EXPLOITING STRUCTURE IN DECENTRALIZED MARKOV DECISION PROCESSES

A Dissertation Presented

by

RAPHEN BECKER

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2006

Computer Science

UMI Number: 3242298

UMI Microform 3242298

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

EXPLOITING STRUCTURE IN DECENTRALIZED MARKOV DECISION PROCESSES

A Dissertation Presented

by

RAPHEN BECKER

Approved as to style and content by:

Victor Lesser, Co-chair

Shlomo Zilberstein, Co-chair

Sridhar Mahadevan, Member

Abhi Deshmukh, Member

Bruce Croft, Department Chair
Computer Science

ACKNOWLEDGMENTS

I would like to thank my advisors and mentors Victor Lesser and Shlomo Zilberstein for their support and insight. Their guidance in both my research and my life has been and continues to be invaluable to me. I would also like to express my appreciation for my other committee members, Sridhar Mahadevan and Abhi Deshmukh. Their comments and questions helped to clarify and polish this thesis.

Thanks also goes to my friends and colleagues in the Multi-Agent Systems lab and the Resource Bounded Reasoning lab, both past and present. You have provided me with many hours of valuable discussions on my research, your research, other research, and non-research. You have made my time here both exciting and fun.

I would also like to thank all of my friends who came to the MAS lab game nights, as well as the other games I played in, and especially Joel and Monica Sieh. You guys provided me with a welcome distraction from work and a much needed bonus on my sanity rolls.

And finally, a special thanks goes to my loving wife, Jiaying Shen, whose love and encouragement helped me through the frustrating times. I can always count on you to put me back on the right track when I am confused.

ABSTRACT

EXPLOITING STRUCTURE IN DECENTRALIZED MARKOV DECISION PROCESSES

SEPTEMBER 2006

RAPHEN BECKER

B.A., GRINNELL COLLEGE

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Victor Lesser and Professor Shlomo Zilberstein

While formal, decision-theoretic models such as the Markov Decision Process (MDP) have greatly advanced the field of single-agent control, application of similar ideas to multi-agent domains has proven problematic. The advantages of such an approach over traditional heuristic and experimental models of multi-agent systems include a more accurate representation of the underlying problem, a more easily defined notion of optimality and the potential for significantly better solutions. The difficulty often comes from the tradeoff between the expressiveness of the model and the complexity of finding an optimal solution. Much of the research in this area has focused on the extremes of this tradeoff. At one extreme are models where each agent has a global view of the world, and solving these problems is no harder than solving single-agent problems. At the other extreme lie very general, decentralized models, which are also nearly impossible to solve optimally.

The work proposed here explores the middle-ground by starting with a general decentralized Markov decision process and introducing structure that can be exploited to reduce the complexity. I present two decision-theoretic models that structure the interactions between agents in two different ways. In the first model the agents are independent except for an extra reward signal that depends on each of the agents' histories. In the second model the agents have independent rewards but there is a structured interaction between their transition probabilities. Both of these models can be optimally and approximately solved using my Coverage Set Algorithm. I also extend the first model by allowing the agents to communicate and I introduce an algorithm that finds an optimal joint communication policy for a fixed joint domain-level policy.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

There are as many definitions of what constitutes an agent as there are agent researchers. The view ranges from a simple entity that performs a task when activated, perhaps nothing more than a function in a program, to sophisticated entities that observe the world around them and make intelligent decisions based upon those observations. The view of an agent taken in this dissertation is closer to the latter rather than the former. One of the key requirements is autonomy, whereby an agent does not simply *react*, but has the ability to *act* and does so based upon its local information.

Within the human experience, human agents rarely act in isolation. Instead we form cooperative and competitive multi-agent systems at many different levels, from the macroscopic levels of countries, societies and economies to the microscopic levels of companies, families and sports. When designing an artificial system to address or solve a particular problem, the designers are often faced with a choice to design a system with a single autonomous agent (assuming that autonomy is appropriate and desired for this problem) or to have multiple autonomous agents. Multi-agent systems are appropriate in competitive settings where the different competing interests each need representation, like an auction where each party involved is trying to maximize the value of goods they buy while minimizing the price they pay. They are also appropriate in cooperative settings where the information necessary to solving the problem is distributed spatially or geographically and centralizing the information is not practical. Even when the information is centrally located, a multi-agent solution

may still provide a way of adapting to changes in the environment given an ability to swap agents in and out, or enabling scaling to larger problems given their numerous processors, sensors and effectors.

The research presented in this dissertation is entirely focused on cooperative multi-agent systems. It assumes that using a collection of autonomous agents has already been deemed an appropriate, preferred or only solution to the problem. It also applies only to cooperative problems, which means that all of the agents share the same goals or are trying to maximize a social utility instead of an individual, personal utility.

1.1 Cooperative Multi-Agent Systems

The field of cooperative multi-agent systems has made significant progress in the past few decades, yet it has not kept pace with the progress of research on single-agent systems. This is not surprising given that single-agent systems are a special case of multi-agent systems, and therefore much simpler to understand and model. Consider rover exploration of Mars. The goal is to gather information about Mars by sending semi-autonomous rovers there to explore and beam information back to us on Earth. To date, every rover sent to Mars has been part of a single-agent system. The most recent landing involved two rovers, but they were placed on different sides of the world and have no interaction. Thus they form two single-agent systems instead of one two-agent system.

There are many reasons why having multiple rovers close enough to interact is beneficial. For example, it may be valuable to have two pictures taken of the same object from different angles simultaneously. One rover could observe another rover to help determine its exact location, to help it navigate, or to help diagnose problems. One rover could relay information to and from another rover exploring places where geography limits its communication range. Multiple heterogenous rovers can expand

the type of information collected, and having multiple rovers increases the redundancy and reduces the chance of catastrophic failure.

However, multiple rovers also increases the complexity of the system. Due to that complexity, most traditional research in multi-agent systems has taken a heuristic or approximate approach that restricts the space of possible agent behaviors *a priori*, for example GRATE*[39], the contract net protocol [66], GPGP/TAEMS [41] and STEAM [69]. All four frameworks have been used very successfully in multi-agent systems, yet each of them only approximate the underlying problem and most solutions only find approximate answers within the framework. There is no guarantee on the quality of the solution and performance is usually measured relative to other approximate answers.

Recently, a number of researchers have been looking at the success of formal, decision-theoretic models in single agent systems like the Markov Decision Process (MDP) and are attempting to migrate those ideas to systems with two or more agents. The major advantage of the decision-theoretic models is that they have the potential to lead to significantly better solutions. They are both a much more accurate representation of the underlying problem and the notion of optimal solutions is easily defined within them.

Within the multi-agent decision-theoretic community, the research has been in two primary directions. On one side is a direct attempt to modify the MDP for multiple agents. A classic example of this is Boutilier's Multi-agent Markov Decision Process (MMDP) [9]. The assumption used in the MMDP is that each agent observes the entire state of the world. Solving this problem is no more difficult than solving an MDP, but it has the disadvantage of being restricted to solving only fully observable problems or problems with free communication.

The other direction uses the much more general assumption that each agent does not have complete knowledge of the world state. This is more of an exten-

sion of Partially Observable Markov Decision Processes (POMDPs) [40] to multiple agents. One example is Bernstein et al.'s Decentralized POMDP (DEC-POMDP) [6]. This assumption is much more widely applicable to multi-agent systems, but it also significantly increases the complexity. Just as POMDPs (PSPACE-complete) [48, 56] are much more difficult to solve than MDPs (P-complete) [56], DEC-POMDPs (NEXP-complete) [6] are significantly more complicated than MMDPs (P-complete) [9]. These and other related models will be discussed in detail in Chapter 2.

Understanding these two dichotomies (heuristic vs. decision-theoretic, and fully observable vs. partially observable) is central to understanding the importance of this work. Most of the existing work is found at the extremes of these dichotomies. Heuristic models have the advantage of being easily solved and widely applicable but at the cost of no guarantee on the quality of the solution. The decision-theoretic models have quality guarantees, but the fully observable models are not widely applicable and the partially observable models are not tractable. This work is an attempt to identify and solve models of multi-agent systems that lie in the middle and incorporate the advantages of each.

1.2 Contributions

An example of the classes of problems I am focusing on in this work can be found by looking at the Mars rover exploration problem in more detail. There are three homogeneous rovers exploring Mars, R_1, R_2 and R_3 (see Figure 1.1). Each is given a region of space that it is responsible for exploring. Each rover has a number of interesting locations to visit and collect data from, but there is not enough time in the day to do them all. Additionally, some of the locations are on the border between regions and can be explored by multiple rovers. Having two rovers collect data at the same location can either be beneficial (complementary data) or wasteful (redundant data). The decision problem concerns which locations to visit. The difficulty is that

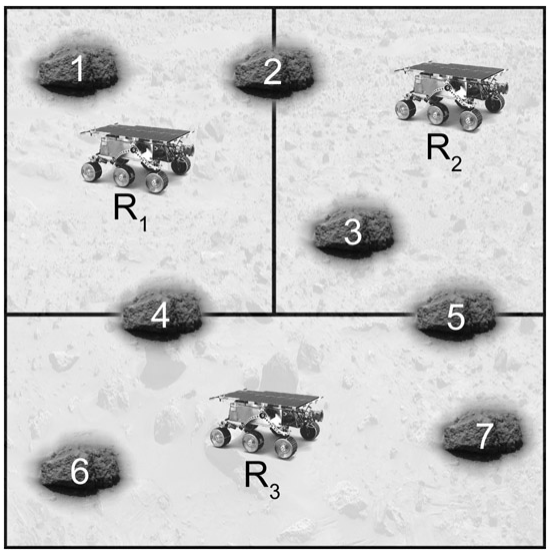


Figure 1.1. Illustration of the Mars rover example.

there is uncertainty in travel time between locations and in the time it takes to collect the data. The goal is to maximize the value of all of the collected information from the three rovers.

When the rovers are not allowed to communicate, the interaction between them is only through the global reward function being maximized. The actions each rover takes does not affect and is unaffected by the actions taken by the other rovers. This class of problems is called Transition Independent DEC-MDPs (TI-DEC-MDPs) and is formally defined in Chapter 3. This problem is more general than the MMDP because the agents have different and incomplete views of the world. It is also more specific than the DEC-MDP because the interaction between the agents is only through the reward function. The complexity of problems in this class is NP-complete, which is also harder than the MMDP but much easier than the general DEC-MDP.

I also investigate other types of interactions between agents. Instead of allowing the agents to interact through the reward function, the agents interact through the transition function using a mechanism I call event-driven interactions. The global

reward being maximized is just the sum of the rewards accumulated by each agent individually, but the actions taken by one agent can affect actions taken by another agent. For example, one rover may carry equipment needed by another and it could deposit the equipment at a prespecified location. The other rover can detect the existence of the equipment when it arrives at that location. This model is called Decentralized MDPs with Event-Driven Interactions and is formally defined in Chapter 5.

Identifying a class of problems by itself is not useful unless there exists a good algorithm to find solutions. To this end, Chapter 4 describes an algorithm I developed called the Coverage Set Algorithm (CSA), which returns an optimal solution to the TI-DEC-MDP and the DEC-MDP with Event Driven Interactions. A simple hill-climbing algorithm for these types of problems would converge to a local maxima. The CSA gets around this problem by efficiently searching for all of the local maxima. In general, the number of local maxima is significantly fewer than the number of policies so the algorithm can prune a majority of the policies. The algorithm can also return the best of the local maxima found at any point, making it an anytime algorithm. It has performed very well in experimental work both as an exact algorithm and as an anytime solution.

Neither of the two models I have discussed allow a general form of communication between the agents. This is because finding optimal solutions with communication is significantly more difficult than without, even though certain communication protocols have the same worst-case complexity [28]. Chapter 6 extends the TI-DEC-MDP to allow communication between the agents. I then illustrate how the common myopic assumptions used to deal with communication can lead to undesirable behavior in the agents. Finally, I present a new algorithm that addresses these sources of error and produces significantly better results. This new algorithm finds the optimal communi-

cation policy for fixed domain-level policies. An efficient algorithm to simultaneously find optimal communication and domain policies remains an open problem.

