optimality of team performance for problems in which a domain-action policy is given and a communication policy can be constructed from models of teamwork. This topic will be discussed in more detail in Section 7.4.

An extension of COM-MTDP that takes advantage of the use of roles for reducing the complexity of action selection and sets of domain-level actions is presented in R-COM-MTDP (Jung et al., 2002). In this model, a set of possible roles is identified for each agent. An agent's actions are now role-taking (the agent takes on a specified role) or role-execution actions. It is assumed that the set of role-execution actions for a specific role are a subset of the full set of possible domain actions and this restriction in actions can lead to computational savings. While this model allows for the analysis of different types of team formation and reorganization, it also shows that role decomposition can provide significant reductions in computational complexity. With completely decomposable roles, this approach is similar to factored DEC-POMDPs.

7.1.6 Interactive POMDPs

Unlike DEC-POMDPs and COM-MTDPs, interactive POMDPs (I-POMDPs) are designed to model agents with arbitrary objectives (Gmytrasiewicz & Doshi, 2004). This property makes them more similar to POSGs with arbitrary reward functions than to those with common payoffs. In I-POMDPs, each agent models the problem as a POMDP in which, instead of maintaining a belief over physical states of the world, the agent maintains a belief over interactive states. An interactive state is a combination of a physical state and an instantiation of a set of models for each of the other agents in the problem. These models are designed to express how the other agents will perform and therefore allow for prediction about their behaviour. By modelling the other agents in the problem, solutions to I-POMDPs are not restricted to equilibrium policies as they are in POSGs. Instead, they allow agents to condition policies on beliefs about expected behaviour. That is, if an agent believes all other agents will act according to an equilibrium then it too will play that equilibrium; however, if it has reason to believe that the other agents will diverge from the equilibrium then it can optimize its own behaviour accordingly.[1] This distinction, however, is not important when agents are in a team and it can be assumed that they are all Bayesian rational (i.e., play according to an equilibrium policy).

A general model of an agent is a function that maps from possible observation histories of that agent to a probabilistic prediction of its action choice. However, if all agents are known

[1] This idea ties into the notion that a game-theoretic equilibrium is only a 'good' solution if all agents are rational. If an agent is not playing rationally, then the other agents could do much better by exploiting that information. For example, in Rock-Paper-Scissors the equilibrium solution is to play each of the three actions randomly with equal probability. However, if the opponent only ever plays rock then one can do much better by always playing paper instead of the equilibrium solution.

to be using I-POMDP for decision making, then the models of them should reflect this fact. This goal is accomplished by using an intentional model in which the modelled agent is assigned a type and assumed to be Bayesian rational. The type of the agent is defined to be a combination of its belief (both over physical states of the world and the models of all other agents) and the remaining parameters of its I-POMDP. This type is used in the Bayesian game sense: an agent's beliefs are private information upon which decision making is based. Because each agent does not necessarily know the specific parameters of the other agents' I-POMDPs, then it must maintain a distribution over these models, much as a player maintains a distribution over the type space of other players in a Bayesian game. Unfortunately, the types of an intentional model involve possibly infinitely nested beliefs over others' types and their beliefs about others, leading to computationally infeasible solutions for I-POMDPs.

I-POMDPs are related to the recursive modelling method (RMM) in which agents try to maximize their expected utility given their beliefs (Gmytrasiewicz & Durfee, 1995). In RMMs, an agent constructs a payoff matrix that represents its beliefs about the environment, its capabilities and its utility function. In order for an agent to characterize what it thinks others will do, it also models the other agents in terms of their payoff matrices. A recursive nesting of these models, however, results because, for decision making, an agent needs to model the fact that other agents are modelling it and so on. In practice, like I-POMDPs, only a finite nesting of models is used.

In Rathnasabapathy & Gmytrasiewicz (2003), the authors propose using an I-POMDP for a common-payoff problem with socially rational agents in which agents are restricted to models with second-order beliefs (however, no experimental results are provided). That is, they propose using a model of a teammate that captures that teammate's belief about its physical state and about other agents' beliefs about their physical state. While this finite level recursion can result in sub-optimal behaviour, it is necessary to trade off the computational costs associated with deepening the belief hierarchy with the expected gain in performance. Noh & Gmytrasiewicz (2005) perform this kind of analysis for RMMs by using off-line machine learning to find a function that associates the amount of information included in the model of other agents (model depth) with expected performance gain. At run time, an agent calculates the value of time and then uses the learned function to select the appropriate model depth to use for action selection.

Given intentional models with Bayesian rationality and a shared reward function, I-POMDPs (and RMMs) can be considered equivalent frameworks to POSGs with common payoffs so long as the model nesting or belief hierarchy is infinite. Otherwise, they are approximations of POSGs.[2] The POSG model gets around the infinite recursion in the belief hierarchy by not explicitly representing a knowledge hierarchy. Approximation methods for POSGs only need to worry about the effects of truncating the belief hierarchy too early if they explicitly represent it. Otherwise, their approximations will come from other sources. For example, in the BaGA algorithm, the belief hierarchy is represented implicitly in the common prior over all possible joint histories at timestep t. In fact, it is essential to build up the BaGA in such a way as that the common prior assumption is not invalid in order not to use an explicit representation of such a belief hierarchy: work done by Sakovics (2001) shows that in order not to have a common prior assumption in a Bayesian game, knowledge hierarchies of this form must be explicitly represented (in his work, each player models its opponents as having a finite-order belief hierarchy one layer shorter than its own).

7.2 Exact Dynamic-Programming Algorithm for POSGs

In Hansen et al. (2004), a dynamic-programming algorithm for POSGs is presented that combines dynamic programming for POMDPs and iterated elimination of dominated strategies from normal-form games. One way to solve POSGs is to convert them into normal-form games and to then iteratively eliminate very weakly dominated strategies from this game until a reduced set remains.[3] A solution is then found from this reduced set of strategies. Unfortunately, because the strategy set of each agent in a POSG is $O\left(|A_i|^{|Z_i|^T}\right)$, this solution technique is computationally intractable. Instead, dynamic programming, using the policy-tree representation of agent strategies, is used to eliminate the very weakly dominated strategies without having to construct the full normal-form game. For common payoff games, this algorithm is exact and results in optimal solutions (for general-sum games, the removal of very weakly dominated strategies is not necessarily a good idea as randomization between payoff-equivalent strategies could be beneficial).

[2] The degree to which one agent models another gives its level: a k-level agent models its teammates as $(k-1)$-level agents and 0-level agents do not reason directly about others at all (Vidal & Durfee, 1997). For example, the second-order finite model for I-POMDPs proposed in Rathnasabapathy & Gmytrasiewicz (2003) results in 2-level agents. Assuming common knowledge of the game and rationality, game-theoretic agents are considered ∞-level even though they do not explicitly construct a model of teammates (Wellman & Wurman, 1998). This result, means that an agent in a POSG is only equivalent to an agent in an I-POMDP if that agent is also an ∞-level agent. Creating an ∞-level agent for an I-POMDP, however, requires infinite model nesting.

[3] Strategy *A* is weakly dominated by Strategy *B* if, regardless of what strategies *C* are picked by others, the expected payoff for *B* is at least as good as *A* and, for one instance of *C*, strictly better. Strategy *A* is very weakly dominated by Strategy *B* if *B* is at least as good as *A* regardless of *C* (i.e., the strict dominance not required).

In the multi-agent dynamic-programming operator, each backup step involves a set of depth t policy trees (if $t = T$, then the trees are just one-step trees) for each agent and a corresponding sets of value vectors. Based on the parameters of the POSG, each tree is exhaustively backed up to form a new set of policy trees and their value vectors. Any dominated tree is then pruned because, as the authors prove in this paper, if a sub-tree is very weakly dominated then the entire tree is actually very weakly dominated. That is, if at the last timestep all but one action is very weakly dominated, then any full-length policy tree that does not have that action as its last step will also be very weakly dominated. The final set of policy trees is the reduced set of strategies in which there are no very weakly dominated strategies. The problem of selecting an equilibrium amongst this reduced set of strategies still exists, but it is now possible to construct this subset without having to iterate over all possible full length strategies.

For common-payoff games, this approach is an exact algorithm that results in a globally optimal solution unlike approximation methods for POSGs that result, at best, in locally optimal solutions (e.g., in the case of BaGA the solution is locally optimal with respect to the heuristic used). While this dynamic-programming algorithm does allow larger problems to be solved than without it, they are still relatively small problems. Hansen et al. (2004) discuss pruning policy trees that are almost very weakly dominated as a way to farther reduce the number of policy trees in exchange for bounded sub-optimality. Unlike BaGA, this algorithm could place guarantees on the performance of the resulting policies.

It would be possible to make use of this multi-agent dynamic-programming operator within the BaGA algorithm as a way to extend its single step of game-theoretic reasoning over the current action selection into n steps of lookahead. As discussed in Section 3.3.3, at each timestep robots would construct a Bayesian game that represented the next n-steps of the problem rather than just one step. Rather than using alternating maximization, however, Hansen et al. (2004)'s algorithm would be used to reduce the set of pure strategies over these n steps and then a search method would be applied to find the optimal joint policy. Robots would then implement the first step of their resulting policies and re-plan. This n-step lookahead would allow robots to trade off speed in calculating the future value of actions with accuracy.

7.3 POSG Approximation Methods

In this section, a variety of algorithms for finding approximate solutions to POSGs and its sub-classes are presented. Figure 7.2 presents a high-level overview of these algorithms showing how they roughly relate to the BaGA algorithm. This figure also includes algorithms discussed in Section 7.1. Algorithms for generating run-time communication policies for sub-classes of POSGs are not included in this section or the diagram, but will be discussed in Section 7.4. Figure 7.3 provides a high-level comparison of the size of problem that has been addressed by each of the different solution techniques for POSGs with common payoffs presented in Section 7.3.1.

7.3.1 POSGs with Common Payoffs

This section summarizes different algorithms for finding solutions to POSGs with common payoffs. As BaGA falls into this class, a high-level comparison of the different algorithms and the size of problem to which they have been applied is shown in Figure 7.3. This comparison is based on published experimental results for each of the algorithms.

Rather than using value-based methods to find locally optimal policies to POSGs with common payoffs (called POIPSGs by the authors), Peshkin et al. (2000) use gradient descent to perform policy search in the space of limited-memory finite-state controllers. The assumption is made that the best joint controller in this restricted set of controllers can be represented as a factored controller. While it is not necessarily true of the globally optimal joint policy, it allows each agent to independently perform gradient descent on its own portion of the controller with the resulting joint controller being the same as if joint gradient descent had been done. Therefore, agents find the best factored controller for a domain even if it is known *a priori* that a factored controller is sub-optimal.

Furthermore, unlike the BaGA algorithm, knowledge of the world model is not assumed, and so stochastic gradient descent is used in which the algorithm samples from the distribution of observation histories by interacting with the world and uses those samples to calculate an estimate of the gradient. The resulting policy is locally optimal and, while every Nash equilibrium solution to the POSGs is a local optimum, the reverse is not true. Agents, therefore, might not play policies that form an equilibrium. In contrast, every set of policies found in the BaGA algorithm is an equilibrium solution given the heuristic function used for calculating utility.

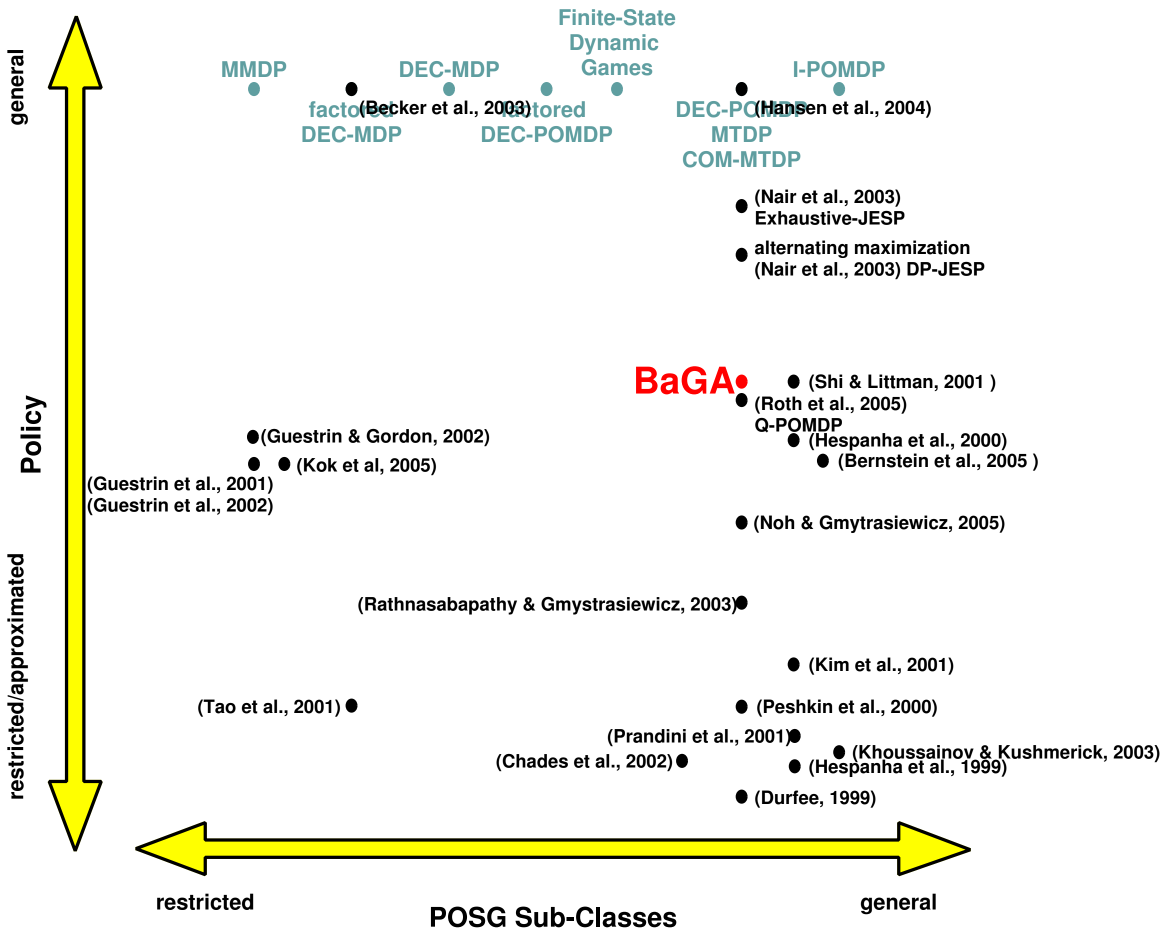


Figure 7.2: Continuum of algorithms for approximating solutions to POSGs. The policy axis is used to indicate the generalness of both assumptions made about policy structure but also the degree of approximation of the resulting solution. Therefore, an approach that places restrictions on the structure of policies but tries to find locally optimal versions of those policies (e.g., Peshkin et al. (2000)) is placed lower than an approach that has no restrictions on policy but a higher level of approximation (e.g., I-POMDP approaches that truncate the belief hierarchy (Rathnasabapathy & Gmytrasiewicz, 2003)), but higher than an approach that also also places restrictions on policy form but makes more approximations (e.g., Chades et al. (2002) who approximate a non-stationary transition function as stationary). The models discussed in Section 7.1 are included (light text), with the optimal policy for each model considered the most general type of policy for that class. Zero-sum POSGs are considered to be slightly more general than POSGs with common payoffs but less so than those with arbitrary payoffs, hence the related solution techniques fall somewhere between the DEC-POMDP and I-POMDP models with respect to the sub-class axis.

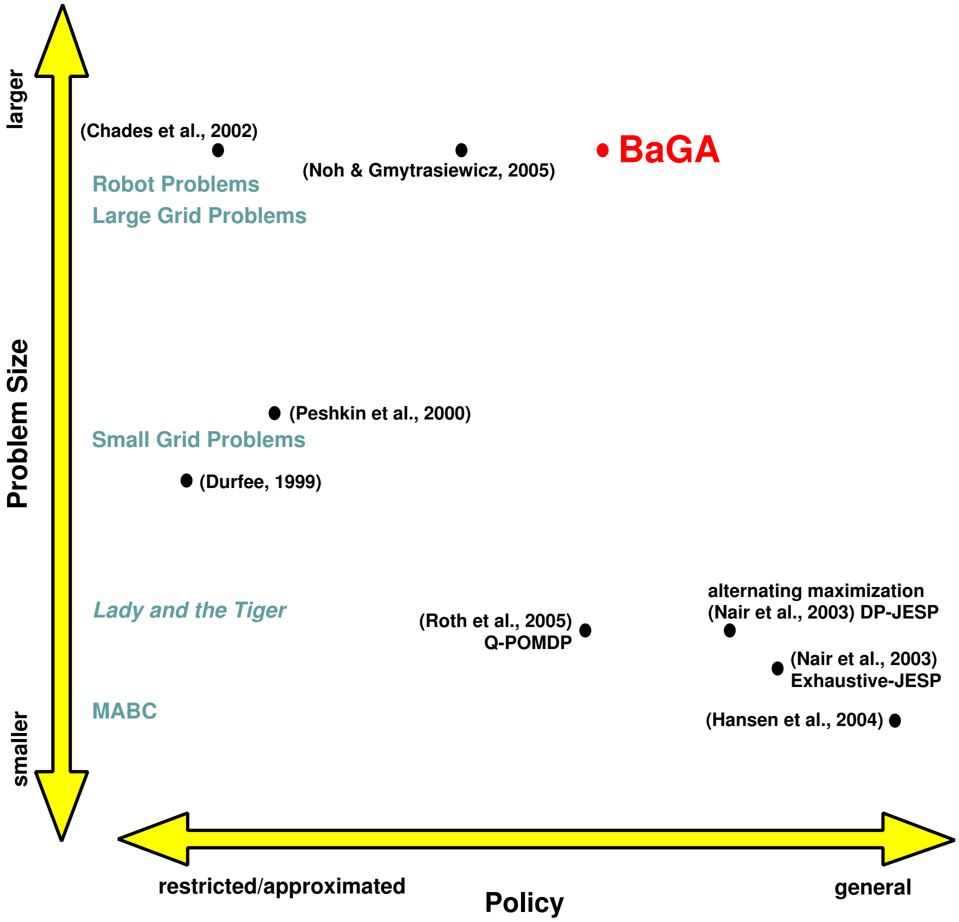


Figure 7.3: Comparison of the size of problems addressed by various algorithms for POSGs with common payoffs. The degree of approximation of these algorithms ranges from exact (Hansen et al., 2004) to RMMs with heuristics (Durfee, 1999). Rough categories of robot problems are included for reference. Problem size comes from published experimental results, with the *Lady and the* Tiger problem considered larger than MABC because, despite its smaller state space, it has been solved for larger time horizons (and with collective partial observability) than MABC.

Peshkin et al. (2000) also trade off the memory that can be stored within the factored controllers for computational tractability. In the worst case, this approach results in reactive policies in which agents condition their action choice on only the most recent observations. The BaGA algorithm does not explicitly trade off between history length (i.e., memory) and computation; however, the clustering of histories in BaGA-Cluster can be thought of as a way of limiting memory because it groups together similar histories. Especially for physical robot problems, this approach tends to reduce the effects of early observations on later decision making.

The approximations in the solutions found by BaGA and POIPSGs come from different sources. POIPSGs limit the amount of memory used in policy construction and the scope of the policy search to factored controllers. Gradient descent is then used to find locally optimal controllers within this subset of controllers. BaGA introduces approximations through the use of a heuristic for evaluating the future value of actions and the alternating-maximization algorithm to find a local optimum.

Another approach that uses finite-memory controllers is that taken by Bernstein et al. (2005) for approximating solutions to infinite-horizon POSGs. The policy for such a POSG is represented as a joint stochastic controller, which is made up of a local controller for each agent. Unlike other work with common-payoff POSGs, these policies allow agents to take stochastic actions at each timestep. With stochastic policies, correlated equilibria tend to generate higher payoffs to the team than Nash equilibria, and so a correlating device is included in the definition of the joint controller. This correlating device allows agents to correlate behaviour (i.e., achieve a correlated equilibrium) without any communication during execution. An example of such a correlating device is shared randomness, which is equivalent to the synchronization of random number generators required by BaGA. In Bernstein et al. 2005's work, a joint controller is found through policy iteration and the algorithm guarantees that on each iteration the performance of the controller will be at least as high as on the previous iteration. The authors compare results for different sized controllers and find that larger controllers usually lead to better solutions.

Nair et al. (2003) present a dynamic-programming-style algorithm for finding locally optimal policies for the full POSG that is very similar to the alternating-maximization algorithm used in this thesis, but with a different representation of the policy than the sequence form. At the core of their work is a joint equilibrium-based search for policies (JESP) that finds locally optimal policies from the set of all possible policies with finite planning horizons. That is, unlike Peshkin et al. (2000), no restrictions are placed on the structure of the policy. This planning is centralized and so the problem of selecting a policy from a set of multiple local optima is avoided. While an exhaustive version of JESP can be performed, a computationally tractable dynamic-programming version (DP-JESP) is preferable. DP-JESP first finds the best set of policies for the last timestep and then backs these values up, using an alternating-maximization-like approach.

Like alternating maximization and the BaGA algorithm, DP-JESP is designed to work for POSGs with common payoffs. Its policy representation requires slightly less space than alternating maximization and so it can be used to solve the full POSG of the *Lady and the Tiger* problem with a planning horizon one larger than that which can be solved with alternating maximization alone (a horizon of 7 vs. a horizon of 6). BaGA, however, can solve this problem with an even larger horizon. Nair et al. (2003) also tend to use very few random restarts in conjunction with DP-JESP (e.g., only three) and so make limited attempts to move out of local optima.

In their work on creating run-time communication policies, Roth et al. (2005) present a Q-POMDP heuristic for multi-agent POMDPs, which is analogous to the Q_{MDP} approximation of POMDPs (Littman et al., 1995). This algorithm requires agents to maintain a probability distribution over all possible joint histories that might have occurred up to that timestep and then implement their part of the joint action that has highest expected reward given the resulting distribution over joint beliefs. The value of taking an action given a specific joint history (and therefore joint belief) is defined to be its value in a fully cooperative POMDP solution to the problem that has been solved off-line. That is, this heuristic acts as if in the next timestep the robots will have full communication. This algorithm guarantees that the team will remain coordinated in the sense that the joint actions selected will never be one of those that are dominated in the fully coordinated version of the problem (e.g., in the *Lady and the Tiger* problem, a joint action that has the robots doing different actions will never be selected) .

Like BaGA, in Q-POMDP agents must maintain the same probability distribution over all possible joint histories. Unlike BaGA, the reason for this choice is to guarantee that the agents all select the same joint action. Because agents only pick one joint action at each timestep, they cannot integrate their true observation histories into the decision-making process unless communication can occur. Without communication, the agents make decisions conditioned only on the problem dynamics because Q-POMDP does not create conditional plans. This Q-POMDP heuristic is therefore actually intended to be used in conjunction with an algorithm for run-time communication decisions (see the discussion of DEC-COMM in Section 7.4) and is not designed for use in problems in which no communication is available to the team.