The work presented here will affect the state-of-the-art in multi-agent systems in four primary ways. First, the three classes of problems expand the body of knowledge about multi-agent systems in an area that is currently quite scarce. It identifies structure in many real multi-agent problems that can be exploited to reduce the complexity of finding optimal solutions. It also grounds problems that were previously solved heuristically in a more formal decision-theoretic framework.

The second effect is that the Coverage Set Algorithm is one of the first practical algorithms to solve optimally a general class of decentralized multi-agent systems. This leads the way for new and innovative solutions for optimal control of multiple agents.

The third impact this work will have is to further the work on approximation algorithms for multi-agent systems. While all heuristic algorithms for multi-agent systems are approximations, approximate algorithms grounded in decision theory are quite new and have the potential to significantly outperform the ad-hoc, heuristic ones. However, very few existing approximate algorithms for multi-agent systems provide any guarantees on the solution outcome. The CSA has the distinction of guaranteeing convergence on the optimal solution.

Finally, I present an analysis of two ways a myopic assumption commonly used to deal with communication can negatively affect the agents' behavior. The algorithm I present to address the errors finds the optimal communication policy given a fixed domain policy. This is the first algorithm that can efficiently find optimal communication policies for problems of this size and complexity.

This dissertation is organized as follows. Chapter 2 introduces the foundational work this proposal builds on in addition to placing this work in the context of related works. Chapter 3 presents the first model, Transition Independent DEC-MDPs. The

Coverage Set Algorithm is discussed in Chapter 4 along with some experimental results from using it to solve TI-DEC-MDPs. The next model, DEC-MDPs with Event Driven Interactions, is the topic of Chapter 5, and Chapter 6 examines adding communication and shows how to find optimal communication policies. Chapter 7 will summarize and conclude the dissertation.

CHAPTER 2

RELATED WORK

The work presented here addresses one of the fundamental challenges within multi-agent systems–coordination. There are many definitions for what coordination is, and the one that I am adopting is *the process in which an agent takes into account other agents when making its decisions*. Coordination does not imply cooperation, however, in this work I am only considering systems of cooperative agents. In this case, the agents are coordinating for their mutual benefit.

A framework for coordination can be divided into two components: 1) a representation of the space of coordinated behaviors, and 2) a mechanism or algorithm for searching that space and selecting a particular set of coordinated behaviors. The traditional approaches to coordination are to find approximate but efficient representations of the space of behaviors and then to do an incomplete but fast heuristic-guided search of that space. The recent decision-theoretic models take the opposite approach of an exact but less efficient representation of the space and perform a more complete, but slower search through that space.

2.1 Traditional Approaches to Coordination

Multi-agent coordination is an extremely complex problem, and a natural way to approach it is to simplify the problem. Traditionally, researchers have done just that by introducing heuristics/assumptions, imposing structure, and settling for satisficing solutions. When viewing coordination as a distributed search process through the space of possible agent behaviors, these approaches correspond to pruning the search

space and halting before the entire space has been searched. Indeed, Durfee goes as far as claiming that the bulk of coordination research has been in developing techniques to make agents **more** ignorant about their situation instead of less [19]. That counter-intuitive claim flies in the face of information theory, which says that additional (correct) information always has a non-negative value. However, in situations where an agent does not have the computational capacity to make the perfect decision, giving it more correct but irrelevant information causes some of that scarce capacity to be taken away from the more relevant data leading to a worse decision. There are other situations in which additional data can *distract* an agent and actively lead it to pursue a bad line of reasoning, resulting in a worse decision [13, 22, 64].

A popular framework for control of a single agent is the belief-desire-intention(BDI) model [14, 11]. The BDI model is inspired by a philosophical model of reasoning in humans, which characterizes the state of an agent by its beliefs about the world, their long-term goals that they desire to achieve, and their short-term goals that they intend to actively work towards. This framework has been extended to multiple agents by the additions of *joint intentions*[42] and *SharedPlans*[29]. Two specific implementations of this approach into a general model for teamwork-based coordination are STEAM and GRATE*.

Both STEAM[69] and GRATE*[39] are rule-based systems, where the space of coordination behaviors are represented and highly constrained by a set of logical rules. These rules are chosen because they make intuitive sense and have resulted in the desired behavior during experimentation, not because they have been evaluated in a quantitative way. Search through this space is done with an inference engine, which selects a set of rules (behaviors) that will achieve the goal, and is generally qualitative in nature. However, STEAM and an extension to it STEAM-L[70] both include a decision-theoretic component that allow pieces of the problem to be quantitatively evaluated and maximized.

Another framework for coordination is the contract net protocol [66]. Interaction between the agents happens through an auction mechanism and is applicable in both cooperative and competitive settings. When one agent needs assistance, it announces its request to other agents. The other agents bid on how well they can do the job and the originating agent awards the contract to its favorite bidder. The space of coordination between the agents is limited to this auction mechanism, and the process of finding an allocation of tasks to agents is very short sighted and the decisions are made with essentially no non-local information. There is no inherent mechanism to handle multi-link negotiations and get the agents to all agree to work towards one goal, which can lead to no goals being accomplished. A contract net protocol is best suited for problems that are naturally decomposable and not tightly constrained.

GPGP/TAEMS [41] is a general model for distributed coordination. TAEMS is a powerful hierarchical task description language that can quantitatively and efficiently describe a set of tasks, decompositions, and logical and temporal constraints among the activities. GPGP is the coordination module, and a copy resides in each agent. When there is an interdependency between two agents (e.g. a subtask for one agent facilitates or hinders a subtask done by another agent, or they both are a part of the same quality accumulation function) the GPGP modules attempt to coordinate the agents' behavior. The coordination space GPGP searches through is represented by commitments one agent makes to another, e.g. an agent may commit to finish its subtask by a particular deadline. This representation is limited by the incomplete knowledge one agent receives about the other agent through this commitment, for example the commitment represents a hard deadline when the task will be completed instead of a more accurate distribution over possible completion times. To address this Ping and Lesser [79] introduced a more sophisticated commitment that incorporated uncertainty.

Given this partial global view, the GPGP component in the agent calls a local scheduler that quantitatively evaluates the quality of the solution achieved with this commitment. The scheduler can also make recommendation about alternative commitments that would lead to a better solution, which GPGP uses to guide the negotiation with the other agent. GPGP chooses from a suite of negotiation strategies the one that best matches the situation. This search is not complete because it is being performed online in a distributed process, subject to communication costs and limited computational power. Due to these limitations GPGP does not analyze all contingencies, does not find the optimal schedule given the constraints, and does not exhaustively search the space of constraints. However, when unexpected contingencies arise it can plan how to handle them on the fly.

To contrast with these approaches, the models introduced in this work use a complete but less efficient representation of the space of coordination behaviors. The two algorithms presented here quantitatively search the complete space of domain actions or communicative actions (but not both simultaneously) to find optimal solutions. However, they are designed to be used offline and they perform a centralized search for a distributed coordination strategy and therefore are not as concerned with the tradeoff between computation time and solution quality, nor the issue of communication bandwidth while planning.

2.2 Decision-Theoretic Models

The work presented here builds on a solid foundation of decision theory, and is related to other multi-agent, decision-theoretic work. The two primary single-agent models, the Markov Decision Process (MDP) and the Partially Observable MDP (POMDP) serve as the foundation to all of the multi-agent models, including those I present in this dissertation. I will categorize the existing models into two classes: one is an extension of MDPs and the other an extension of POMDPs. These two

classes have a number of differences, most notably complexity and generality. The multi-agent extension to POMDPs is much more general, but also more complex than the other. The models presented in this dissertation fall between these two extremes and are attempts to strike a balance between complexity and generality, as well as to present more efficient solutions to several structured classes of multi-agent systems.

2.2.1 Single-Agent Models

There has been a lot of highly successful work in decision-theoretic control of single agent systems. The most successful is the Markov Decision Process (MDP). The MDP is a simple yet powerful model that serves as the basis for this entire field of research. The basic idea is to describe a sequential decision problem as a finite set of distinct states of the world, S. Each state includes all of the information available to the agent making the decision that is necessary for it to make the best possible decision. When an agent makes its decision, known as taking an action from the set of possible actions A, the state changes to a new state according to a known probability distribution P and the agent receives a reward (or penalty) R. When the entire model is known (states, actions, transition probabilities, and reward) the agent can easily compute its optimal choice of actions for each state, which is the action that maximizes the agent's expected cumulative reward. This mapping from states to actions is the policy π. This is a P-complete problem for both the finite-horizon and the discounted infinite-horizon cases [56].

While the MDP can capture a diverse array of problems, many problems contain additional uncertainty that is not easily represented. One type is uncertainty about the current state. Instead of observing the new state after each action, the agent may receive a noisy observation about the new state. This leaves the agent with a probability distribution over the states as its belief about the current state. These noisy observations can be caused by many factors, such as a robot's noisy sensors

returning imperfect information, or an inability to sense parts of the environment. Consider, for example, a robot attempting to locate itself in a building using only a video camera and an internal map of the building [71]. It starts in a hallway, but there may be many such hallways in its mental map and it initially has no way to distinguish them. As it moves around and discovers intersections and other distinguishing features, it can slowly narrow down its possible locations. However, as interpreting video footage is not perfect, it may never be one-hundred percent certain about its location.

The Partially Observable MDP (POMDP) handles this uncertainty about the current state. At each step the agent receives an observation $o \in \Omega$, which changes its belief about the current state. While the problem still has the Markov property (the probability of the next state only depends on the current state and action taken) the agent no longer has direct access to the Markov state. Instead, it must make its decisions based on the information available to it: the sequence of observations made. The agent's policy is now a mapping from a sequence of observations to an action. This creates an exponential increase in the size of the policy, and is a much harder problem to solve: PSPACE-complete for the finite-horizon case [48, 56], and undecidable for the infinite-horizon version [44].

2.2.2 Centralized Multi-Agent Models

I classify the multi-agent decision problems into two groups depending on whether they are extensions of MDPs or POMDPs. The characteristic difference between the two groups is the same partial observability that distinguishes MDPs from POMDPs. The first group is exemplified by Boutilier's Multi-agent Markov Decision Process (MMDP) [9]. The MMDP is a straightforward extension of the MDP to multiple agents. The only change to the model is to factor the action space into actions for each of the agents. The transition and reward functions then become mappings from

states and **joint** actions to a next state and reward, respectively. The expressive power of the MMDP is the same as the MDP and its complexity is also polynomial in the number of states and actions. The same algorithms used for the single agent MDPs can be used to find the optimal joint policy for the MMDP, such as dynamic programming [36], linear programming [46, 16], and LAO* [34]. There has also been some significant work done in multi-agent reinforcement learning (RL) [10, 23, 74].

However, while the change to the model is fairly simple, there are two implications that may not be entirely obvious. First, the action space is now exponential in the number of agents. Second, even though the agents share the state space and it cannot necessarily be factored into a component for each agent, in many problems adding additional agents results in an exponential increase in the number of states.

This curse of dimensionality is a problem for single agent MDPs and POMDPs too, since the state space is exponential in the number of state features. One approach developed for use with MDPs is to use a factored representation of the state space [31]. Some researchers have successfully extended that representation to the MMDP to handle larger numbers of agents [17, 30, 32]. Other researchers have approached the problem using hierarchies, in particular hierarchical reinforcement learning for single agents [2] and for centralized multi-agent problems [24].

The disadvantage of the MMDP and related approaches is that it makes an important assumption that renders it inappropriate for many multi-agent systems. This is that each agent has the same (complete) world view. Agents can have the same world view because their sensors let them directly see all of the relevant information, or because communication is free and the agents share their observations at each step. I call these models centralized multi-agent models because the optimal solution can be easily found through centralizing the views of each agent. These models are different than the ones I present in this dissertation in that I do not assume communication

is free. Instead, it is either not possible or it has a cost and must be reasoned about explicitly before being used.

Ooi and Wornell [53] studied a variation where all of the agents automatically share their state information every K time steps. They developed a dynamic programming algorithm to derive optimal policies for this case, though the state space for the algorithm grows *doubly exponentially* with K. A more tractable algorithm was developed by Hsu and Marcus [38] under the assumption that the agents share state information every time step, though it can take a time step for the information to propagate. Both of these models are still centralized models because, even though the information is not always synchronized, the synchronization happens in a fixed way and is not influenced by or subject to the decisions of the agents.

2.2.3 Distributed POMDPs

Adding partial observability to the MMDP (or extending the POMDP to multiple agents) brings us to the realm of distributed POMDPs. These models are much more general because they allow each agent to make their own partial observations of the world instead of forcing them to share the same complete world view. There are two general classes of distributed POMDPs in common use and a few subclasses that exploit structure in the problems.

2.2.3.1 Decentralized POMDPs

The first of the two general classes of distributed POMDPs is composed of a number of different models. In general, they do to the POMDP what the MMDP did to the MDP: factor the action and observation spaces into an action and observation for each agent, and define the transition, reward and observation functions over the joint actions and joint observations. One particular model is Bernstein et al.'s Decentralized POMDP (DEC-POMDP) [6].

Definition 1 *An n-agent* **DEC-POMDP** *is defined by a tuple* $\langle S, A, P, R, \Omega, O \rangle$, *where*

- S *is a finite set of world states, with a distinguished initial state* s^0.
- $A = A_1 \times A_2 \times \ldots \times A_n$ *is a finite set of joint actions.* A_i *indicates the set of actions that can be taken by agent* i.
- $P : S \times A \times S \rightarrow [0, 1]$ *is the transition function.* $P(s'|s, (a_1 \ldots a_n))$ *is the probability of the outcome state* s' *when the joint action* $(a_1 \ldots a_n)$ *is taken in state* s.
- $R : S \times A \times S \rightarrow \Re$ *is the reward function.* $R(s, (a_1 \ldots a_n), s')$ *is the reward obtained from taking joint action* $(a_1 \ldots a_n)$ *in state* s *and transitioning to state* s'.
- $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ *is a finite set of joint observations.* Ω_i *is the set of observations for agent* i.
- $O : S \times A \times S \times \Omega \rightarrow [0, 1]$ *is the observation function.* $O(s, (a_1 \ldots a_n), s', (o_1 \ldots o_n))$ *is the probability of agents 1 through n seeing observations* o_1 *through* o_n *(agent* i *sees* o_i*) after the sequence* s, $(a_1 \ldots a_n)$, s' *occurs.*
- *Joint partial observability: the n-tuple of observations made by the agents together does not (necessarily) fully determine the current state.*

The policy π for a DEC-POMDP is a joint policy composed of a local policy π_i for each agent, $\pi = (\pi_1 \ldots \pi_n)$. Each local policy is similar to a policy for a POMDP, it is a mapping from sequences of observations to an action, $\pi_i : \bar{\Omega}_i \rightarrow A_i$. The goal is to find a joint policy that maximizes the expected reward for the system (see [6] for the value function).

While the MDP and the MMDP were very similar, the fact that the agents have *different* partial views introduces a significant new source of complexity over the

POMDP. In a POMDP, the single agent maintains a belief about the current state. In the DEC-POMDP, each agent maintains its own different belief about the current state as well as maintaining a belief about the beliefs of each other agent. This belief of beliefs complicates the problem significantly, giving it a complexity of NEXP-complete [6], compared to the POMDP's PSPACE-complete.

Early work in this area identified two characteristics that the models were differentiated by. The first characteristic is *joint partial observability* versus *joint full observability*. Joint partial observability is where the set of observations made at time t together would not (necessarily) identify the current world state. This is the case of the DEC-POMDP (see Definition 1). With joint full observability the set of observations together do identify the current world state: if $O(s, a_1 \ldots a_n, s', o_1 \ldots o_n) > 0$ then $P(s'|o_1 \ldots o_n) = 1$. The Decentralized MDP (DEC-MDP) is defined identically to the DEC-POMDP except that it has joint full observability instead of joint partial observability. The differences between the DEC-MDP and the DEC-POMDP comes out if we allow free communication between the agents or reduce the number of agents. A DEC-MDP with free communication or only one agent reduces to an MDP. The DEC-POMDP in those cases reduces to a POMDP.

It turns out that this distinction, while potentially useful when adding additional restrictions to the class, does not change the expressiveness of the models. The DEC-MDP is trivially a subclass of the DEC-POMDP because joint full observability is a special case of joint partial observability. Similarly, one can take an arbitrary DEC-POMDP and convert it to an equivalent DEC-MDP by adding an additional agent that observes everything and does nothing–there is now joint full observability. Essentially, an n-agent DEC-POMDP is equivalent to an $n + 1$-agent DEC-MDP. The question was whether a 2-agent DEC-MDP was easier to solve than a 2-agent DEC-POMDP. Unfortunately, it is just as difficult.

	Implicit Communication	**Explicit Communication**
Joint Full Observability	DEC-MDP [6]	Dec-MDP-Com [28] Xuan Model [76]
Joint Partial Observability	DEC-POMDP [6] MTDP [59] POIPSG [58] RMTDP [49]	Dec-POMDP-Com [28] Com-MTPD [59]

Table 2.1. Equivalent distributed POMDP models sorted by characteristics.

The second characteristic is how communication between agents is handled. In some models like the DEC-MDP and DEC-POMDP, there is no explicit communication. To address this perceived shortcoming, new models like the Dec-POMDP-Com [28] and the Com-MTDP [59] were developed. In these models, explicit communication is handled by adding the language of communication Σ, and the cost for sending a message C_Σ. The decision process is then broken up into two phases. First, the agents all take an action (sometimes called a domain action to distinguish it from communication actions). Then the state transitions based on the joint actions of the agents and they recieve their observations. In the next phase, the agents decide what to communicate, with ϵ denoting an empty message (no communication). The agents all receive the messages sent to them and then the next step starts with phase one. Like the previous characteristic this distinction does not change the expressiveness of the models. A communication action can be represented by a domain action that just changes the observation received by the receiving agents (see [62] for a proof).

However, an explicit representation of communication can be very useful in that it allows one to specify different properties of domain-level actions that communicative acts would violate. Worth mentioning specifically is RMTDP [49], which takes a similar approach but with a different type of coordination action based on role allocation.

There are a number of equivalent models that fall into these four categories, see Table 2.1. Generally, the models within a category are trivially equivalent. See [28, 62, 61] for equivalence results of the various models in different categories.

2.2.3.2 Interactive POMDPs

Interactive POMDPs (I-POMDPs) [25] are another attempt to extend POMDPs to multiple agents, however it takes a very different approach from the models like the DEC-POMDP. The basic idea is to not only represent an agent's belief about the world, like in a POMDP, but to also explicitly represent its beliefs about other agents in the world. Given these beliefs, the agent can locally compute its optimal action. This approach is more general than the DEC-POMDP in that it can model self-interested agents. However, it also has some limitations which make it more of an approximation for cooperative problems.

Models of other agents can be classified into levels. A level-0 model is essentially having no model of the other agent. A level-1 model assumes the other agent has a level-0 model of this agent. In general, a level-n model of an agent assumes that the modeled agent only has level-$n-1$ models in its beliefs about other agents. Trying to model an agent assuming it is of the same level leads to level-∞ agents because of the self-reference.

Level-∞ models are not directly computable. However, some coordination problems require it to find the optimal solution. For example, consider a fully observable coordination problem in which the coordinated action *risky* has high value if both agents do it and extreme negative value if only one attempts it. A level-0 agent would assume that there is some chance of failure and take the *safe* action. A level-1 agent would assume that the other agent is a level-0 agent, which chooses *safe*, and therefore it would also choose *safe*. By induction, a level-n agent will always choose *safe*. When a joint policy is evaluated in a DEC-POMDP it is irrelevant to any level

of modeling in the agents, which would make it analogous to level-∞ agents. In other words, I believe that there exist problems for which the best computable solution in an I-POMDP are arbitrarily bad. However, since optimal solutions are intractable for all but small toy problems, I-POMDPs may be able to provide competitive approximate solutions [18] within a reasonable time compared to other distributed POMDP models and algorithms, and therefore it should not be dismissed.

They claim the complexity of exactly solving an I-POMDP to be equivalent to solving $O(M^l)$ POMDPs, where M is a bound on the number of models of the other agent and l is the level of the agent. Since M is at least as large as the number of policies of the other agent, the complexity is worse than a brute-force search over the policies in a DEC-POMDP. I suspect this is because the model and solution are more general to also handle competitive problems. This suggests that approaches tailored to cooperative problems may perform significantly better on cooperative problems than more general approaches.

2.2.3.3 Subclasses of DEC-POMDPs

An important approach to solving multi-agent problems is to identify significant subclasses with a lower complexity. We do this by identifying structure in the problem that we can exploit in the search for an optimal policy. This is the primary approach taken in this dissertation. The two primary models I present here, Transition Independent DEC-MDP (TI-DEC-MDP) and DEC-MDP with Event Driven Interactions (DEC-MDP-EDI) can be found in detail in chapters 3 and 5. Here, I will mention the related models that came before, as well as how others have built on these models or used some of the same structure to define their own models.

Xuan et al. [77] did some important early work with agents that interact only through a global reward function. This is an equivalent class of problems to the TI-DEC-MDP, which I formally defined relative to the DEC-MDP, and proved its

complexity to be NP-complete [4, 5]. I defined a set of independence relationships which describe naturally occurring structure in many multi-agent systems. The Networked Distributed POMDP (ND-POMDP) [52] uses the same structure but assumes joint partial observability instead of joint full observability. This change, however, increases the complexity to PSPACE-hard (since a single agent ND-POMDP reduces to a POMDP).

The second model I present, DEC-MDP-EDI, can represent more sophisticated interactions between agents, including the interactions found in GPGP. However, as the expressiveness of this model is greater than the TI-DEC-MDP, so is the complexity– NEXP in the number of interactions between the agents. Beynier et al. [7, 8] have applied this model to additional problems.

The final set of subclasses is based on Goal-Oriented DEC-MDPs (GO-Dec-MDP). A GO-Dec-MDP is a DEC-MDP with a set of goal states. For every step the agents receive a negative reward until they reach a goal, where they receive an additional (presumably positive) reward. Goldman et al. [28] examines the complexity of a number of subclasses of GO-Dec-MDPs.

2.2.3.4 Algorithms for Distributed POMDPs

The general models for distributed POMDPs have a very high complexity for finding an optimal joint policy, NEXP-complete [6]. There are only three known algorithms to optimally solve these models: Hansen et al.'s Dynamic Programming (DP) for DEC-POMDPs [33], Szer et al.'s Multi-Agent A* (MAA*) [67] and brute-force policy search. DP for DEC-POMDPs can prune regions of the policy space that are dominated by other policies, similar to DP for POMDPs, though the rules for domination are more complex. Its performance has not yet been extensively studied, however at best it will not be able to outperform DP on POMDPs, which has received significant attention and is quite difficult.

MAA* is a heuristic search algorithm based on the well-known A* algorithm. It relies on having an admissible heuristic to prune branches in the search tree. They present several domain-independent heuristics: using the underlying centralized MDP, the underlying POMDP, and recursive calls to MAA* itself. They found in comparing it to DP that it considered more policy pairs than DP, but that it required significantly less memory than DP. Thus, it was able to solve problems in which DP runs out of memory (4 states, 2 actions, 2 observations and a horizon of 4).