The *coop-POMDP* utility function used for the *Lady and the Tiger* problem of Section 3.4.1.2 has some similarities to the Q-POMDP heuristic in that both use a fully cooperative version of the problem to find a POMDP solution in which all agents can share all observations. In Q-POMDP, agents will then select the best joint action given that the problem will have full communication on the next timestep. If *coop-POMDP* is used as a utility function for BaGA, the agents come up with a set of conditional plans that specify the best individual action for each agent to take for each of its possible histories assuming that the agents have full communication on the next timestep. The best individual action to take for a history is the one with maximum expected reward given the distribution over all possible joint histories (and therefore joint beliefs) that is induced by that history. Unlike Q-POMDP, this approach allows the robots to condition their action selection on observed information even if communication is not available to the team.

Q-POMDP guarantees that agents will always remain coordinated in the sense that they will always choose the same joint action to implement and that that joint action will not be one that is dominated in a version of the problem with full communication. The definition of coordination in a POSG (and therefore a BaGA), however, is different: being coordinated means that agents will implement policies that are from the same set of best-response policies. This difference in the notion of what it means to be coordinated comes from the use of the joint-action space versus the individual-action space of the agent in policy definition. Communication also plays a role: if the problem has full communication then a fully coordinated policy in a POSG is equivalent to that of the Q-POMDP definition. If there is no communication, then a joint action that is dominated in the full communication case may no longer be dominated in expectation and so implementing policies that have this joint action as an outcome could lead to higher expected reward than restricting oneself to non-dominated joint actions.

Chades et al. (2002) look at a similar type of problem to the *Gates Tag* problems of Chapter 6: they model a team of robots that are trying to find an evader moving with a random policy as a POSG with common payoffs. Because of the computational intractability of this model, the authors use a heuristic approach in which they assume that the problem can be solved with reactive policies in which robots only condition their action on their most recent observation. Furthermore, the global reward function is assumed to be broken up into a sum of local reward functions for each robot (state transitions, however, are not factored as in the DEC-POMDP model). Using these two assumptions, one can convert the original reward function into local reward functions that map from observations to rewards.[4] The

[4] Because the *Simple Gates Tag* and *Realistic Gates Tag* problems only require one robot to tag the oppo-

POSG is then converted into several MDPs that are solved concurrently by each robot. This conversion is done by treating each robot in the problem as being modelled by a subjective MDP in which the state space is its set of observations in the POSG and the transition function over these states in the average probability transition between observations. This last approximation is quite significant because, while the probability transitions between observations is not stationary, it is being treated as such.

If communication is available to the robot team, then each robot initializes itself to have a random policy. One randomly selected robot will ask its teammates for their policies, integrate that information into its subjective MDP (i.e., into the state transitions) and then find its optimal policy. This procedure is then performed for the next randomly picked robot until the system settles. The integration of the policies of other agents into a subjective MDP model is not trivial but can be done.

If no communication is available to the team, Chades et al. (2002) make the further assumption that, because all robots have the same model of the world and the same local goals, every robot will have the same mapping from observations to actions. This policy is found in parallel by each member of the team using the following algorithm: 1) initialize with an identical arbitrary *a priori* policy for each robot 2) hold all but the last robot's policy fixed to π^t and find a new policy π^{t+1} for the last robot 3) reset all the other robot's policy to π^{t+1} and find the new best policy for the last agent. At some point, the algorithm will terminate at a fixed point (if the state and action spaces are finite) and the robot implements the resulting policy. Provided that each robot initializes the algorithm with the same initial policy, all members of the team will find the same policy without communication.

The two approaches to solving POSGs taken by Chades et al. have some analogies to the alternating-maximization algorithm: the policy of all but one robot is held fixed while that robot's policy is optimized. However, unlike alternating maximization, this algorithm either requires communication or the assumption that all agents will end up with identical policies. While in some problems this might be true, in general it is not. For example, in *Simple Gates Tag* the global reward function can be modelled as the sum of local reward functions; however, the robots did not end up with identical policies. Indeed, as shown by the MLS heuristic in which robots essentially follow the same policy given the same observations, such a policy could do arbitrarily badly. Furthermore, Chades et al. keep

nent, the reward function used for those problems also has this property: it is always the sum of the rewards to each agent. *Team Gates Tag*, however, does not have this property because both robots are required to tag the opponent. While certain parts of the reward function are equivalent to the sums of local rewards, those parts of the reward function that deal with *tag* actions cannot be split in this way.

policy search manageable by using only memoryless policies (giving it some ties to Peshkin et al. (2000)), which results in further approximations to the globally optimal solution.

As mentioned briefly in Section 7.1.6, Noh & Gmytrasiewicz (2005) approximate the full belief hierarchy required by a recursive modelling method like I-POMDP by trading off between available computation time and the depth of the model using a learned function relating the two. This approximation technique was applied only to problems with common payoffs rather than the more general arbitrary-payoff case. They also considered another type of approximation in which, off-line, rules were learned for reducing the action set by solving randomly generated instances of the problem with a fixed belief-hierarchy depth. At run time those rules were then applied to reduce the action set to only those actions that are known not to be sub-optimal. While the same hierarchy depth was still used for solving the problem on-line, the reduced action set led to faster computation of solutions. This work applies machine learning to learn functions, or rules, rather than the hand-coded heuristics used for making coordination practical in RMMs by Durfee (1999). For example, in Durfee's work, agents use heuristic estimates (based on previous experience) about the impact of expanding different parts of the nested model to determine whether or not expansion is necessary for the current situation. Noh & Gmytrasiewicz (2005) instead learn a function relating model depth to solution quality and then, at run time, trade off between available time and solution quality. Durfee (1999) also considered prohibiting actions at run time using organizational constraints imposed on the system as well as the use of communication to aid in deciding which parts of the models to expand or to reduce uncertainty.

7.3.2 Factored DEC-MDPs

Multi-agent OLPOMDPs take a policy search approach to factored DEC-MDPs in which distributed agents learn cooperative behaviour without explicit communication (Tao et al., 2001). The system is modeled as a factored MDP in which agents share a reward function but make action choices based only on a local parameterized stochastic policy. In order to climb the gradient of the global average reward, agents do not need access to the network topology but only their own local information and the reward signal. The authors include reward shaping in the reinforcement signal in order to penalize behaviour that would not appear in the optimal solution. This approach, however, is not valid in problems in which the reward signal is not available to agents (e.g., the *Lady and the Tiger*) and to a certain extent requires the system designer to know what actions are undesirable in order to introduce reward shaping appropriately.

7.3.3 Non Common-Payoff Problems

In this section, approximation algorithms for POSGs without common payoffs are discussed. These algorithms are of interest as they allow agents to find policies in the presence of an opponent who is actively working against them; however, because this is the most general formulation of a POSG, algorithms cannot take computational advantage of assumptions such as deterministic policies or of short-cuts in finding equilibria that are a direct consequence of a common reward function. As a result, they tend to be restricted to either two agents or two teams, each modeled as a single agent.

Poker is a well known game that can be modelled as a zero-sum POSG. As a result, some of the approximation techniques for POSGs with common payoffs, such as using alternating maximization to find a locally optimal solution rather than a more involved linear program or linear complementarity program, do not apply. Instead, researchers turn to ways of reducing the overall size of the problem in order to help make it tractable. As discussed in Chapter 3, at each timestep, BaGA performs full game-theoretic reasoning about the current action choices of the robots but uses a heuristic function to evaluate the future value of those actions. Shi & Littman (2001) use a similar approach for finding solutions for a scaled-down version of Texas Hold'Em with multiple rounds of betting. While these rounds are not independent, the authors treat them as quasi-independent: for example, in the first round of betting, players use an expected value for future cards that will be dealt in order to evaluate their betting strategy. Rather than calculate this value exhaustively, randomized simulation is performed to estimate the average expected payoff.

The authors also consider the use of abstraction to keep the combinatorial explosion of possible hands under control. Rather than solve the full game, hands of cards are grouped together and game play is then conditioned on these groups rather than on specific hands. Hands are given a rank based upon their strength in poker and grouped into equal sized bins. Each bin contains hands with similar rankings. The game is then solved using bins: rather than being assigned hands, players are assigned a bin and compute their betting strategy appropriately. Shi & Littman (2001) found that using as few bins as 10% of the total number of possible hands results in near-optimal play at a huge computational savings. This binning strategy can also be applied in the case where partial, rather than full, hands of cards are dealt. The relationship between hands and bins is analogous to that between the full set of histories and clusters in BaGA-Cluster. BaGA-Cluster, however, is a bit more flexible than the binning strategy because a maximum number of clusters is not given *a priori* (except for certain versions of BaGA-Comm). Incorporating a maximum number of clusters into BaGA is a possible extension and can be done by increasing the allowable expected loss between merged clusters until the number of clusters satisfies the maximum number of clusters allowed.[5]

There has been quite a bit of work done in the area of multi-agent pursuit-evasion games, both with respect to robot teams (e.g., Hespanha et al. (2000)) as well as in the classic game-theory literature (Bernhard et al., 1987; Olsder & Papavassilopoulos, 1988). Frequently, the assumption is made that the information available to the agents is asymmetric with the evader being all-knowing (an assumption also made in the tag domains examined in the thesis). A common approach for robot teams is to model the problem as a team of robots pursuing a single evader. Rather than treat it as a game with n agents, it is considered a game between two agents, the evader and the pursuer, with the robot team modelled as a larger meta-agent that has full inter-robot observability. Therefore, while this overall problem class does have partial observability over the global state (e.g., the pursuer does not know the evader's true position), the robot pursuer team effectively shares a brain, usually due to the assumption of free communication. The state transition probabilities for the actions of evaders and pursuers are assumed to be independent of the other player; however, clearly the observation functions cannot be.

Even assuming an omniscient evader, it is still a challenging problem as it is a zero-sum POSG. Approaches have ranged from finding optimal feedback controllers for the pursuer team assuming a known policy for the evader (Prandini et al., 2001), to greedy approaches that can guarantee that an evader can be found in finite time assuming that the evader's policy is known or can be estimated (i.e., it is not trying to actively evade capture) (Hespanha et al., 1999).

Memoryless policies dependent on probability measures over both possible obstacle maps and positions of the evader (called evader maps) are proposed in Kim et al. (2001) as a way of finding a computational feasible solution to the POSG model. In a *greedy* policy, the pursuers move the next reachable position that has the highest probability of containing an evader over all possible evader maps at the next time instant. In a *global-max* policy

[5] This procedure can be done automatically by setting the maximum-allowable expected-loss parameter to be extremely high but setting the minimum number of clusters to be equal to the maximum desired. As the algorithm will iteratively merge the two closest clusters until the minimum number of clusters is reached, this approach has the same effect as increasing the maximum-allowable expected-loss parameter until the number of clusters is below some maximum number. While the first approach does not require the expected loss parameter to be modified during run-time, it does not allow BaGA-Cluster to ever find a set of clusters that is smaller in number than the maximum as the algorithm terminates at this point. The second approach, dynamically adjusting the threshold parameter, may come up with a number of clusters smaller than the maximum, should the initial clusters be close together.

the pursuers move to the next reachable position that is closest to the cell in the entire environment with highest probability of containing the evader. These policies have some parallels to the MLS heuristic used as a comparison controller in the tag domains analyzed in this thesis.