Optimal algorithms for the subclasses of DEC-POMDPs are also very few. In chapter 4 I present the Coverage Set Algorithm (CSA), which finds optimal solutions to both the TI-DEC-MDP and the DEC-MDP-EDI using an anytime approach, making it fairly versatile. Goldman and Zilberstein [28] show that for some highly structured GO-Dec-MDPs the problem is P-complete and their OptNGoals algorithm will find the optimal solution.

Due to the complexity of the problems, there are many more approximate algorithms than optimal ones. We can categorize the approximate algorithms into four general classes of algorithms: iterative, decomposition, centralized and reinforcement learning. The first three categories of algorithms are primarily designed to be run offline in a centralized fashion (with a few exceptions). However, the policies generated are distributed, meaning that they generate a local policy for each agent that is a mapping from its observation(s) to an action, while not knowing exactly what action the other agents are taking. Many of these algorithms can be adapted for either running online or in a distributed way, but they were designed as offline, centralized algorithms. The RL algorithms, however, are designed to be run online and distributed, which means that each agent is computing its own policy based on its past experience.

Iterative Algorithm

By far the most common class of approximate algorithms are the iterative algorithms. The basic idea is that if all of the agents but one are assigned an arbitrary policy, computing that agent's optimal corresponding policy reduces to an easier problem (a POMDP if the original problem was a DEC-MDP or DEC-POMDP or equivalent). That agent then fixes its policy and another agent finds its corresponding optimal policy. This process repeats until no agent can change its policy to improve the expected value. This process is guaranteed to converge because at each iteration the expected value of the joint policy is either increased or stays the same. If it increases, it will have to stop increasing when it reaches the expected value of the optimal joint policy. If it stays the same for all agents it has converged to a Nash equilibrium, which in this case is a local maxima.

Researchers have published a number of variations of this iterative algorithm for the DEC-POMDP class of distributed problems [12, 63, 20, 21]. Nair et al. [51] presented a family of iterative algorithms known as JESP, for which they have published many extensions. For example, DP-JESP [51] uses dynamic programming to find the agents' optimal corresponding policy, Communicative DP-JESP [50] uses communication to reduce the convergence time and improve the quality of the solution, CS-JESP [72] runs on continuous spaces, and LID-JESP [52] exploits the locality of interaction when not every agent interacts with every other agent.

Peshkin et al. [58] presented an algorithm that is a little different than the typical iterative approach called joint gradient descent. In it each of the agents runs a gradient descent algorithm in parallel. While it is not technically an iterative algorithm since the agents are not iterating their improvement, it is performing a similar type of search.

Decomposition Algorithms

Decomposition algorithms use a mechanism to decompose the search process into a

separate search for each agent, typically represented as an MDP or a POMDP. The agents can then easily compute their optimal policy using single agent algorithms like dynamic programming for MDPs. Xuan et al. [77] and Goldman et al. [27] both worked on goal oriented problems where finding a common goal for the agents to work towards decomposes their control processes, for example the agent meeting problem. In that problem the agents would mutually decide on a location to meet and given that location there was no interaction between them until one agent decides that they should change the goal. This approach works well for those problems in which good goals can be easily computed.

Beynier et al. [7, 8] used the idea of opportunity cost to decompose the agents local decision processes. The opportunity cost represents the loss in expected reward for taking different actions due to non-local effects. This makes the most sense in problems like the DEC-MDP-EDI where the global reward function is the sum of local reward functions, one for each agent. If agent 1 uses all of some shared resource, its local expected reward may go up, but at the same time the local reward of agent 2 may go down. The opportunity cost reflects this loss in reward so agent 1 can make a more informed decision about using the resource. Computing the correct opportunity cost function that will completely decompose the agents is no easier than solving the original problem. However, efficient approximations of the opportunity cost can result in good joint solutions.

Centralized to Decentralized Algorithms

This class of algorithms reduces the distributed problem to a centralized problem like an MMDP or POMDP and finds the centralized solution. It then presents a way of converting this centralized algorithm back into a distributed algorithm. The typical way of generating the centralized solution is to assume the agents can communicate freely at each step. Xuan et al. [76] starts with that centralized policy and reduces

unnecessary communication by having the agents communicate only when the optimal action they should take is ambiguous without communication. That policy conforms to the original centralized policy but with less than complete communication. They also provide a series of non-conforming transformations that allows them to reduce the amount of communication but at the cost of also reducing the reward.

Shen et al. [63] take a different approach. Instead of using the centralized policy directly, they use the values of the states given that policy. The agents compute their best actions given their current belief state by taking the sum of the expected value of the action in each state times the agent's belief that it is in that state.

Roth et al. [60] have yet another approach. They have each agent compute the same tree of possible joint beliefs, ignoring the actual observations that the agents made. The joint action is chosen from this tree and the centralized policy. The agents then decide whether or not to communicate depending on whether the observation they did make would suggest them to take a different action.

The algorithm I present in Chapter 6 takes an opposite approach in that I assume no communication between the agents and compute their optimal corresponding policies. Then I add communication back in by examining the expected increase in value if they do communicate. This algorithm also does not fall into the category of decomposition algorithms because even though the agents are not communicating, they still interact through the reward function.

Reinforcement Learning Algorithms

Another approach to solving distributed POMDPs is reinforcement learning (RL) [24, 1, 65]. The three previous categories of algorithms generally use an offline, centralized computation of a distributed policy, however, RL allows the agents to learn policies online in a distributed way. There are a number of advantages and disadvantages to using RL. Some advantages are that RL algorithms generally do not require a

model of the problem, and they can take advantage of the distributed resources of the multiple agents.

On the down side, it is difficult to guarantee that the agents will not jump off the cliff, metaphorically speaking, as they explore the space. It has also been difficult to provide strong theoretical convergence guarantees like those found in single-agent cases and in the fully observable multi-agent case. The best convergence guarantees have only been found in restricted circumstances [1]. This is because from the perspective of each learning agent, the problem is non-stationary due to the other learning agents changing their behavior. While in practice the partially observable multi-agent RL algorithms tend to converge, very few have theoretical proofs of convergence to a local maxima, and to the best of my knowledge, no one has proven convergence to the globally optimal solution.

2.3 Hybrid Approaches

Distributed POMDPs have such a high complexity that it is not clear that they are even feasible for approximating real world sized problems. However, a very promising way for them to impact real systems is in hybrid approaches [68]. A hybrid approach is one that combines the traditional approach to multi-agent systems, which is good at dealing with high complexity, with a distributed POMDP, which excels at finding quantitatively good solutions. The basic idea is to use the traditional approach to handle the complexity of the problem, and to model a smaller, critical component of the whole with a distributed POMDP.

Nair and Tambe [49] took a BDI approach and added a Role-based Markov Team Decision Problem (RMTDP) to quantitatively analyze the allocation of roles to agents. In their approach, the BDI team plans are able to provide initial, incomplete RMTDP policies, which reduces the search space of the distributed POMDP. Their hybrid approach significantly outperformed the pure BDI approach.

Pynadath and Tambe [59] show another use of distributed POMDPs: evaluating the performance of traditional approaches. They evaluate the communication decisions made by STEAM and GRATE*. Analysis of heuristics allows us to better understand when and why they succeed and fail, and can guide us towards designing better heuristics.

CHAPTER 3

TRANSITION INDEPENDENT DECENTRALIZED MDPS

This section will discuss one direction for reducing the complexity of multi-agent problems. Many traditional models of multi-agent systems make a clear distinction between the local decision problems for the agents and the more global coordination problem. For example, GPGP includes a coordination module that communicates with other agents to negotiate constraints over behavior for the agents. The agents take those constraints and do local planning and scheduling to estimate how well those constraints work for the agent. GPGP's approach, then, is coming from the single agent perspective. It starts with multiple single agents, each with their own local tasks to complete. The agents coordinate because their local problems interact and the expected utility can be increased through coordination.

When a problem is modeled by a DEC-MDP it loses the distinction between the local decision problems and the global coordination problems because there is only one world state that encompasses all of the agents. There is no distinction made between an action that only directly affects one agent (local problem) and one that directly affects all agents (global coordination problem). While some models like the COM-MTDP explicitly represent communication, the domain actions have not been restricted in a way that precludes direct communication between the agents and thus they see no computational benefit. The ideas presented in this section start with a DEC-MDP and add back in a notion of local problems for the agents that is separate from the global coordination problem. This new structure can be exploited to reduce the complexity of the problem.

3.1 Relation Between an n-agent DEC-MDP and n MDPs

To understand where an agent's local problem fits into the DEC-MDP one first needs to understand the relationship between an n-agent DEC-MDP and n single-agent MDPs. I will start with an n-agent DEC-MDP and detail the additional structure necessary to turn it into n separate MDPs. A DEC-MDP has the same definition as a DEC-POMDP, as described in the previous section, with the addition that there is joint full observability. This simply means that the set of observations the agents make at each step can collectively identify the current world state. The first structure the problem must have is a factored state space.

Definition 2 *A* **factored**, *n-agent DEC-MDP is a DEC-MDP such that the world state can be factored into $n+1$ components, $S = S_0 \times S_1 \times ... \times S_n$.*

Factoring the state space of a DEC-MDP could be done in many ways. The intention of such a factorization is a separation of features of the world state that belong to one agent from those of the others and from the external features. This separation is strict, meaning that no feature of the world state may belong to more than one group. S_0 refers to external features, which are parts of the state that the agents may observe and be affected by but do not affect themselves, such as weather or time. S_i refers to the set of state features for agent i, which is the part of the world state that an agent observes and affects. This will be made clearer by the following definitions.

I refer to $\hat{s}_i \in S_i \times S_0$ as the *local* state, $a_i \in A_i$ as the local action, and $o_i \in \Omega_i$ as the local observation for agent i. Since the agent is affected by the external features its local state must include them. Just as in a DEC-MDP, a local policy (or just policy) for one agent is a mapping from sequences of observations to local actions (I will simplify this later). A **joint policy**, $(\pi_1 ... \pi_n)$, is a set of policies, one for each agent.

Let us illustrate this with the rover exploration problem introduced previously. There are two rovers, each exploring adjacent areas. The global state is composed of each rovers' location, the information collected so far (including the agent that collected it), and the time left in the day. This state can be factored into a local state for each agent containing that rover's location, the information collected by that rover, and the time left in the day. The action each of the rovers can take is a movement action or an information collection action.

Definition 3 *A factored, n-agent DEC-MDP is said to be* **transition independent** *if there exist* P_0 *through* P_n *such that*

$$P(s_i'|(s_0 \ldots s_n),(a_1 \ldots a_n),(s_0' \ldots s_{i-1}', s_{i+1}' \ldots s_n')) = \begin{cases} P_0(s_0'|s_0) & i = 0 \\ P_i(s_i'|\hat{s}_i, a_i, s_0') & 1 \leq i \leq n \end{cases}$$

That is, the new local state of each agent depends only on its previous local state, the action taken by that agent, and the current external features. The external features change based only on the previous external features. This implies that $P((s_0' \ldots s_n')|(s_0 \ldots s_n),(a_1 \ldots a_n)) = \prod_{i=0}^{n} P_i(\cdot)$.

Definition 4 *A factored, n-agent DEC-MDP is said to be* **observation independent** *if there exist* O_1 *through* O_n *such that* $\forall o_i \in O_i$

$$P(o_i|(s_0 \ldots s_n),(a_1 \ldots a_n),(s_0' \ldots s_n'),(o_1 \ldots o_{i-1}, o_{i+1} \ldots o_n)) = P(o_i|\hat{s}_i, a_i, \hat{s}_i')$$

That is, the observation an agent sees depends only on that agent's current and next local state and current action.

Definition 5 *A factored, n-agent DEC-MDP is said to be* **locally fully observable** *if* $\forall o_i \ \exists \hat{s}_i \ : P(\hat{s}_i|o_i) = 1$.

That is, each agent fully observes its own local state at each step. While local full observability and observation independence are related, it is possible for one to be true without the other. However, when both are true then Ω and O in the definition of a factored n-agent DEC-MDP are redundant and can be removed. This also simplifies the policies. Instead of a local policy being a mapping from a sequence of observations to actions, it is a mapping of local states to actions just like in an MDP.

Definition 6 *A factored, n-agent DEC-MDP is said to be* **reward independent** *if there exist f and R_1 through R_n such that*

$$R\left((s_0 \ldots s_n),(a_1 \ldots a_n),(s'_0 \ldots s'_n)\right) = f\left(R_1(\hat{s}_1,a_1,\hat{s}'_1)\ldots R_n(\hat{s}_n,a_n,\hat{s}'_n)\right)$$

and

$$\begin{aligned} R_i(\hat{s}_i,a_i,\hat{s}'_i) &\leq R_i(\hat{s}_i,a'_i,\hat{s}''_i) \quad \Leftrightarrow \\ f\left(R_1 \ldots R_i(\hat{s}_i,a_i,\hat{s}'_i)\ldots R_n\right) &\leq f\left(R_1 \ldots R_i(\hat{s}_i,a'_i,\hat{s}''_i)\ldots R_n\right) \end{aligned}$$

That is, the overall reward is composed of a function of the local reward functions, each of which depends only on the local state and local action of one of the agents. This function is such that maximizing each of the local reward functions individually maximizes the function itself. An example function is $f(R_1 \ldots R_n) = \sum_i R_i$.

An n-agent factored DEC-MDP with the three independence relationships between the agents and local full observability is equivalent to n separate MDPs of the form $\langle S_0 \times S_i, A_i, P_i, R_i \rangle$. Each of the three independence relations detaches one component of the agents from each other: transitions, observations, and reward. The local full observability simplifies the policy to be a function of local states. The decision problem is correspondingly easier to solve, consisting of n smaller polynomial problems instead of one monolithic NEXP problem.

However, a problem so partitioned is no longer a multi-agent system. The space of problems I am examining falls in the middle. As a first step I assume each of those definitions hold except for the reward independence. This means that the agents are independent of one another except for the reward, which is still a function of all agents. There is also no communication between the agents because that would violate several of the assumptions, for example observation independence (the observation an agent sees would depend on the message sent by another agent). Communication will be added as a future step.

3.2 Zero Communication

The class of problems studied in this paper is characterized by two or more cooperative agents solving (mostly) independent local problems. The actions taken by one agent can not affect any other agents' observation or local state. Consequently, an agent can not observe the other agents' states and actions and can not communicate with them. The interaction between the agents happens through a global value function that is not simply a sum of the values obtained through each of the agents' local problems. The non-linear rewards combined with the decentralized view of the agents make the problem more difficult to solve than the MMDP, while the independence of the local problems make it easier to solve than the general DEC-MDP. I call this class of problems Transition Independent DEC-MDPs.

I am looking at problems where the value of an activity performed by one agent may depend on the activities of other agents. One example is information collection, which includes taking a picture, performing an experiment and performing complex calculations on data. The information collected may be *complementary* (e.g., different pictures of the same area can assist in building a 3D model), or they may be *redundant* (e.g., computing the same result wasting valuable time). Both complementary and redundant information present a problem: the global utility function is no longer

additive over the agents. When experiments provide redundant information there is little additional value to completing both so the global value is subadditive. When experiments are complementary, completing just one may have little value so the global value is superadditive. I am generally not looking at temporal constraints between agents or the information they collect, however, some temporal constraints can be captured by my model (see Section 3.3.3).

I motivate this class of problems with two examples. The first example is the problem of controlling the operation of multiple planetary exploration rovers, such as the ones used by NASA to explore the surface of Mars [75]. Periodically, the rovers are in communication with a ground control center. During that time, the rovers transmit the scientific data they have collected and receive a new mission for the next period. Each rover has its own area to explore, and a mission consists of a set of sites at which a rover could take pictures, conduct experiments, and in general collect data. Some of these sites are overlapping sites (lie on the border between two or more rovers' area) and multiple rovers can visit them.

The rovers must operate autonomously (without communication between them) until communication with the control center is feasible. Because the rovers have limited resources (computing power, electricity, memory, and time) and because there is uncertainty about the amount of resources that are consumed at each site, a rover may not be able to complete all of its objectives. The overall goal is to maximize the value of the information received by ground control. However, this is not the same as maximizing the value of the information collected by each rover individually because two rovers performing experiments at overlapping sites produce redundant information. Zilberstein et al. [81] shows how to model and solve the single rover control problem by creating a corresponding MDP.

My second motivating example is UAVs (unmanned aerial vehicles) performing reconnaissance in a combat environment. Here, communication between the UAVs

is too costly because it would reveal their location to the enemy. The UAVs are sent out to collect information from a number of locations and then to return to safety. The information sought by a UAV may depend on other information that it has collected. For example, if it discovers something interesting at one location it may spend additional time there collecting more detailed information and choose to skip other locations.

Independence between the local problems holds true in many domains, and in others it serves as an approximation. In the Mars rover example the rovers are operating in distinctly separate areas. For example, NASA's Mars Exploration Rover mission landed Spirit and Opportunity on separate sides of the planet[1]. An "overlapping site" could consist of similar geographic structures that lie on opposite sides of the planet. Given the rovers distance from each other there is no possibility of interaction of the local decision problems other than through the value of the information they collect, nor is direct communication feasible. The rover example used throughout this paper is an abstraction of this idea.

The UAV example illustrates a different type of situation where the local problems are independent except for the reward. Here the agents are highly mobile, but the level of abstraction used to represent this planning problem allows them to coexist in the same airspace. Essentially, one square of the grid world represents a mile of airspace so there is no problem with many UAVs traveling through it simultaneously, and the actions of one agent will not interfere with those of another agent.

The Transition Independent DEC-MDP is defined by the general class of factored n-agent DEC-MDPs that exhibit transition and observation independence, and local full observability, but not reward independence. Instead, the reward function is divided into two components. The first component is a set of local reward functions

[1]http://marsrovers.jpl.nasa.gov/home/

that are reward independent. The second component is a reward signal the system receives (not any individual agent) that depends on the actions of multiple agents and is therefore not reward independent. It is defined by the joint reward structure and is captured in the global value function being maximized.

The types of problems that I am looking to solve are those in which the agents are not tightly integrated through the reward function. I take advantage of that by using a different representation of the reward, called the Joint Reward Structure. Intuitively, the difference between the general reward function and the joint reward structure is similar to the difference between common representations for matrices and sparse matrices. In addition, this new representation explicitly illustrates how the agents are interacting.

3.2.1 Joint Reward Structure

I now introduce further structure into the global reward function. To define it, I need to introduce the notion of an occurrence of an event during the execution of a local policy.

Definition 7 *A* **history** *for agent i, $\Phi_i = [\hat{s}_i^0, a_i^0, \hat{s}_i^1, a_i^1, ...]$ is a valid execution sequence that records all of the local states and actions for one agent, beginning with the local starting state for that agent, $\hat{s}_i^0$.*

Definition 8 *A* **primitive event**, *$e = (\hat{s}_i, a_i, \hat{s}_i)$ is a tuple that includes a local state, an action, and an outcome state. An* **event** *$E = \{e_1, e_2, ..., e_h\}$ is a set of primitive events.*

Definition 9 *A primitive event $e = (\hat{s}_i, a_i, \hat{s}_i')$* **occurs** *in history Φ_i, denoted $\Phi_i \models e$ iff the tuple $(\hat{s}_i, a_i, \hat{s}_i')$ appears as a sub-sequence of Φ_i. An event $E = \{e_1, e_2, ..., e_h\}$* **occurs** *in history Φ_i, denoted $\Phi_i \models E$ iff*

$$\exists e \in E \ : \ \Phi_i \models e.$$

Events are used to capture the fact that an agent accomplished some task. In some cases a single local state may be sufficient to signify the completion of a task. But because of the uncertainty in the domain and because tasks could be accomplished in many different ways, I generally need a set of primitive events to capture the completion of a task.

In the rover example, the events I want to capture represent an agent collecting data at a particular site. This event will take many primitive events to describe because time is part of the local state. For example, the event *Collect Data at Site 4* can be accomplished by the primitive event where the agent executes the *collect* action at site 4 starting at time 6 (and ending at any possible next state). It can also be accomplished by a similar primitive event where the agent starts collecting at time 8 instead of time 6. More formally, the event can be described by the following set of primitive events:

$$\{\ (\langle l,t\rangle, a, \langle l',t'\rangle)\ \mid \langle l,t\rangle = \langle 4,*\rangle,\ a = collect,\ \langle l',t'\rangle = \langle *, < t\rangle\ \}\,,$$

where the local state is $\langle l,t\rangle$; l is the current location and t is the time left in the day.

Definition 10 *A primitive event is said to be* **proper** *if it can occur at most once in each possible history of a given MDP. That is:*

$$\forall \Phi_i = \Phi_i^1 e \Phi_i^2\ :\ \neg(\Phi_i^1 \models e) \wedge \neg(\Phi_i^2 \models e)$$

Definition 11 *An event $E = \{e_1, e_2, ..., e_h\}$ is said to be* **proper** *if it consists of mutually exclusive proper primitive events with respect to some given MDP. Mutually exclusive means:*

$$\forall \Phi_i\ \neg\exists x \neq y\ :\ e_x \in E\ \wedge\ e_y \in E\ \wedge\ \Phi_i \models e_x\ \wedge\ \Phi_i \models e_y$$

The event *Collect Data at Site 4* described earlier can easily be turned into a proper event. First, because time is part of the local state, no single primitive event could happen more than once — time only flows in one direction. Time is not a necessary condition, but in this example it suffices. Second, it is not difficult to restrict a rover from collecting data more than once at a particular site, either through additional bits in the state or through a fixed acyclic ordering for visiting the sites built into the structure of the MDP.

I limit the discussion in this paper to proper events. They are sufficiently expressive for the rover domain and for the other applications I consider, while simplifying the discussion. Later I show how some non-proper events can be modeled in this framework.

The joint reward structure can be viewed as a list of x constraints between the agents that describe how interactions between their local policies affect the global value of the system. A particular constraint k exists between some subset of agents G_k, where $|G_k| \geq 2$. The semantics of constraint k, $(E_k^{g_{k1}}, E_k^{g_{k2}} \ldots E_k^{g_{k|G_k|}}, c_k)$, is that each agent involved must satisfy its part of the constraint for the constraint to be satisfied. Agent g_{ki} satisfies its part of constraint k if event $E_k^{g_{ki}}$ occurs in that agent's history $\Phi_{g_{ki}}$. If each agent in a constraint does this, then the system would receive an additional c_k reward. There is no restriction on the number of the constraints that can be satisfied.

Definition 12 *Let Φ_1 through Φ_n be histories for n agents. Let G_1 through G_x be subsets of the n agents. Each set G_k contains $|G_k|$ agents g_{k1} through $g_{k|G_k|}$. A* **joint reward structure**

$$\rho = [(E_1^{g_{1,1}}, E_1^{g_{1,2}} \ldots E_1^{g_{1|G_1|}}, c_1), ..., (E_x^{g_{x1}}, E_x^{g_{x2}} \ldots E_x^{g_{x|G_x|}}, c_x)],$$

specifies the reward (or penalty) c_k that is added to the global value function if $\Phi_{g_{k1}} \models E_k^{g_{k1}}$ and $\Phi_{g_{k2}} \models E_k^{g_{k2}}$ and ... and $\Phi_{g_{k|G_k|}} \models E_k^{g_{k|G_k|}}$.