A more principled approach is taken to the same problem by Hespanha et al. (2000) in which agents find greedy one-step policies that approximate the true solution. At each timestep the pursuer and evader create and solve a one-step general-sum game in which the pursuers are trying to maximize the probability of capturing the evader in the next timestep and the evader is trying to minimize this probability (i.e., the heuristic used for the expected value of actions is their immediate expected value). While the overall problem is zero-sum, this one-step game is general-sum because the evaders and pursuers have different sets of information: the expected value of a set of policies for the pursuers and evader would be evaluated differently by each because the expectation is done using the information known by each player. Because the evader's information set includes that of the pursuers, it could calculate the expected payoff, or cost, function used by the pursuers in addition to its own and therefore find a Nash equilibrium of the game. The pursuers, however, cannot solve this problem as they can only estimate the value of a policy to the evader.

Rather than treat this problem as a Bayesian game, in which the pursuers have a probability distribution over possible types of the evader (e.g., a type could be the evader's value function or some part of its state), a solution is found by treating the problem as the fictitious zero-sum game in which the pursuers' estimate of the evader's cost is considered to be correct. It turns out that (assuming the evader knows that this procedure is how the pursuers will solve the problem), by playing their part of the Nash equilibrium strategy in this zero-sum game, the pursuers can force a rational evader to play a strategy in response that is equivalent to a Nash equilibrium solution in the original general-sum game. In order to avoid the problem of selecting from a set of possible Nash equilibria, at run time it is assumed that the evader will always pick an equilibrium in the zero-sum game that maximizes its deterministic distance to the team, and therefore pursuers also pick that equilibrium.

These approaches have been applied to problems of impressive size, for example, a team of three pursuers searching for an evader over a grid of 400 cells leading to a state space of size 400^4. The effects of uncertainty in the evader's position, however, is dealt with by estimating the costs for the evader, given a belief about its position held by all members of the team, and then playing a strategy that would force a rational (omniscient) evader, to play a best response that turns out to be a Nash equilibrium in the original game. This approach would not work if the evader did not have access to all information held by the pursuers or the pursuers could not share information. Additionally, the value of pursuers' actions in this game are taken to be their expected immediate value given beliefs about the evader's position. Attempts to approximate long term effects of actions are not made.

While this problem class is different from that addressed by BaGA, they are somewhat complementary. These approaches to pursuit-evasion assume that some means exists with which to fully coordinate the members of the team allowing them to be treated as a meta-agent during planning. Instead, one could imagine a two-step approach to these problems in which, periodically, the robot team makes use of its communication bandwidth to pick a high-level strategy against the evader as if they were a single agent and then, in between communication intervals, each robot uses BaGA and the reward function induced by that high-level strategy to pick individual level actions conditioned on its specific observations. For example, a high-level strategy might be to pick an area to search while individual actions deal with actual motion control: in a pursuit problem in all of the Gates Building, this high-level strategy would be to pick the A-wing for exploration and then BaGA would be used to actually coordinate the search of A-wing (as in *Realistic Gates Tag*) without further communication between team members.

Khoussainov & Kushmerick (2003) model the competition between search engines as a POSG with arbitrary payoffs and then apply reinforcement learning to find a solution. The idea in this model is that the utility for a search engine is based on the number of queries it processes and the resources used to answer them. While specializing in a topic allows an engine to process queries more effectively (and results in higher quality query results, causing users to prefer that engine for that topic); its overall performance also depends on what topics its competitors are indexing. Because an engine does not have access to the topics indexed by competitors (although observations of this information can be gained by submitting queries to opponents), there is uncertainty in the actions taken by other engines, making POSG an appropriate model. As this problem is computational intractable in general, the authors make the assumption of bounded rationality in which decision makers do not act optimally in the game-theoretic sense due to incomplete information about the environment or limited computational resources. For example, modelling other agents with a finite hierarchy of beliefs rather than an infinite hierarchy is a case of assuming bounded rationality. Under bounded rationality, agents would find a solution that is optimal with respect to the finite hierarchy but not necessarily optimal in the game-theoretic sense.

In this search-engine problem, each agent uses reinforcement learning to find a good behaviour strategy against its competitors. Learning is not performed on a model of the game but rather gradient ascent is performed on the policy space. The resulting policies can be non-deterministic and are conditioned on observations. Against competitors playing fixed policies, an agent is able to find a locally optimal strategy that takes advantage of the sub-optimal play of the opponents. In self-play there is no guarantee that gradient-based learning will converge, however, in practice relatively stable solutions are found when two instances of the algorithm learned simultaneously.

With this reinforcement-learning approach, Khoussainov & Kushmerick (2003) deal with the intractability of a POSG by converting the problem to that of learning a policy in a non-stationary environment. If competitors are playing a fixed strategy, then such an approach will converge to a locally optimal policy; however, if other agents in the environment are also adapting their policies, then convergence guarantees no longer hold. While this work is an interesting example of modelling a real-world problem as a POSG, the algorithm appears more related to work on single-agent learning in non-stationary environments than approximating POSG solutions. In contrast, the learning algorithms in POIPSGs (Peshkin et al., 2000) and multi-agent OLPOMDPs (Tao et al., 2001) deal explicitly with the problem of simultaneous policy adaption and show how their algorithms do converge in situations in which their assumptions about the game model hold true.

7.3.4 Finding Approximate Solutions to One-Step Games

In BaGA, a Bayesian-Nash equilibrium for each timestep is found using an alternating-maximization algorithm that makes use of dynamic programming. Singh et al. (2004) present an algorithm for finding approximate Bayesian-Nash equilibria in (one-step) games of incomplete information that have sparse graphical models representing the interaction between agents. In this thesis, the focus has been on tightly-coupled problems; however, for more loosely-coupled domains with discrete type sets, the Approximate-TreeBNE algorithm would be able to provide guarantees on the time complexity for finding an ϵ-Bayesian Nash equilibrium.[6] That is, not only can the algorithm trade off between computational costs and the 'goodness' of the equilibrium found, the authors can make statements about the quality of the equilibria found, which cannot be made for BaGA. The representation, however, scales exponentially with the maximum number of agents involved in any interaction. This property would make it only appropriate for POSGs that model problems with periods of loose interaction (e.g., a problem with robots that only need to worry about tight coordination while physically close to each other and are otherwise making independent

[6] An ϵ-best-response strategy for an agent is one in which it can do no better than ϵ by switching its strategy. It is used in standard definitions of approximate equilibria.

action selections). Additionally, the algorithm for finding the ϵ-Bayesian Nash equilibrium has an exponential dependence on the number of types. A strong dependence on the size of the joint-type space is also present in BaGA and is why BaGA-Cluster was developed.

Similar approaches have been applied to general-sum one-stage games of imperfect information with sparse interactions by Kearns et al. (2001). In this work, graphical models are used to represent the relationship between the reward functions of each agent and approximate solutions are found by only considering neighbouring agents in policy generation. Vickrey & Koller (2002) use hill-climbing and constraint satisfaction algorithms to find approximate equilibria in these graphical games. Scalability of such algorithms are dictated by the degree of interaction between agents and the presence of a communication channel that would allow agents with dependent actions to interact.

7.4 Run-Time Communication Policies

There are many different types of run-time communication policies that have been implemented for POSGs and its sub-classes. In this thesis, BaGA generates policies that are conditioned on sets of clustered histories and so the question of what to communicate was answered by using the identifying numbers of clusters. Differences in expected value or policy were some of the ways in which the question of when to communicate was addressed. Many of the approaches discussed in this section use similar ways in which to answer the question of when to communicate. All of the algorithms are motivated by an understanding that, in the real world, communication incurs a cost or can reveal that information to an opponent. Table 7.1 summarizes the algorithms for generating run-time communication policies that are discussed in this section.

7.4.1 POSGs with Common Payoffs

The COM-MTDP framework (Pynadath & Tambe, 2002) has been used as a way to analyze the heuristic communication policies given by theories of teamwork such as GRATE* (Jennings, 1995) and STEAM (Tambe, 1997). The resulting communication policies are applied to a problem in which agents are trying to achieve a set of goals. The authors then analyze these policies, in addition to a locally optimal communication policy, to determine under what conditions they result in optimal behaviour.

Table 7.1: Summary of algorithms for generating run-time communication policies in POSGs. Forced sync means that if one robot initiates communication, all robots must share information. While the models used for communication in COM-MTDP and the work by Goldman & Zilberstein (2003) do not require forced synchronization (and theoretically allow for optimal policies), they discuss approximation algorithms that do require synchronization.

Approach	**Model/Sub-Class**	**Forced Sync**	**Comm Type**	**Comm'd Information**	**Comm Decision**
BaGA-Comm	POSGs w/ Common Payoffs	no	*and-comm*	# of cluster to which true history belongs	Myopic expected value change, myopic policy-difference, # clusters too high
COM-MTDP (Pynadath & Tambe, 2002)	POSGs w/ Common Payoffs	no in general, yes as implemented	*and-comm*	in general, any message in Σ_i, as implemented, goal achievement	myopic expected value change
Dec-Comm (Roth et al., 2005)	POSGs w/ Common Payoffs	no	*and-comm*	history since last comm	myopic policy-difference for any agent
Nair et al. (2004)	POSGs w/ Common Payoffs	yes	*or-comm*	history since last sync	locally optimal policy i.e., comm selected if best action using DP-JESP
Xuan et al. (2001)	factored DEC-MDP	yes	*and-comm*	local state (sync on global state)	myopic expected value change, evaluated with heuristic
Xuan & Lesser (2002)	factored DEC-MDP	yes	*and-comm*	local state (sync on global state)	own action selection is ambiguous
Goldman & Zilberstein (2003)	factored DEC-POMDP	no in general, yes as implemented	*and-comm*	in general, any message in Σ_i, as implemented, local state	myopic expected value change assuming no more comm possible

In the domain examined by the authors, agents have only one possible communication message, which indicates that a certain joint goal has been achieved (i.e., the question of what to communicate has been answered as goal achievement). In GRATE*, an agent communicates whenever it believes a joint goal to have been achieved, while in STEAM an agent communicates if an estimated cost of miscoordination is greater than the cost of communication. Pynadath & Tambe present a locally optimal communication policy that outperforms those of GRATE* or STEAM. Like STEAM, agents only communicate if the costs of execution after achieving a goal outweighs the cost of communicating the fact that the goal has been achieved; however, the true expected cost of miscoordination is used rather than an estimated cost.

This locally optimal policy is similar to the run-time EVD communication decision used in BaGA-Comm. However, while agents in COM-MTDP reason about their full action and observation histories, they can only communicate one part of that history: goal achievement. If this piece of information is not sufficient for decision making, then this choice of message for COM-MTDP is not as informative as the messages used in BaGA-Comm. Furthermore, agents only make a decision about communication if they identify that a goal has been achieved. In BaGA-Comm agents are free to communicate at any point in the task. It should be noted, however, that this restriction is specific to the domain and not to COM-MTDPs in general.

The Dec-Comm algorithm (Roth et al., 2005) makes run-time *and-communication* decisions in a similar way to BaGA-Comm with the PD decision. Dec-Comm uses the Q-POMDP heuristic as a way to evaluate the effects of communication. Each robot calculates the Q-POMDP action given the current distribution over joint histories and then calculates what the Q-POMDP action would be given the distribution over joint histories induced by factoring in its true history. If those actions are different joint actions, then it broadcasts its history to its teammates (i.e., all observations made since it last communicated). The teammates integrate that information into their distribution over joint histories and each one calculates a new Q-POMDP action. They also recalculate the Q-POMDP action that would now occur if they broadcast their history. If it is different, then they do so. This recursive algorithm is bounded by the number of robots as, in the worst case, one robot decides to communicate, which causes the next robot to decide to communicate and so on. Because Dec-Comm can result in multiple instances of communication at each timestep, robots wait a fixed period of time before acting in order to make sure that no more communication acts will be received.

As stated, the Dec-Comm approach has similarities to the PD version of BaGA-Comm.