This section focuses on factored DEC-MDPs that are transition independent, observation independent with full local observability and whose global reward function R is composed of a local reward function for each agent R_1 through R_n plus a joint reward structure ρ. This allows us to define n *underlying* MDPs, $\langle S_i \times S_0, A_i, P_i, R_i \rangle$ even though the problem is not reward independent. These processes are indeed Markov, meaning that the local policies are mappings from local states to actions instead of histories to actions as with DEC-MDPs. A local state was proved to be a sufficient statistic for the history of observations sensed by an agent controlling a decentralized MDP with independent transitions and observations [28]. Therefore, the optimal local policy of such an agent can be expressed as a mapping from local states to actions.

Given a local policy, π_i, the probability that a particular primitive event $e = (\hat{s}_i, a_i, \hat{s}'_i)$ will occur during any execution of π_i, denoted $P(e|\pi_i)$, can be expressed as:

$$P(e|\pi_i) = \phi \sum_{v=0}^{\infty} P_v(\hat{s}_i|\pi_i) P_i(\hat{s}'_i|\hat{s}_i, a_i), \ \phi = \begin{cases} 1 & \pi_i(\hat{s}_i) = a_i \\ 0 & otherwise \end{cases}$$

where $P_v(\hat{s}_i|\pi_i)$ is the probability of being in state $\hat{s}_i$ at time step v. $P_v(\hat{s}_i|\pi_i)$ can be easily computed for a given MDP from its transition model, and $P_i(\hat{s}'_i|\hat{s}_i, a_i)$ is simply the transition probability. ϕ sets the probability to zero if the primitive event e is inconsistent with the policy π_i. Similarly, the probability that a proper event $E = \{e_1, e_2, ..., e_m\}$ will occur is:

$$P(E|\pi_i) = \sum_{e \in E} P(e|\pi_i).$$

I can sum the probabilities over the primitive events in E because they are mutually exclusive.

Definition 13 *Given a joint policy* $(\pi_1 ... \pi_n)$ *and a joint reward structure* ρ*, the* **joint value** *is:*

$$JV(\rho|\pi_1 ... \pi_n) = \sum_{k=1}^{|\rho|} c_k \prod_{i=1}^{|G_k|} P(E_k^{g_{ki}}|\pi_{g_{ki}})$$

Definition 14 *The* **global value function** *of a transition independent decentralized MDP with respect to a joint policy* $(\pi_1 ... \pi_n)$ *is:*

$$GV(\pi_1 ... \pi_n) = \sum_{i=1}^{n} V_{\pi_i}(\hat{s}_i^0) + JV(\rho|\pi_1 ... \pi_n)$$

where $V_{\pi_i}(\hat{s}_i^0)$ *is the standard value of the underlying MDP for agent* i *at starting state* $\hat{s}_i^0$ *given policy* π_i*.*

While $V_\pi(s)$ is generally interpreted to be the value of state s given policy π, I will sometimes refer to it as the value of π because I am only interested in the value of the initial state given π. Similarly, the global value function GV is defined over policies and not states because the interest is in the expected value of the starting state only. The goal is to find a joint policy that maximizes the global value function.

Definition 15 *An* **optimal joint policy***, denoted* $(\pi_1 ... \pi_n)^*$*, is a set of local policies that maximize the global value function, that is:*

$$(\pi_1 ... \pi_n)^* = \text{argmax}_{\pi'_1 ... \pi'_n} GV(\pi'_1 ... \pi'_n).$$

To summarize, a problem in my transition independent decentralized MDP framework is defined by n underlying MDPs, $\langle S_1 \times S_0, A_1, P_1, R_1 \rangle$ through $\langle S_n \times S_0, A_n, P_n, R_n \rangle$, and a joint reward structure ρ.

3.3 Expressiveness of the Model

Transition independent DEC-MDPs with a joint reward structure may seem to represent a small set of domains, but it turns out that the class of problems I address is quite general. The joint reward structure ρ is just another way of representing the general reward function over all of the agents, $R((s_0 \ldots s_n)(a_1 \ldots a_n)(s'_0 \ldots s'_n))$. How efficient a representation the joint reward structure is of the general reward function depends on the level of independence in the reward function. Intuitively it is similar to the difference between standard matrix representations and the representations for sparse matrices.

Often if a problem meets the independence and observability requirements but does not obviously have a properly structured reward function, it can be represented by adding extra information (in the form of a bit) to the local state or by modifying the state in some way. In many cases these bits are already necessary to represent the problem as a DEC-MDP and do not cause an increase in the size of the state space to represent in the joint reward structure. Since MDPs suffer from the curse of dimensionality the size of the state space grows exponentially with the number of bits added.

This section will discuss several properties of the joint reward structure and how problems can be modified to fit. It will also show how some temporal constraints can be represented by this class of problems.

3.3.1 Mutual Exclusion

One property that some problems do not naturally adhere to is the mutual exclusion among primitive events. The mutual exclusion property guarantees that *at most one* primitive event within an event set can occur. This section presents three alternatives. For each of them, the bits mentioned are already present in the DEC-MDP representation of the problem.

At least one primitive event – Suppose that multiple primitive events within an event set can occur and that an additional reward is added when *at least one* of them does occur. In this case the state can be augmented with one bit, initially set to 0. When a primitive event in the set occurs, the bit is set to 1. If I redefine each primitive event to include the bit switching from 0 to 1, then the event set is now proper because the bit is toggled from 0 to 1 only on the first primitive event encountered.

All primitive events – Suppose that an event is composed of x primitive events, *all* of which must occur to trigger the extra reward. In this case, each local state must be augmented with x bits (one per primitive event), which start at 0 and are toggled to 1 when the corresponding primitive event occurs (ordering constraints among primitive events could reduce the number of bits necessary). The new event set occurs when the last bit still 0 is flipped.

Counting occurrences – Suppose that an event is based on a primitive event (or another event set) repeating at least x times. Here, the local state can be augmented with $\log x$ bits to be used as a counter. The extra reward is triggered when the desired number of occurrences is reached, at which point the counting stops.

3.3.2 All Events

Another property of the joint reward structure is that every event in a constraint must occur for the system to receive the extra reward. Other useful constraints that one may wish to express may involve the occurrence of *at most x events*, *exactly x events*, or *at least x events*. These can be represented by creating new constraints, one for each possible way of solving the existing constraint. For example, suppose there is a constraint between three agents (events E_1, E_2, E_3), and the constraint is satisfied (extra reward received) if exactly two of the three events occur. This constraint can

be represented by three new constraints with the semantics of *all events must occur*: $E_1E_2\neg E_3$, $E_1\neg E_2E_3$, $\neg E_1E_2E_3$.

The event $\neg E_x$ represents the original event E_x not occurring. This event can be represented by adding two additional bits to the state space. One bit represents whether the original event has occurred and the other represents whether the original event could occur in the future. The new event $\neg E_x$ occurs when the original event has not occurred and the cannot-occur-in-the-future bit flips to *true*.

There is also the possibility that the algorithm presented in Section 4 could be modified to handle these other constraints more directly and efficiently. This remains the topic of future work.

3.3.3 Temporal Constraints

So far I have focused on global reward structures that do not impose any temporal constraints on the agents. Other types of constraints are soft temporal constraints like *facilitates* and *hinders* [15]. A *facilitates* constraint between activities A and B means that if A is finished before B is started, then execution of B is somehow facilitated. Facilitation can take many forms like reduced consumption of resources, less time to execute, higher reward received. The *hinders* constraint is similar, but instead involves making the execution of B more costly.

Soft temporal constraints that just involve a change in the expected reward can be represented in this framework if time is enumerated as part of the local states. For example, suppose that event E^1 in agent 1 facilitates/hinders event E^2 in agent 2, that is, the occurrence of E^1 *before* E^2 leads to an extra reward c. To properly define ρ, I need to create new events E^1_i and E^2_i for all $1 \leq i \leq maxTime$, and $c_i = c$. E^1_i represents E^1 occurring at time i and E^2_i represents E^2 occurring after i. If both E^1_i and E^2_i occur, then reward c is gained because E^1 happened before E^2. ρ now becomes $[(E^1_1, E^2_1, c), (E^1_2, E^2_2, c) \ldots (E^1_{maxTime}, E^2_{maxTime}, c)]$.

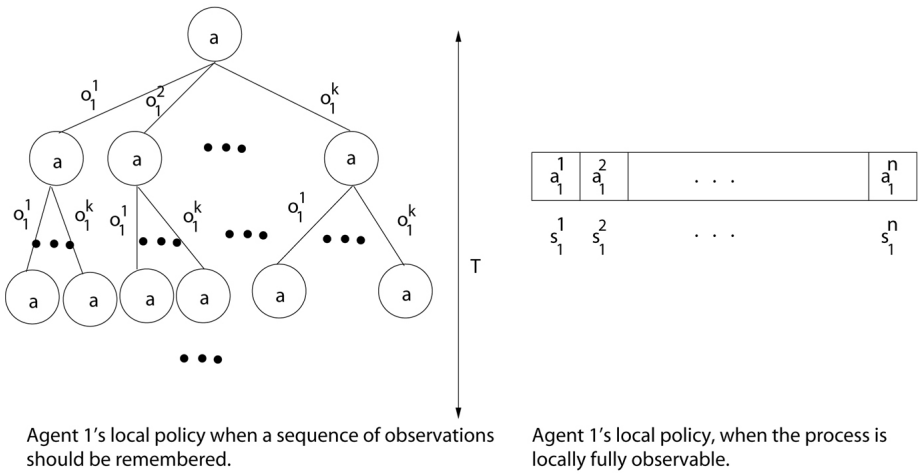


Figure 3.1. Exponential-sized vs. Polynomial-sized Policies.

There are a couple of important limitations for temporal constraints. First is that not all types of temporal constraints can be handled. The temporal constraints are limited to affecting the reward and not the duration (or any other characteristic of the facilitated/hindered event that the agent observes) because that would violate the transition independence assumption. Second, representing a temporal constraint is not very practical because one temporal constraint gets represented as $maxTime$ non-temporal constraints in ρ. A more compact representation of temporal constraints remains the subject of future research.

To summarize, there is a wide range of practical problems that can be represented within my framework. Non-temporal constraints have a more natural, compact representation, but some temporal constraints can also be captured.

3.4 Complexity Analysis

It is known that solving optimally a general DEC-MDP is NEXP-complete [6]. The leftmost diagram in Figure 3.1 illustrates this result by showing that a local policy of a general DEC-MDP is exponential in the size of the observations. Each

local policy is a mapping from sequences of observations to actions. Therefore, there are $|A_i|^{|\Omega|^T}$ policies for agent i, where T is the horizon. Each agent j needs to build a belief-state MDP for each one of agent i's local policies. The number of states in this MDP is exponential in the size of Ω_j. This results in an algorithm that will solve optimally a DEC-MDP problem using a complete search algorithm in double exponential time[2].

The class of DEC-MDPs studied in this paper is characterized by having independent transitions and observations and being locally fully observable. Because the current local view of a single agent is a sufficient statistic for the past history of observations of agent i (see [28]) a local policy for agent i is a mapping from its local states to its possible actions, i.e., $\pi_i : S_i \times S_0 \rightarrow A_i$ (see the rightmost diagram in Figure 3.1). This results in agents' policies being of size *polynomial* in the number of local states (as opposed to exponential in the size of the observation set as in the general case). Theorem 1 (adapted from [28]) shows that the complexity of solving optimally such a DEC-MDP is easier than solving the general case.

Theorem 1 *Deciding a DEC-MDP with independent transitions and observations, local full observability, and joint reward structure ρ is NP-complete.*

Proof. Since the current local state of agent i is a sufficient statistic, a local policy for that agent is of size polynomial in $|S_i \times S_0|$. There are $|A_i|^{|S_i \times S_0|}$ policies (mappings from $S_i \times S_0$ to A_i). The number of states in each agent i's belief-state MDP is polynomial in $|S_i \times S_0|$ (for a fixed and known policy for the other agents). Evaluating one such local policy can be done in polynomial time (by running dynamic programming on the belief-state MDP), but there are exponentially many such policies for which

[2]Assuming that the finite horizon T is similar in size to the number of global states of the system $|S|$. This assumption is necessary for the complexity of the general DEC-MDP, but not for the complexity of the class of problems dealt with in this paper.

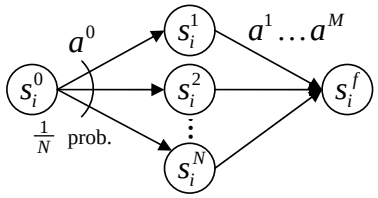


Figure 3.2. DTEAM reduces to an MDP of this form.

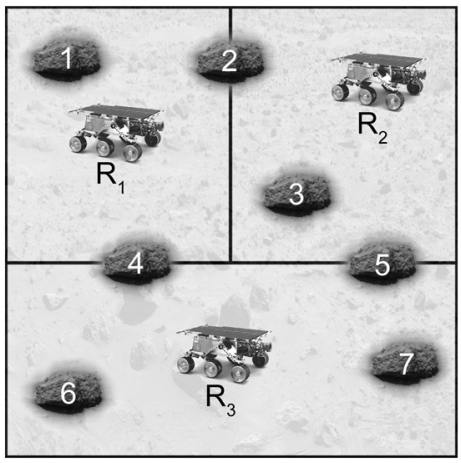
this should be done. Therefore, the upper bound for the decision problem stated in this theorem is NP.

To prove the lower bound I will reduce the NP-complete problem DTEAM [54, 55] to this problem for two agents, which is sufficient for a lower bound. DTEAM is a single-step discrete team decision problem. There are two agents. Agent i, $i = 1, 2$, observes a random integer k_i, $1 \leq k_i \leq N$, and takes an action $\gamma_i(k_i) \in \{1, \ldots, M\}$. Their actions incur cost $c(k_1, k_2, \gamma_1(k_1), \gamma_2(k_2))$. The problem is to find policies γ_1 and γ_2 that minimize the expected cost:

$$\sum_{k_1=1}^{N} \sum_{k_2=1}^{N} c(k_1, k_2, \gamma_1(k_1), \gamma_2(k_2)).$$

The reduction is quite straightforward. The local MDP for agent i consists of an initial state ($\hat{s}_i^0$), N intermediate states ($\hat{s}_i^1 \ldots \hat{s}_i^N$), and a single final state ($\hat{s}_i^f$), see Figure 3.2. The observation is the current state. There is one action (a^0) available in the initial state, and M actions ($a^1 \ldots a^M$) in the intermediate states. The local rewards are always 0. The joint reward ρ includes an entry for every combination of intermediate state and action taken by the agents:

$$\begin{aligned} \rho = [\quad & (\{(\hat{s}_1^1, a^1, \hat{s}_1^f)\}, \{(\hat{s}_2^1, a^1, \hat{s}_2^f)\}, -c(1,1,1,1)), \\ & (\{(\hat{s}_1^1, a^1, \hat{s}_1^f)\}, \{(\hat{s}_2^1, a^2, \hat{s}_2^f)\}, -c(1,1,1,2)), \\ & \ldots \end{aligned}$$



Site x	$t(x)$	$r(x)$	Total Value
1	10	6	$r(1)$
2	**12**	**5**	$\mathbf{3}r\mathbf{(2)}$
3	8	7	$r(3)$
4	**11**	**9**	$r\mathbf{(4)}$
5	**13**	**8**	$\mathbf{1.5}r\mathbf{(5)}$
6	9	6	$r(6)$
7	14	7	$r(7)$

Figure 3.3. Illustration of the Mars rover example. Each of the rovers can collect data from sites within their region. Sites 2, 4 and 5 fall on the boundaries and can be visited by two rovers each. The table entries in bold are the shared tasks.

$$(\{(\hat{s}_1^N, a^M, \hat{s}_1^f)\}, \{(\hat{s}_2^N, a^M, \hat{s}_2^f)\}, -c(N, M, N, M)) \ ]$$

Finding the joint policy that maximizes the expected reward will be the same policy that minimizes the expected cost in the original problem. □

3.5 Example

This example is an instance of the rover problem presented earlier. There are three rovers, each with their own region of space to explore, as shown in Figure 3.3. Each site x has a value for the data collected there, $r(x)$, and a time to collect the data, $t(x)$. The local state for rover 1 is composed of the current location l, the time left in the day t, and the data to be collected $d_1,\ d_2,\ d_4$ (0 if not collected, 1 if collected). The available actions are *collect* and *go to site x*. The transition on a *collect* action, when at site x, is from $\langle l = x, t, d_a, d_b, d_x = 0\rangle$ to $\langle l = x, t - t(x), d_a, d_b, d_x = 1\rangle$. The reward received is $r(x)$ if $t - t(x) \geq 0$, otherwise 0. The transition on a *go to site x* action when at site y is from $\langle l = y, t, d_1, d_2, d_4\rangle$ to $\langle l = x, t - t(y, x), d_1, d_2, d_4\rangle$ and

a reward of 0. $t(y,x)$ is the time it takes to move from site y to site x. The joint reward structure is:

$$\rho = \Big[(E_2^1, E_2^2, r(2)),\ (E_4^1, E_4^3, -r(4)),\ (E_5^2, E_5^3, -0.5r(5)) \Big].$$

Event E_x^i is an event that represents agent i collecting data at site x. This event is composed of primitive events of the form:

$$\{(\hat{s}, a, \hat{s}') | \hat{s} = \langle l = x, t, d_a, d_b, d_x = 0 \rangle,\ a = collect,\ \hat{s}' = \langle l = x, t - t(x), d_a, d_b, d_x = 1 \rangle\}.$$

Each agent locally receives $r(x)$ when it collects at site x. The Total Value column in Figure 3.3 lists the total reward received by the system for each site. For sites $x = \{2, 4, 5\}$ this is the reward received after both agents collect there. The difference between the total value and $2r(x)$ is put into ρ as the extra reward the system receives for site x for those sites that overlap two agents. For example, for $x = 5$, $1.5r(5) - 2r(5) = -0.5r(5)$. When rover 1 collects data at site 5 it receives $r(5)$. When rover 2 collects data there it receives $r(5)$. The system also receives a penalty of $-0.5r(5)$ because both rovers collected that data. The net reward received by the system is $1.5r(5)$.

CHAPTER 4

COVERAGE SET ALGORITHM

This section presents a novel algorithm to find optimal joint policies for transition independent decentralized MDPs. To the best of my knowledge this is the first algorithm to tractably and optimally solve a significant subclass of DEC-MDPs. I have also applied it to a different class of problems in which the agents were reward independent but not transition independent [3], which demonstrates that it is not limited to the class of problems described in this paper. Most other work on distributed problems have used approximate solutions, such as heuristic policy search and gradient descent e.g., [58], or assumed complete communication at every step or when the optimal action is ambiguous e.g.,[9, 76]. The former are not guaranteed to converge on the optimal solution and the latter are not practical when communication is not possible or very expensive.

While the formal problem description deals with n agents, for clarity the description of the algorithm will only deal with two agents (i and j). Section 3.5 discusses the n-agent extension of the algorithm.

The algorithm is divided up into three major parts:

1. Create *augmented MDPs*. An augmented MDP represents one agent's underlying MDP with an augmented reward function.

2. Find the *optimal coverage set* for the augmented MDPs, which is the set of all optimal policies for one agent that correspond to *any* possible policy of the other agent. As I show below, this set can be represented compactly.

3. Find for each policy in the optimal coverage set the corresponding best policy for the other agent. Return the best among this set of joint policies, which is the optimal joint policy.

Pseudo-code for the coverage set algorithm is shown in Figure 4.1. The main function, CSA, takes a transition independent DEC-MDP (as described in Section 2) as input, and returns an optimal joint policy. The remaining functions are described in detail below.

4.1 Creating Augmented MDPs

The first part of the algorithm is to create the augmented MDPs, which are essentially the underlying MDPs for each agent with an augmented reward function. The new reward is calculated from the original reward, the joint reward structure and the policy of the other agent. The influence of the other agent's policy on the augmented MDP can be captured by a vector of probabilities, which is a point in the parameter space.

An augmented MDP has three properties. First, an augmented MDP is an MDP defined over the states and actions for agent i given a policy π_j for agent j. The transition function and reward function can depend on π_j.

Second, the augmented MDP maximizes the global value of the system for a given policy for agent j. This means that the policy for agent j can be evaluated independently of the policy agent i adopts, and the global value is equal to the independent expected value of agent j's policy plus the expected value of the augmented MDP given both policies: $GV(\pi_i, \pi_j) = V_{\pi_j}(\hat{s}_j^0) + V_{\pi_i}^{\pi_j}(\hat{s}_i^0)$.

The third and final property of the augmented MDP is that the value function of the augmented MDP, $V_{\pi_i}^{\pi_j}(\hat{s}_i^0)$, must be a linear combination of a set of parameters computed from the policy for agent j. Note that any nonlinear function can be made linear by adding additional parameters.

```
function CSA(MDP_1, MDP_2, ρ)
    returns the optimal joint policy
    inputs:   MDP_1, underlying MDP for agent 1
              MDP_2, underlying MDP for agent 2
              ρ, joint reward structure

    optset ← COVERAGE-SET(MDP_1, ρ)
    value ← −∞
    jointpolicy ← {}
    /* find best joint optimal policy */
    for each policy_1 in optset
        policy_2 ← SOLVE(AUGMENT(MDP_2, policy_1, ρ))
        v ← GV({policy_1, policy_2}, MDP_1, MDP_2, ρ)
        if (v > value)
            then value ← v
                 jointpolicy ← {policy_1, policy_2}
    return jointpolicy

function COVERAGE-SET(MDP, ρ)
    returns set of all optimal policies with respect to ρ
    inputs:   MDP, underlying MDP
              ρ, joint reward structure

    planes ← {} /* planes are equivalent to policies */
    points ← {}
    /* initialize boundaries of parameter space */
    for n ← 1 to |ρ|
        boundaries ← boundaries ∪ {x_n = 0, x_n = 1}
    /* loop until no new optimal policies found */
    do
        newplanes ← {}
        points ← INTERSECT(planes ∪ boundaries)
        /* get optimal plane at each point */
        for each point in points
            plane ← SOLVE(AUGMENT(MDP, point, ρ)
            if plane not in planes
                then newplanes ← newplanes ∪ {plane}
        planes ← planes ∪ newplanes
    while |newplanes| > 0
    return planes
```

Figure 4.1. Coverage Set Algorithm

If an augmented MDP can be created for a problem, then the CSA can find the optimal joint policy for that problem.

Definition 16 *The* **parameter space** *is a $|\rho|$ dimensional space where each dimension has a range of $[0, 1]$. Each policy π_j has a corresponding point in the parameter space, $\bar{x}_{\pi_j}$, which measures the probabilities that each one of the events in ρ will occur when agent j follows policy π_j:*

$$\bar{x}_{\pi_j} = [P(E_1^j|\pi_j), P(E_2^j|\pi_j), ..., P(E_{|\rho|}^j|\pi_j)].$$

Given a point in the parameter space, $\bar{x}_{\pi_j}$, agent i can define a decision problem that accurately represents the global value instead of just its local value. It can do this because both the joint reward structure and agent j's policy are fixed. This new decision problem is defined as an augmented MDP.

Definition 17 *An* **augmented MDP**, *$MDP_i^{\bar{x}_{\pi_j}}$, is defined as $\langle S_i \times S_0, A_i, P_i, R_i', \bar{x}_{\pi_j}, \rho \rangle$, where $\bar{x}_{\pi_j}$ is a point in the parameter space computed from the policy for agent j, ρ is the joint reward structure and R_i' is:*

$$R_i'(e) = R_i(e) + \sum_{k=1}^{|\rho|} \phi_k P(E_k^j|\pi_j) c_k, \;\; \phi_k = \begin{cases} 1 & e \in E_k^i \\ 0 & otherwise \end{cases}$$

Note that $e = (\hat{s}, a, \hat{s}')$ so $R(e)$ is the same as $R(\hat{s}, a, \hat{s}')$.