In PD, if any robot's part of $\sigma^{i,t}$ is different from that of π^t, then robot i will communicate, which is somewhat equivalent to communicating if a joint action is different. As with BaGA-Comm, if the only way in which a change in joint action would occur is through a joint communication act, then this event would not be discovered because the recursive check for communication in Dec-Comm can only be triggered if at least one robot discovers a policy difference. While BaGA-Comm does not implement a recursive communication check, it could be added (i.e., BC'^t and a $BGC'^{i,t}$ are also calculated and the resulting policies compared repeatedly until no new communication acts are triggered). Unlike BaGA-Comm, robots using Dec-Comm cannot condition their action choice on individual observations in between communication acts.

Nair et al. (2004) look at the computational savings gained by enforcing periodic synchronizations of observations in between steps of full game-theoretic reasoning about the beliefs of others. In this *or-communication* decision problem, agents communicate at a minimum of every K steps, but can also communicate more frequently if it would lead to improvement in performance. This communication algorithm approximates the optimal communication policy, given the *or-communication* constraint, for the full POSG by using DP-JESP (Nair et al., 2003). Unlike the BaGA-Comm approach, this approach would technically be able to identify cases in which only joint communication actions would yield an improvement in performance. The authors, however, use a synchronized communication action, *sync*, which means that if one robot decides to communicate, all robots will synchronize their information about the world, leading to a unified joint belief. This constraint greatly simplifies the calculation of the expected value of communicating and makes identifying these joint communication cases a moot point.

In *or-communication*, the *sync* action is considered part of the regular action space and so the only modification that must be made to DP-JESP is in how to evaluate the effects of a *sync* during dynamic programming. While performing dynamic programming, a *sync* action results in the cost of communication plus the expected value of the possible resulting synchronized belief states. If the action is not *sync* then two things can happen: with a certain probability another agent performs a *sync*, over-riding this agent's action choice (as it must also perform the *sync*) or no agent performs a *sync*. As a result, the value of this domain-level action is some combination of the expected cost of making a *sync* action (as calculated before) and the expected immediate reward plus the expected future reward of the domain-level action under consideration. In order to evaluate these quantities during DP-JESP, DP-JESP is run for any possible starting joint belief b with t steps to go. This approach allows the algorithm to slot in the appropriate value for the expected value of a synchronized belief state when calculating the value of taking *sync* at future timesteps.

DP-JESP can be used in this way to find the locally optimal joint policy without any constraints on communication or with the addition of a constraint that enforces synchronization at least every K steps. With this addition, there can be no observation histories greater than length K, which in turn limits the space requirements of the algorithm. This constraint is somewhat similar to the $|C_i^t| > k$ decision used in BaGA-Comm in which a robot communicates its cluster to the team if it has more than k clusters. Like the K-step constraint, this communication decision is designed to reduce computational requirements of the problem; however, it is not tied to a specific history length but rather to how much clustering can occur in a specific domain.

Unlike BaGA-Comm and many of the other approaches for run-time communication decisions discussed in this section, the communication decisions made by Nair et al. are not myopically greedy. They are still run-time communication decisions, however, because a robot will only communicate at a timestep if it has received specific observation histories. The authors, however, only run DP-JESP with one random starting policy and so the quality of the locally optimal equilibrium policy they find can be poor (the authors acknowledge this tradeoff between better searching of the space and achieving good timing results).

7.4.2 Factored DEC-MDPs and Factored DEC-POMDPs

Xuan et al. (2001) look at the problem of run-time communication policies in a sub-class of POSGs similar to the factored DEC-MDP model of Becker et al. (2003). In these types of problems, because robots have full observability of their local state, the communication decision is tied to deciding when global information necessary to make progress toward the goal rather than how to resolve conflicts in belief about the world or how to augment one's own belief about the global state. Specifically, they consider a grid-world domain in which two robots receive a large reward for meeting in any of the grid cells. The robots know the starting position of their teammate and so can use a fully observable version of the problem to select a goal cell that minimizes time to meeting. Robot actions, however, are noisy and so robots need to periodically adjust this goal location. Without communication, they do not know the position of their teammate and so cannot select a new goal. Communication has a cost and is assumed to be of the synchronized form. Therefore, if one robot communicates, both robots know the full global state of the world.

Because the optimal communication policy is intractable, three fixed communication heuristics were considered: never communicate, always communicate or communicate if the current plan is no longer achievable (i.e., a robot knows that it can no longer reach the goal cell at the right time). In addition, a hybrid policy was examined in which each robot constantly calculates whether or not there is a better goal state and, if so, the expected gain in reward from changing to that new goal. If the expected gain is greater than the cost of communication, then the robot communicates, making this approach similar to the BaGA-Comm EVD decision. Like BaGA-Comm, a heuristic is also used to evaluate the effects of communicating: the selection of what new goal states to evaluate and what their expected reward will be are evaluated with a heuristic rather than exactly. As expected, the hybrid approach has the best performance if communication is not free, otherwise always communicate is the most appropriate policy to use.

In Xuan & Lesser (2002) the authors take a more principled approach to finding communication policies for these problems. While a version of the factored DEC-MDP with full communication can be solved, yielding a centralized policy for the problem (the CP), at execution time the agents must convert this CP into decentralized policies, or DPs. Communication is essential in order to provide a conversion between the two that allows the DPs to perform similarly to the CP. Each robot has a copy of the CP and, at run time, it calculates whether or not its part of the joint action as given by the CP is the same for all global states consistent with its local state. If its part of the joint action is the same, then it has no ambiguity on its own action choice and does not need to communicate. If it is not, then there is ambiguity, which could be resolved by communication. This myopically greedy communication policy is somewhat similar to the PD condition for BaGA-Comm; however, robots only consider policy differences in their own action choice and not of those of their teammates because they use only their part of the joint action for comparison, and not the entire joint action. They do not consider how their local information might change the action choice of their teammate as does the PD decision in BaGA-Comm.

While one solution for converting CPs into DPs is for robots to communicate whenever the local state results in an ambiguity towards action choice, in the worst case this approach will result in communication at every timestep. Xuan & Lesser (2002) also do hill-climbing on the mapping between ambiguous states and the communication decision to reduce the amount of communication. A second way to reduce communication is to generate exact communication policies for a modified CP instead of the true CP. In a modified CP, robots effectively merge together sets of local histories for decision making. This view has some similarities to BaGA-Cluster.

Goldman & Zilberstein (2003) extend the factored Dec-MDP model to cover the sub-case of POSGs in which agents have partial observability of their state. In their 2-agent factored

DEC-POMDP-COM model, they examine the NEXP-complete problem of finding domain-level and communication policies that together optimize a global reward function. An agent must not only consider the problem of when to communicate, but is also free to select from any of the messages in a message set Σ. Agents condition action choice on histories of both observations and previously received communication messages.

The authors do present a formulation of how the optimal policy would be found; however, because it is computational intractable, they also include an approximate algorithm in which, at each timestep, agents greedily decide if they should send a message assuming that no additional communication will be possible: a single-communication policy. That is, assuming that agents can only communicate once, they greedily decide on what timestep to do so. However, because agents make this communication decision on each timestep, they can actually communicate more than once.

The single-communication policies are found by using the optimal policy for a version of the problem without communication as the expected value of what would happen after communication takes place. The optimal single-communication policy is to communicate at the earliest timestep for which the expected cost of communicating and then following the optimal policy for the no-communication case is less than that of following the no-communication case from the very beginning. This approach is similar to BaGA-Comm with the EVD decision because both approaches used a locally myopic greedy policy to select communication actions based on the difference in expected value. Furthermore, both approaches evaluate the expected value of policies using heuristics that are based on some (non-true) assumptions about future communication. While in DEC-POMDP-COM the heuristic assumes no more communication will be possible, BaGA-Comm was applied to problems in which the heuristics used for utility varied from assuming full communication (*Robotic Tag*) or no communication in the future (*simple-heuristic* for *Lady and The Tiger*).

7.5 Other Multi-Robot Frameworks

This section discusses some other important multi-robot frameworks and how they allow a robot to take the actions of teammates into account during decision making. While frameworks such as behaviour-based or market-based systems appear very different from POSGs, they are frequently modelling similar problems (albeit at a different resolution). Indeed, their solutions can be related to approximate solutions of POSGs by looking at how they handle the integration of teammates into decision making.

7.5.1 Theories of Teamwork

Distributed systems have been a subject of research outside of the areas of decision- and game-theoretic control as they provide a natural way to treat real-world systems. Usually, there is some natural structure that facilitates the deconstruction of a problem into its components, such as a power plant into various turbines and generators. It is, however, not always intuitive to specify the full set of interactions between components, especially for low-probability events. Theories of teamwork, such as shared plans (Grosz & Kraus, 1996) and joint intentions (Cohen & Levesque, 1991), were designed to help resolve brittleness in systems and their designers' inability to predict all possible agent interactions. Theories of teamwork address how agents decide upon mutual goals and implemented systems then make use of expert-like rules for deciding agent behaviour once sub-goals have been selected (e.g., GRATE* (Jennings, 1995) and STEAM (Tambe, 1997)).

A simplistic view of joint intentions is that they represent the commitment of a group of agents to the achievement of a goal. Unfortunately, if one agent believes that the goal is no longer achievable, it will selfishly abandon its progress towards the goal and not inform its team. In STEAM, mechanisms exist to ensure that if an agent believes that a joint goal is no longer achievable, it cannot abandon its intention without also informing the other agents with which it has the joint intention. This type of coordination requires a communication channel; however, Tambe (1997) extends joint intentions with decision-theoretic communication in such a way that agents only communicate information that has high utility with respect to the completion of the plan. For example, a team of helicopters might not communicate to each other the achievement of each way-point along a route to the goal because, with a high probability, the fact that a helicopter missed a way-point can be deduced from its actions. However, if one helicopter sees an enemy it would communicate that information to the team because it is a low-probability event that results in high costs.

To a certain extent these theories of coordination were designed to help cut the recursion of beliefs about what a teammate is doing: if a team has a joint goal, individual members do not have to worry about what they believe that their teammates believe (and so on) about progress towards that goal because they have a joint intention to achieve it. A teammate cannot abandon that goal without first informing the team and so agents can make assumptions about the behaviour of others while choosing their own actions. Theories of teamwork, therefore, provide a high-level way to take teammates into account during action selection.

7.5.2 Behaviour Based

ALLIANCE is a framework for fault-tolerant cooperative control of heterogeneous robot teams performing loosely-coupled sub-tasks (Parker, 1998). It is a behaviour-based architecture that uses the idea of motivations in action selection to facilitate cooperative behaviour and allows for the re-allocation of tasks in the case of robot failure. Robots are assumed to be able to detect the effects of their own actions and detect the actions of other robots (through any available means, communication or observation), although some uncertainty in this detection is permitted. While communication may be available to the team, it is not guaranteed and therefore knowledge of robot failure cannot necessarily be transmitted to the rest of the team.

Within behaviour-based frameworks, the output of lower-level behaviours, such as obstacle avoidance, can be suppressed by high-level behaviours such as exploring or map building. If there are a number of competing actions that cannot be implemented in parallel, additional structure is needed to select which action to achieve. In ALLIANCE, these actions are grouped into behaviour sets that are either active as a group or hibernating. Each of these behaviour sets correspond roughly to a sub-task and one behaviour set is selected by using motivational behaviours (a motivational behaviour tracks how much a robot wants to perform a task). While only one behaviour set can be active at a time, the robot always has a motivation for each performing each one. Robots are motivated to continue performing a task if they are making forward progress towards the associated goal but, if they judge that progress is no longer being made, their motivation starts to drop. Their motivation for performing other tasks also increases as time goes by. The motivation for performing a task increases quickly if no other robot is performing it (impatience) and more slowly if the robot believes some other robot is working towards the associated goal (acquiescence). This combination of impatience and acquiescence models allows a robot to start doing a sub-task if it feels the original robot has failed, but also drives robots to start doing different tasks.

This impatience and acquiescence within the motivation behaviours provide a rough way for robots to model their teammates during action selection. It is not an explicit model and it relies on observations of teammates to determine what progress is being made rather than an explicit calculation of what actions or behaviour-sets a teammate will select. While coordination could be thought of as an emergent property of this framework, a robot does take teammates into account during its decision-making process by allowing their actions to influence its own motivations. This property makes ALLIANCE an interesting framework because it has attempted to capture, through heuristics, some of the intuition behind decentralized decision making that is formalized in POSGs. While the optimality of the resulting policies cannot be evaluated, ALLIANCE performs well in practice with the quality of solutions being positively affected by how aware robots are of the actions of their teammates. These experimental results validate the use of more complex models that take into account the interdependence of action selection through either explicit models of teammates (RMM) or implicit models (POSGs): the more agents coordinate their action selection, the higher the quality of the resulting policies.

Move Value Estimation for Robot Teams (MVERT) is a behaviour-based framework in which teammate modelling becomes more explicit because it is designed to be applied to tightly-coupled tasks (Stroupe, 2003). In the problems addressed by MVERT, robots need to coordinate on the selection of movement actions but trade off optimality for computational efficiency. Actions in MVERT are defined as selecting a pose to move to in the next timestep. Mathematical value functions that map a state (pose of all robots in the team and location of any objects in the environment) and potential actions to numerical values representing progress towards a different tasks are defined. There are no restrictions placed on these functions, or the tasks they represent, so long as they are a computable mathematical function. Using the current world state, which can include uncertainty, and models of teammates' capabilities, a robot approximates what its teammates' contributions will be to the value functions on the next timestep. Given these estimated contributions, a robot then selects an action for itself that maximizes the value functions. It is assumed that a robot can obtain information about the pose of teammates through communication or through observation and inference. Therefore, while there can be uncertainty over the current world state, robots are assumed to have access to the same information.

While MVERT is a behaviour-based approach, the problem class it deals with is somewhat analogous to POSGs with common payoffs. It is not quite this full problem class, as there is the further assumption that robots hold the same beliefs about world state, either through the communication of their own local state (and any associated uncertainty) or through observations of each other. While one way to satisfy this assumption is through the use of communication, MVERT does not also use communication to help resolve action choice or improve the estimates of actions taken by others.[7]

MVERT also has some analogies to approximation methods for POSGs that include explicit teammate modelling. Robots myopically select their next best action while reasoning

[7] See the discussion of passive coordination in Hoplites (Kalra et al., 2005) in Section 7.5.3 for an example of how communication can be used to improve the estimate of the action choice of others.

about the action selection of their teammates. However, unlike other approaches to teammate modelling, robots in MVERT use only a first-order model: teammates are modeled as selecting actions based only on their beliefs about the world. While this type of decentralized problem solving does require the use of a truncated belief hierarchy (because teammates are explicitly modeled), work done by Noh & Gmytrasiewicz (2005) on the tradeoff between solution quality and model-depth suggests that a first-order model is only very approximate.

Behaviour-based methods are attractive for controlling robot teams because of their speed. However, it is important to understand what parts of the method are approximate because of the use of behaviours and what parts are approximate because of how they deal with decentralized decision making. By relating MVERT to solution algorithms for POSGs that also use finite-order teammate models, insight is gained into what types of approximations are being made in the decentralized decision making. The problems examined by Stroupe (2003), however, are much larger than those examined in the POSG literature and are, therefore, a testament to the computational tractability of MVERT.

7.5.3 Market Based

While market-based, or auction, approaches are usually associated with loosely-coupled robot systems (e.g., task allocation rather than step-by-step coordination), there has been work done on applying them to tightly-coupled problems. Unlike the work with POSG-related models and their approximations, market-based approaches require a communication system to be available for the auction process itself. A centralized auctioneer is not always required (Dias & Stentz, 2003; Gerkey & Matarić, 2001); however, robots must be able to communicate in order to negotiate task allocation. In the TraderBots system (Dias & Stentz, 2003), robots bid on sub-tasks to execute, and receive rewards for successful task execution and penalties for consumed resources. As with non-cooperative game theory, the robots are self-interested in maximizing their expected payoffs; however, the price of resources and tasks can be hand-crafted to help ensure robots allocate sub-tasks in a way that also helps maximize the profit of the team as a whole. An individual robot can also try to create a better joint plan for the entire team (ideally an optimal plan), or subset of robots, and then sell that plan on the market. This approach, however, is fundamentally meant for loosely-coupled systems. The Hoplites framework is designed to extend frameworks like TraderBots to tightly-coupled tasks (Kalra et al., 2005).

In tightly-coupled tasks not only do the robots need to have plans that condition action selection on the actions of others (tight coordination), but they also require a solution to the credit-assignment problem of how the rewards and penalties associated with a task should be distributed amongst the robots performing the task together. In Hoplites, two types of coordination, passive and active, are added to the usual market framework. In passive coordination, rather than select the most profitable tasks, robots select the most profitable plans (i.e., policies). The value of possible plans are estimated using the actions of others and the environment, and the most profitable plan transmitted to the rest of the team. Using this information, each robot can revise its estimated profits (e.g., use the newest set of plans to evaluate the actions of others) and send out its new best plan. This type of coordination iteratively incorporates the most recent information in teammates' plans and as such has similarities to the alternating-maximization algorithm. Because it is done in a distributed fashion, however, a fixed point may not be found as multiple robots can change their plan at once. For example, if the best plan has two robots performing the same action but the currently proposed plans has one robot performing A and the other B, incorporating this information could lead to a new set of proposed plans in which the first robot now performs B while the second A. This type of coordination is therefore only useful in cases where it is easy to accurately 'guess' the actions of others correctly during the first set of plan evaluations or else when the actions of others have relatively little impact on the robot's performance.

In environments where tight coordination is required, active coordination leads to better performance. If a robot's best individual plan is not very profitable, it develops a team plan that specifies actions for each member of the team that would maximize its own profit. The robot then receives price quotes from teammates that reflect the profit they would require before they would abandon their own plans to adopt this team plan. If the sum of these payments is less than the expected profit to the planning robot, it bids for its teammates participation. The search for this team plan involves a search over joint-action space but Kalra et al. (2005) do not specify a specific algorithm for Hoplites to use. Rather, they suggest the use of an algorithm that sacrifices optimality for speed.

The Hoplites approach can be thought of as similar to modeling the problem as a factored MDP when actions are loosely coupled but then moving to a more accurate framework such as POSGs with common payoffs when tight coordination is known to be required (i.e., trade off representational power with computational speed if it is known to be less important). Market-based techniques are then used to approximate the solution rather than one of the algorithms of Section 7.3. This relationship between Hoplites and different POSG sub-classes allows one to identify what types of approximations are being made by

Hoplites and under what circumstances the plans would be (near) optimal. For example, in problems that are essentially factored DEC-MDPs, passive coordination could lead to locally optimal joint plans; however, if the actions are more tightly coordinated, it could fail to find a fixed point. This insight into the relationship between the frameworks can also help identify what algorithms could be useful for finding team plans.

7.6 Summary

This chapter has covered a wide variety of frameworks for decentralized multi-robot control and their relationship to both POSGs and BaGA. The majority of these frameworks have been decision or game theoretic and, while originally formulated with respect to their relationship to MDPs and POMDPs, they can all be viewed as sub-classes of POSGs. DEC-POMDPs (Bernstein et al., 2002) and COM-MTDPs (Pynadath & Tambe, 2002) are particularly relevant because these frameworks are equivalent to the class of POSGs with common payoffs.

For POSG-based algorithms, BaGA represents a good compromise between the size of problem that can be solved and solution accuracy. BaGA can be used to solve much bigger problems than game-theoretic algorithms with similar or better accuracy such as Q-POMDP (Roth et al., 2005), alternating maximization, DP-JESP (Nair et al., 2003) and exact dynamic programming (Hansen et al., 2004). Furthermore, the real-time robot controllers generated by BaGA in Chapter 6 are for problem domains equivalent in size to those addressed by Chades et al. (2002) and Noh & Gmytrasiewicz (2005), but with fewer restrictions or approximations required to generate policies.

In addition to these POSG-related models of multi-robot control, theories of teamwork, behaviour-based approaches and market-based approaches were also discussed and analyzed with respect to how robots take their teammates into account during decision making. In general, these other multi-robot frameworks are able to handle much larger problems than those looked at in the POSG literature. However, there are similarities in how teammates are treated in frameworks like MVERT (Stroupe, 2003) and Hoplites (Kalra et al., 2005), and in how they are treated in POSG-based algorithms. This relationship between the different multi-robot frameworks could be used to gain insight into applying game-theoretic control to larger and more realistic problems.

Chapter 8

Conclusions

This dissertation motivates the use of POSGs for modelling robot teams and, in order to deal with the computational intractability of such models, presents a set of algorithms for finding approximate solutions. The resulting policies allow robots to coordinate their action selection for tightly-coupled tasks in partially observable Markovian environments under limited communication.

8.1 Modelling Robot Problems as POSGs

In tightly-coupled robot problems, robot policies need to take into account what all robots are doing because the reward function and state transitions are dependent on joint actions. If a communication infrastructure exists, then centralized approaches such as MDPs or POMDPs could be used to model these problems. If a problem is fully observable by each robot, or each robot has identical observations, then a centralized version of the problem can be solved off-line and executed in a decentralized fashion without communication. However, if robots do not receive identical observations and cannot share them through communication, there is no easy way to reduce these problems to a known class of problems such as POMDPs or MDPs.

It is obvious that, even with limited communication, robots still need to reason about what actions will be taken by their teammates when selecting actions, because their payoffs will depend on them. However, in order to properly capture the fact that teammates are also taking the robot's action selection into account when selecting actions, an infinite belief hierarchy becomes necessary to capture what each robot believes about the other robots and vice versa. POSGs use game theory to implicitly represent this belief hierarchy about action selection: the solution to a POSG is a set of policies for each robot that simultaneously optimizes action selection for each robot with respect to all other robots. Unlike models that explicitly include models of other robots in the state space of the problem, POSGs do not require approximate solutions that truncate the belief hierarchy at some finite level. They are, however, computationally intractable as shown by the NEXP-completeness result for finite-horizon POSGs with common payoffs (Bernstein et al., 2002).

From a theoretical standpoint, POSGs provide a powerful framework for modelling tightly-coordinated robot-team problems in partially observable Markovian environments.[1] Of course, their practical applicability to such problems is limited as real robot problems have continuous state and action spaces and lengthy time horizons. Even if the assumption of discrete state and action spaces is allowed, as in Chapter 6, these more realistic problems are still relatively small.

POSGs do provide a way to evaluate the performance of computationally efficient multi-robot heuristic approaches such as those like MVERT or Hoplites. Generally, such algorithms suffer from the problem that only experimental results are used to validate performance because an optimal solution is unknown or (correctly) considered too difficult to compute. However, for smaller problems, POSGs can be used to calculate, off-line, an upper bound on performance for these problems. Even for larger domains, by understanding what sub-class of POSG the algorithm is modelling and any structure that can be exploited, it may be possible to evaluate the algorithm's solution quality and computational trade-offs in a more principled way than with just experimental validation alone.[2] For example, by realizing that in MVERT a first order model of teammates is used in action selection, understanding is gained into how this framework relates to more formal POSG models.

It is actually anticipated that for many robot problems, heuristic approaches do a good job of achieving high performance quality and that in the future this claim will be validated

[1] POSGs are not limited to modelling only tightly-coupled coordination problems; however, the associated computational overhead is not necessarily appropriate for loosely-coupled problems. For example, a DEC-MDP in which the reward function includes no joint-robot structure (global reward is the sum of individual rewards), could be modelled as a set of totally independent MDPs with no loss in solution quality. If the only interaction comes from task allocation then a single-stage game could be used to model this part of the problem and independent MDP or POMDPs used to handle task achievement.

[2] Gerkey & Matarić (2003) present such a principled analysis of approaches to the multi-robot task allocation problem. By comparing various algorithms to sub-classes of the optimal assignment problem from Operations Research, a better understanding of their solution quality and computational tradeoffs can be made than through domain specific experimental validation. Pynadath & Tambe (2002) use COM-MTDP to perform a similar function for models of teamwork: they characterize several of these frameworks within the larger context of COM-MTDPs in order to compare their computational costs and performance.

by comparing such heuristics with the output of a POSG solver. It is also anticipated that for those parts of robot problems that do require tight coordination, POSG-based solutions will give insight into appropriate heuristics. For example, BaGA-Comm generated a run-time communication decision for the *Robotic Tag* problem that had a robot communicate if it was in the target cell for opponent tagging but did not see the opponent. Because it is a low-probability event based on the possible robot histories, this observation inspired a communication policy for the MLS and Q_{MDP} heuristic that had robots communicate whenever something unexpected happened. The resulting policy improved performance over a fixed communication policy for these heuristics.

8.2 BaGA Algorithm and Extensions

The primary algorithmic contribution of this thesis is BaGA, which deals with the intractability of POSGs by transforming them into a series of smaller Bayesian games. Each of these games are then solved to find one-step policies that, together, approximate the globally optimal solution of the original POSG. The Bayesian games are kept efficient to solve through pruning of low-probability histories and the use of heuristics to calculate the utility of actions. This process results in policies for the POSG that are locally optimal with respect to the heuristic used.

While pruning of low-probability histories is a simple way to keep the size of each Bayesian game manageable, clustering is an effective way to reduce the number of maintained histories in a principled way. BaGA-Cluster, which includes the addition of clustering to the basic BaGA algorithm, results in faster computation, which helps make the algorithm more appropriate as a controller for real-world problems. While low-probability clustering is a computationally faster operation, minimum-distance clustering is able to find a more natural set of clusters that best represents the overall history space of each robot.

Another effective way to reduce the number of histories that must be tracked is to permit communication between robots on the team. If done in a sensible way, robots only need to communicate infrequently and therefore will not exceed bandwidth limitations. Using BaGA-Comm for generating run-time communication policies allows one to answer the question of what to communicate because the BaGA-Cluster algorithm creates clusters of histories that are grouped together according to their effects of decision making. By transmitting the identifying numbers of clusters rather than specific histories, robots share only the relevant portions of these histories. This framework also allows communication decisions to be made that reduce the computational burdens of planning. The overall result is communication policies that lead to similar performance as full communication, but at a fraction of the number of communication acts.

This game-theoretic approach to controlling robot teams is not just limited to simulated problems. While POSGs and BaGA are based on relatively abstract notions of control and optimality, real-time robot controllers can be generated for physical robot teams. In these problems, BaGA is applied to a discretized version of the problem (that still captures relevant noise in physical robot problems) and then low-level controllers are used to map back and forth between the discrete and continuous worlds. By combining clustering and communication with game-theoretic principles, BaGA is able to meet the requirements for real-time robust control of a team of robots.

8.2.1 Limitations

BaGA uses three types of approximation in order to solve POSGs with common payoffs. It uses heuristics to evaluate the expected future value of actions and it uses alternating maximization to find a locally optimal policy with respect to the heuristic. Although an exhaustive search would be required to guarantee that the policies found with alternating maximization are globally optimal, random-restarts are used to move the search out of local maxima. Finally pruning and/or clustering is performed to keep the number of histories maintained by BaGA low. Useful performance bounds on the effects introduced by these approximations cannot be found.

BaGA-Comm generates communication policies that are locally optimal with respect to a robot and myopic due to the one-step lookahead approximation. Currently robots do not evaluate the effect of joint communication actions and, if that is the only way to improve expected reward or effect policy changes, then communication will not happen. The resulting communication policies, however, do show a large improvement in performance over policies without communication, but with a fraction of the number of communication acts required by a fixed communication policy.

Finally, while BaGA has been applied to problems substantially larger than similar approaches, these problems are still much smaller than the realistic multi-robot problems that are solvable with behaviour-based or market-based approaches. To date, robot problems with fine-grained action and observation space are handled through state, action and observation abstraction, with local controllers then used for finer motor control. Clustering and communication are also used to further minimize computational overhead. However, if a real problem includes only somewhat limited communication (e.g., communication is periodic with high frequency or constant but with latency), then BaGA can still be used to improve action selection over more selfish approaches. This more regular communication would allow BaGA to only reason about relatively short observation histories and therefore be applicable even in larger problems.

8.3 Contributions

The BaGA algorithm presents a computationally tractable way to find approximate solutions to POSGs. It is able to solve common problems in the literature, such as the *Lady and the Tiger* (2 world states, 9 joint actions, 4 joint observations), for longer planning horizons, and has been applied, without communication, to versions of robotic tag with 97200 world states, 25 joint actions and 81 joint observations. While there are some exceptions (e.g., the work on pursuit-evasion problems (Hespanha et al., 1999, 2000; Kim et al., 2001; Prandini et al., 2001) although uncertainty in world state is handled quite differently, and work with poker (Shi & Littman, 2001)), these problems are substantially bigger than those tackled by other, similar, frameworks.

The BaGA algorithms have been applied, not just in simulation, but on physical robots in order to show that real-time robot control can be accomplished with what is a relatively abstract notion of optimal control. Furthermore, BaGA can handle issues that come up with real-world problems by including models of partial or total robot malfunctions (as discussed in Section 6.4) within the problem definition.

In BaGA, the quality of the resulting solution is tied to the quality of the heuristic used for estimating the future value of actions. In this thesis, relatively simplistic heuristics were used; however, more accurate estimates could be made. Improving heuristics requires either a more involved off-line computation (e.g., finding an approximate POMDP solution to the *Robotic Tag* problem), or more on-line work by increasing the one-step lookahead to a multi-step lookahead. This framework gives BaGA a certain flexibility and provides an avenue for future research into analyzing the tradeoff between computation and solution quality.

The set of all possible observation histories for each robot grows in size exponentially with time. Clearly representing all possible histories quickly becomes impossible for modern computers. BaGA-Cluster is able to provide an automated way of grouping together histories that have the same (or similar) reward profiles and therefore lead to the same decisions in action choice. This approach will retain outliers in reward-profile space, even if they have low probability, and therefore better represent the space of possible histories than methods that are tied to probability alone.

POSGs are a model of coordination that deserves consideration, not just in abstract game-theory domains, but also in real-world robot problems. While the argument can be made that their computational intractability renders them inappropriate for robot control, POSG approximations such as BaGA show how to reduce computational costs in a principled way. Furthermore, success in applying these abstract models to concrete examples shows that this notion of optimal control for decentralized robot teams is not infeasible and that one day it will be possible to properly evaluate the tradeoff between computational costs and performance quality offered by heuristic approaches in such domains. After all, it is only by understanding optimal behaviour that a system designer is able to gain insight into when it is possible to take computational shortcuts.

Bibliography

Aumann, R. (1974). Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, *1*, 67–96.

Barbuceanu, M. & Fox, M. S. (1995). Cool: A language for describing coordination in multiagent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, (pp. 17–24). San Francisco, CA, USA. AAAI Press.

Basar, T. & Olsder, G. J. (1982). *Dynamic Noncooperative Game Theory* (First ed.). New York: Academic Press.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2003). Transition-independent decentralized Markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (pp. 41–48). New York, NY, USA. ACM Press.

Bernhard, P., Colomb, A.-L., & Papavassilopoulos, G. P. (1987). Rabbit and hunter game: Two discrete stochastic formulations. *Computers & Mathematics With Applications*, *13*(1-3), 205–225.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840.

Bernstein, D. S., Hansen, E. A., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference Artificial Intelligence (IJCAI)*, (pp. 1287–1292). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Bertsekas, D. P. & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, (pp. 195–210). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference Artificial Intelligence (IJCAI)*, (pp. 478–485). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Browning, B., Kaminka, G., & Veloso, M. (2002). Principled monitoring of distributed agents for detection of coordination failure. In *Proceedings of DARS-2002, the Sixth International Symposium on Distributed Autonomous Robotic Systems*. Fukuoka, Japan.

Brusey, J., Makies, M., Padgham, L., Woodvine, B., & Fantone, K. (2001). RMIT United. In P. Stone, T. Balch, & G. Kraetzschmar (Eds.), *RoboCup 2000: Robot Soccer World Cup VI*, volume 2019 of *Lecture Notes in Computer Science* (pp. 563–566). Berlin: Springer.

Cassandra, A. R. (1998). *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University,Department of Computer Science, Providence, RI.

Cassandra, A. R. (1999). Tony's pomdp-solve page. Software available for download at http://www.pomdp.org/pomdp/code/index.shtml.

Chades, I., Scherrer, B., & Charpillet, F. (2002). A heuristic approach for solving decentralized-POMDP: Assessment on the pursuit problem. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, (pp. 57–62). New York, NY, USA. ACM Press.

Chu, F. & Halpern, J. Y. (2001). On the NP-completeness of finding an optimal strategy in games with common payoffs. *International Journal of Game Theory,*, *30*(1), 99–106.

Claus, C. & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, (pp. 746–752). Menlo Park, CA, USA. American Association for Artificial Intelligence.

Cohen, P. & Levesque, H. (1991). Teamwork. *Nous*, *25*(4), 487–512.

Dean, T. & Givan, R. (1997). Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, (pp. 106–111). Menlo Park, CA, USA. American Association for Artificial Intelligence.

Dias, M. B. & Stentz, A. (2003). Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination. Technical Report CMU-RI -TR-03-19, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Dias, M. B., Zinck, M. B., Zlot, R. M., & Stentz, A. (2004). Robust multirobot coordination in dynamic environments. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 3435–3442). Piscataway, NJ, USA. IEEE Press.

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

Emery, R., Balch, T., Shern, R., Sikorski, K., & Stroupe, A. (2001). CMU Hammerheads team description. In P. Stone, T. Balch, & G. Kraetzschmar (Eds.), *RoboCup 2000: Robot Soccer World Cup VI*, volume 2019 of *Lecture Notes in Computer Science* (pp. 575–578). Berlin: Springer.

Emery, R., Balch, T., & Sikorski, K. (2002). Protocols for collaboration, coordination and dynamic role assignment in a robot team. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 3008–3015). Piscataway, NJ, USA. IEEE Press.

Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2004). Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (pp. 136–143). Washington, DC, USA. IEEE Computer Society.

Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2005). Game theoretic control for robot teams. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 1175–1181). Piscataway, NJ, USA. IEEE Press.

Everitt, B. S., Landau, S., & Leese, M. (2001). *Cluster Analysis* (Fourth ed.), Chapter 4. London: Arnold Publishers.

Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1999). Common knowledge revisited. *Annals of Pure and Applied Logic*, *96*, 89–105.

Fershtman, C. & Pakes, A. (2004). Finite state dynamic games with asymmetric information: A computational framework. Harvard Institute of Economic Research Discussion Paper No. 2041.

Fox, D., Burgard, W., & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, *11*, 391–427.

Fudenberg, D. & Tirole, J. (1991). *Game Theory*. Cambridge, MA: The MIT Press.

Gerkey, B. P. & Matarić, M. J. (2001). Principled communication for dynamic multi-robot task allocation. In D. Rus & S. Singh (Eds.), *Experimental Robotics VII: Proceedings of the International Symposium on Experimental Robots (ISER 2000)*, volume 271 of *Lecture Notes in Control and Information Sciences* (pp. 353–362). Berlin: Springer-Verlag.

Gerkey, B. P. & Matarić, M. J. (2002). Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, *18*(5), 758–786.

Gerkey, B. P. & Matarić, M. J. (2003). Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 3862–3868). Piscataway, NJ, USA. IEEE Press.

Gmytrasiewicz, P. J. & Doshi, P. (2004). A framework for sequential planning in multi-agent settings. In *Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics (AI&M)*. Fort Lauderdale, FL.

Gmytrasiewicz, P. J. & Durfee, E. H. (1995). A rigorous, operational formalization of recursive modeling. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, (pp. 125–132). Menlo Park, CA, USA. AAAI/MIT Press.

Goldman, C. V. & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (pp. 137–144). New York, NY, USA. ACM Press.

Greenwald, A. & Hall, K. (2003). Correlated Q-learning. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, (pp. 242–249). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Grosz, B. J. & Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, *86*(2), 269–357.

Guestrin, C. & Gordon, G. (2002). Distributed planning in hierarchical factored MDPs,. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, (pp. 197–206). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Guestrin, C., Koller, D., & Parr, R. (2001). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems NIPS-14*, (pp. 1523–1530). Cambridge, MA, USA. MIT Press.

Guestrin, C., Lagoudakis, M. G., & Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML)*, (pp. 227–234). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Gutmann, J.-S., Hatzack, W., Herrmann, I., Nebel, B., Rittinger, F., Topor, A., Weigel, T., & Welsch, B. (1999). The CS Freiburg robotic soccer team: Reliable self localization, multirobot senser integration and basic soccer skills. In M. Asada & H. Kitano (Eds.), *RoboCup-1998: Robot Soccer World Cup II*, volume 1604 of *Lecture Notes in Computer Science* (pp. 93–108). Berlin: Springer.

Hansen, E., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*, (pp. 709–715). Menlo Park, CA, USA. AAAI Press/MIT Press.

Harsanyi, J. C. (1967-1968). Games with incomplete information played by 'Bayesian' players, parts I, II and III. *Management Science*, *14*(3,5 and 7), 159–182, 320–334 and 486–502.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *13*, 33–94.

Hespanha, J. P., Kim, H. J., & Sastry, S. (1999). Multiple-agent probabilistic pursuit-evasion games. In *Proceedings of the 38th IEEE International Conference on Decision and Control*, (pp. 2432–2437). Piscataway, NJ, USA. IEEE Press.

Hespanha, J. P., Prandini, M., & Sastry, S. (2000). Probabilistic pursuit-evasion games: A one-step Nash approach. In *Proceedings of the 39th IEEE International Conference on Decision and Control*, (pp. 2272–2277). Piscataway, NJ, USA. IEEE Press.

Hu, J. & Wellman, M. P. (2003). Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, *4*(Nov), 1039–1069.

Jennings, N. R. (1995). Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, *75*(2), 195–240.

Jung, H., Nair, R., Tambe, M., & Marsella, S. (2002). Computational models for multiagent coordination analysis: Extending distributed POMDP models. In *Formal Approaches to Agent-Based Systems*, (pp. 103–114). Norwell, MA. Kluwer Academic Publishers.

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

Kalra, N., Ferguson, D., & Stentz, A. (2005). Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 1182–1189). Piscataway, NJ, USA. IEEE Press.

Kaminka, G. & Tambe, M. (2000). Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, *12*, 105–147.

Kearns, M. J., Littman, M. L., & Singh, S. P. (2001). Graphical models for game theory. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI)*, (pp. 253–260). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Khoo, A. & Horswill, I. D. (2002). An efficient coordination architecture for autonomous robot teams. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 287–292). Piscataway, NJ, USA. IEEE Press.

Khoussainov, R. & Kushmerick, N. (2003). Automated index management for distributed Web search. In *Proceedings of the Twelfth ACM International Conference on Information and Knowledge Management*, (pp. 386–393). New York, NY, USA. ACM Press.

Kim, H. J., Vidal, R., Shim, D. H., Shakernia, O., & Sastry, S. (2001). A hierarchical approach to probabilistic pursuit-evasion games with unmanned ground and aerial vehicles. In *Proceedings of the 40th IEEE International Conference on Decision and Control*, (pp. 634–639). Piscataway, NJ, USA. IEEE Press.

Kok, J. R., Spaan, M. T. J., & Vlassis, N. (2005). Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, *50*(2-3), 99–114.

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

Littman, M. L. (2001). Friend-or-Foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, (pp. 322–328). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Littman, M. L., Cassandra, A. R., & Pack Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML)*, (pp. 362–370). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Matarić, M. J. (1995). Designing and understanding adaptive group behavior. *Adaptive Behavior*, *4*(1), 51–80.

McKelvey, R. & McLennan, A. (1996). Computation of equilibria in finite games. In H. Amman, P. A. Kendrick, & J. Rust (Eds.), *Handbook of Computational Economics*, volume 1 (pp. 87–142). B. V., Amsterdam: Elsevier Science.

Montemerlo, M., Roy, N., & Thrun, S. (2002). Carnegie Mellon robot navigation toolkit. Software package for download at http://www.cs.cmu.edu/˜carmen.

Nair, R., Roth, M., Yokoo, M., & Tambe, M. (2004). Communication for improving policy computation in distributed POMDPs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (pp. 1098–1105). Washington, DC, USA. IEEE Computer Society.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference Artificial Intelligence (IJCAI)*, (pp. 705–711). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, *36*(1), 48–49.

Nettleton, E., Thrun, S., Durrant-Whyte, H., & Sukkarieh, S. (2003). Decentralized SLAM with low-bandwidth communication for teams of airbourne vehicles. In *Proceedings of the International Conference on Field and Service Robotics (FSR 2003)*. Lake Yamanaka, Japan.

Noh, S. & Gmytrasiewicz, P. (2005). Flexible multi-agent decision-making under time pressure. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*. To Appear.

Olsder, G. J. & Papavassilopoulos, G. P. (1988). When to use a searchlight. *Journal of Mathematical Analysis and Applications*, *136*, 466–478.

Ooi, J. M. & Wornell, G. W. (1996). Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proceedings of the 35th IEEE International Conference on Decision and Control*, (pp. 293–298). Piscataway, NJ, USA. IEEE Press.

Owen, G. (1968). *Game Theory* (First ed.). Philadelphia, PA: W.B. Saunders Company.

Papadimitriou, C. H. & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Parker, L. E. (1998). ALLIANCE: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, *14*(2), 220–240.

Peshkin, L., Kim, K.-E., Meuleau, N., & Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, (pp. 489–496). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Pineau, J., Gordon, G., & Thrun, S. (2003a). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference Artificial Intelligence (IJCAI)*, (pp. 1025–1032). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Pineau, J., Gordon, G., & Thrun, S. (2003b). Policy-contingent abstraction for robust robot control. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, (pp. 477–484). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Prandini, M., Hespanha, J. P., & Pappas, G. J. (2001). Greedy control for hybrid pursuit games. In *Proceedings of the 2001 European Control Conference*. Porto, Portugal.

Pynadath, D. V. & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, *16*, 389–423.

Rathnasabapathy, B. & Gmytrasiewicz, P. (2003). Formalizing multi-agent POMDP's in the context of networking routing. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*. Big Island, HI, USA.

Rosencrantz, M., Gordon, G., & Thrun, S. (2003). Decentralized sensor fusion with distributed particle filters. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, (pp. 493–500). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Roth, M., Simmons, R., & Veloso, M. (2005). Decentralized communication strategies for coordinated multi-agent policies. In A. Shultz, L. E. Parker, & F. Schneider (Eds.), *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings of the 2005 International Workshop on Multi-Robot Systems*, volume IV (pp. 93–106). Norwell,MA: Kluwer Academic Publishers.

Russell, S. & Norvig, P. (2002). Section 6.5: Games that include an element of chance. In *Artificial Intelligence: A Modern Approach* (Second ed.). Upper Saddle River, NJ: Prentice Hall.

Sakovics, J. (2001). Games of incomplete information without common knowledge priors. *Theory and Decision*, *50*(4), 347–366.

Samet, D. (1997). Iterated expectations and common priors. *Games and Economic Behavior*, *24*(1-2), 131–141.

Selten, R. (1975). Re-examination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, *4*, 22–55.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, *39*(10), 1095–1100.

Shi, J. & Littman, M. L. (2001). Abstraction methods for game theoretic poker. In T. Marsland & I. Frank (Eds.), *Computers and Games*, volume 2063 of *Lecture Notes in Computer Science* (pp. 333–345). London: Springer-Verlag.

Singh, S., Soni, V., & Wellman, M. (2004). Computing approximate Bayes Nash equilibria in tree-games of incomplete information. In *Proceedings of the Fifth ACM Conference on Electronic Commerce (EC'04)*, (pp. 81–90). New York, NY, USA. ACM Press.

Stone, P. & Veloso, M. (1998). Communication in domains with unreliable, single-channel, low-bandwidth communication. In A. Drogoul, M. Tambe, & T. Fukuda (Eds.), *Proceedings of the First International Workshop on Collective Robotics*, volume 1456 of *Lecture Notes in Computer Science* (pp. 85–97). London: Springer-Verlag.

Stone, P. & Veloso, M. (1999). Task decomposition, dynamic role assignment and low bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, *110*(2), 241–273.

Stone, P. & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, *8*(3), 345–383.

Stroupe, A. (2003). *Collaborative Execution of Exploration and Tracking Using Move Value Estimation for Robot Teams (MVERT)*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, *7*, 83–124.

Tao, N., Baxter, J., & Weaver, L. (2001). A multi-agent, policy-gradient approach to network routing. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, (pp. 553–560). San Francisco, CA, USA. Morgan Kaufmann Publishers, Inc.

Vickrey, D. & Koller, D. (2002). Multi-agent algorithms for solving graphical games. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, (pp. 345–351). Menlo Park, CA, USA. American Association for Artificial Intelligence.

Vidal, J. M. & Durfee, E. H. (1997). Agents learning about agents: A framework and analysis. In *AAAI-97 Workshop on Multiagent Learning*. Providence, RI, USA.

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

von Stengel, B. & Koller, D. (1997). Team-maxmin equilibria. *Games and Economic Behavior*, *21*(1-2), 309–321.

Wang, X. & Sandholm, T. (2002). Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Advances in Neural Information Processing Systems NIPS-15*, (pp. 1571–1578). Cambridge, MA, USA. MIT Press.

Weiß, G. (Ed.). (1997). *Distributed Artificial Intelligence Meets Machine Learning*, volume 1221 of *Lecture Notes in Artificial Intelligence*. Berlin: Springer-Verlag.

Weiß, G. & Sen, S. (Eds.). (1996). *Adaption and Learning in Multi-Agent Systems*, volume 1042 of *Lecture Notes in Artificial Intelligence*. Berlin: Springer-Verlag.

Wellman, M. P. & Wurman, P. R. (1998). Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, *24*(3-4), 115–125.

Xuan, P. & Lesser, V. (2002). Multi-agent policies: From centralized ones to decentralized ones. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (pp. 1098–1105). New York, NY, USA. ACM Press.

Xuan, P., Lesser, V., & Zilberstein, S. (2001). Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents)*, (pp. 616–623). New York, NY, USA. ACM Press.

Index