An intuitive way to think about the augmented MDPs is in terms of a credit assignment problem. The system receives an extra reward if an event occurs for both agents. However, instead of giving that reward to the system, the reward could be divided up between the agents. An augmented MDP for agent i represents giving all of the credit (extra expected reward) to agent i.

Theorem 2 *The value of a policy π_i over $MDP_i^{\bar{x}_{\pi_j}}$ is:*

$$V_{\pi_i}^{\bar{x}_{\pi_j}}(\hat{s}_i^0) = V_{\pi_i}(\hat{s}_i^0) + JV(\rho|\pi_i, \pi_j).$$

Proof. The value of an MDP given a policy can be calculated by summing over all time steps t and all events e, the probability of seeing e after exactly t steps, times the reward gained from e:

$$\begin{aligned}
V_{\pi_i}^{\bar{x}_{\pi_j}}(\hat{s}_i^0) &= \sum_{v=0}^{\infty} \sum_{e} P_v(e|\pi_i) R'(e) \\
&= \sum_{v=0}^{\infty} \sum_{e} P_v(e|\pi_i) \left(R_i(e) + \sum_{k=1}^{|\rho|} \phi_k P(E_k^j|\pi_j) c_k \right) \\
&= \sum_{v=0}^{\infty} \sum_{e} P_v(e|\pi_i) R_i(e) + \\
&\quad \sum_{v=0}^{\infty} \sum_{e} \sum_{k=1}^{|\rho|} \phi_k P_v(e|\pi_i) P(E_k^j|\pi_j) c_k \\
&= V_{\pi_i}(\hat{s}_i^0) + \sum_{k=1}^{|\rho|} P(E_k^j|\pi_j) c_k \sum_{v=0}^{\infty} \sum_{e \in E_k^i} P_v(e|\pi_i) \\
&= V_{\pi_i}(\hat{s}_i^0) + \sum_{k=1}^{|\rho|} P(E_k^j|\pi_j) P(E_k^i|\pi_i) c_k \\
&= V_{\pi_i}(\hat{s}_i^0) + JV(\rho|\pi_i, \pi_j).
\end{aligned}$$

□

The function AUGMENT in Figure 4.1 takes an MDP, a policy and a joint reward structure and returns an augmented MDP according to Definition 17.

Since the joint value function has been neatly folded into the value of an augmented MDP, the global value function can be rewritten as:

$$GV(\pi_1, \pi_2) = V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) + V_{\pi_2}(\hat{s}_2^0) = V_{\pi_1}(\hat{s}_1^0) + V_{\pi_2}^{\bar{x}_{\pi_1}}(\hat{s}_2^0) \tag{4.1}$$

From this it is easy to show that an optimal joint policy is a Nash equilibrium.

Proposition 1 *An optimal joint policy* $(\pi_1, \pi_2)^*$ *is a Nash equilibrium over the augmented MDPs:*

$$\begin{aligned} V_{\pi_1}^{\bar{x}_{\pi_2}} &= \max_{\pi_1'} V_{\pi_1'}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) \\ V_{\pi_2}^{\bar{x}_{\pi_1}} &= \max_{\pi_2'} V_{\pi_2'}^{\bar{x}_{\pi_1}}(\hat{s}_2^0). \end{aligned}$$

Proof. Assume $\exists \pi_1' \neq \pi_1 : V_{\pi_1'}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) > V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0)$.
From Equation 4.1:

$$\begin{aligned} GV(\pi_1', \pi_2) &= V_{\pi_1'}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) + V_{\pi_2}(\hat{s}_2^0) \\ GV(\pi_1, \pi_2) &= V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) + V_{\pi_2}(\hat{s}_2^0) \end{aligned}$$

Therefore, $GV(\pi_1', \pi_2) > GV(\pi_1, \pi_2)$. This contradicts $(\pi_1, \pi_2)^*$. By symmetry, I can show the same for π_2. Therefore, the optimal joint policy is a Nash equilibrium over augmented MDPs. □

This naturally suggests an iterative hill-climbing algorithm where the policy for one agent is fixed and the optimal policy for the other agent's augmented MDP is computed. Then the new policy is fixed and a new optimal policy for the first agent is computed. Repeating this process will converge upon a locally optimal solution, and with random restarts it becomes an attractive approximation algorithm. Nair et al. [51], and Shen et al. [63] study applications of this approximation algorithm to DEC-POMDPs.

The problem is that it provides no guarantees. No matter how long it is run, there is always the possibility that it will not find the optimal joint policy. The CSA circumvents this problem by finding *all* of the local maxima. In the problems I experimented with this is feasible because the number of local maxima is orders of

magnitude fewer than the number of policies (though a problem could probably be crafted in which this is not the case).

4.2 Finding the Optimal Coverage Set

An augmented MDP is defined over a point in the parameter space, which is a continuous space. This means that for both agents, there are an infinite number of augmented MDPs, however, only a finite number of them are potentially meaningful: the ones where the point in parameter space corresponds to a policy of the other agent. Additionally, most of these augmented MDPs have the same optimal policies, so the set of all optimal policies for all of the augmented MDPs for one agent is quite small. This set is what I call the optimal coverage set.

Definition 18 *The* **optimal coverage set**, O_i, *is the set of optimal policies for* $MDP_i^{\bar{x}}$ *given any point in parameter space,* $\bar{x}$*:*

$$O_i = \{\pi_i \mid \exists \bar{x}, \pi_i = \text{argmax}_{\pi_i'} V_{\pi_i'}^{\bar{x}}(\hat{s}_i^0)\}.$$

Another way to look at the optimal coverage set is to examine the geometric representation of a policy over the parameter space. The value of a policy π_i, given in Theorem 2, is a linear equation. If $|\bar{x}_{\pi_j}| = 1$, then the value function is a line in two dimensions. When $|\bar{x}_{\pi_j}| = 2$, the value function is a plane in three dimensions. In general, $|\bar{x}_{\pi_j}| = n$ and the value function is a hyperplane in $n + 1$ dimensions.

The optimal coverage set, then, is the set of hyperplanes that are highest in the $n+1$ dimension for all points in the parameter space (first n dimensions). The upper surface that I am interested in is a piecewise-linear and convex function formed by these hyperplanes. First I examine the one dimensional case (one constraint). Figure 4.2 shows a graph of the parameter space (x-axis) versus expected value of a policy. On this graph I plot the equation shown in Theorem 2 for particular policies. For

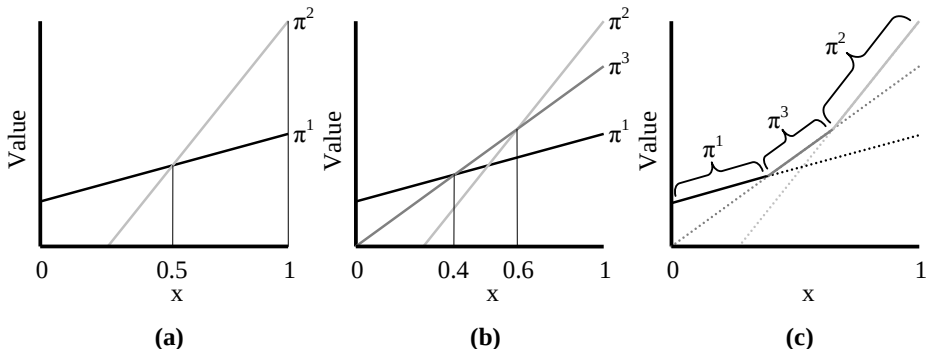


Figure 4.2. Search process in one dimension.

every point in the parameter space, the optimal policy of the augmented MDP that corresponds to that point is a policy in the optimal coverage set.

The algorithm starts by finding the optimal policies in the corners of the parameter space, which in this example are $x = 0.0$ and $x = 1.0$. This yields two optimal policies, π^1 for $x = 0.0$ and π^2 for $x = 1.0$, whose value functions are graphed in Figure 4.2 (a). Those two lines intersect at $x = 0.5$, which is then chosen as the next point. The optimal policy of that point, π^3, is added in Figure 4.2 (b). New points in the top surface are found by intersecting the new line with the previous lines, $x = 0.4$ and $x = 0.6$. If new policies are found at either of those points, those lines are added and new intersections selected. This repeats until all of the current intersection points have optimal policies represented on the graph. That set of policies form the optimal coverage set. In the example no new policies were found at $x = 0.4$ or $x = 0.6$ so Figure 4.2 (c) shows the piecewise-linear and convex surface I am searching for, and those three policies form the optimal coverage set for one agent.

The intersection points of the lines are key because that is where the optimal policy changes from one policy to a different policy. Intuitively, if a policy is optimal at $x = 0.6$ and $x = 1.0$ then that same policy is optimal between those two points because I am working with linear equations.

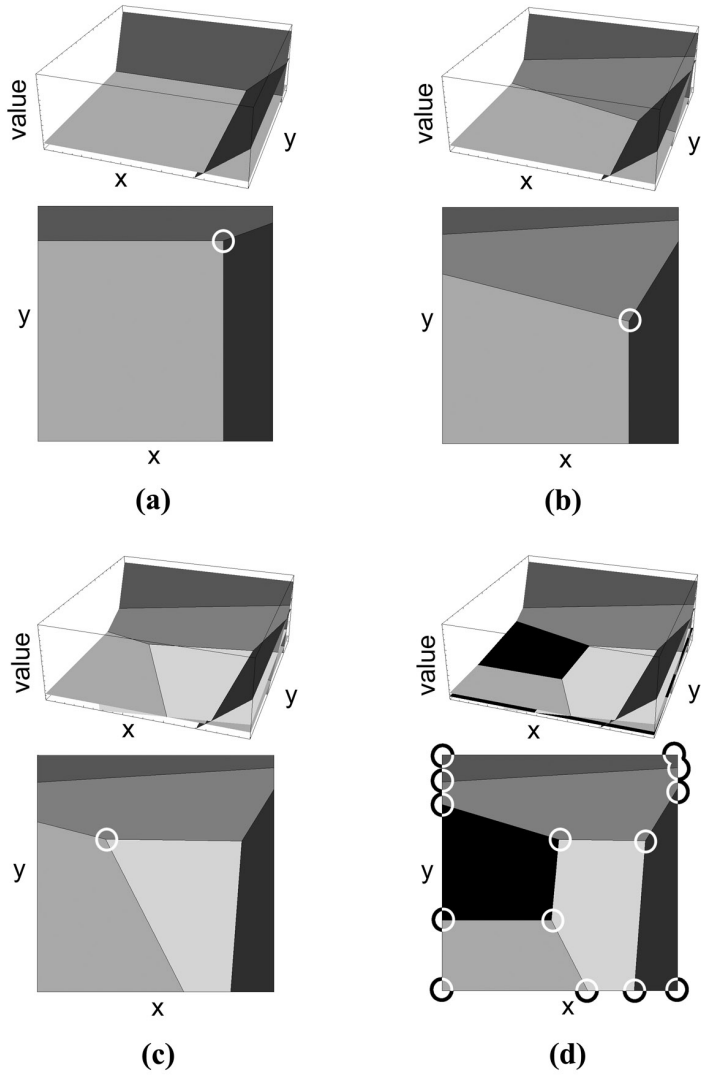


Figure 4.3. Intersecting Planes. (a) The first iteration checks the corners of the parameter space: $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, which yields three planes. In the second iteration one of the intersection points in the top surface is chosen (circled), and a new optimal policy is found and added in (b). This process repeats until all fourteen of the intersections on the top surface between known optimal policies (circled in (d)) have been checked and no new optimal policies found. The six optimal policies in (d) form the optimal coverage set based on these intersection points.

Figure 4.3 shows the same algorithm with a two dimensional parameter space (two constraints). Here I am looking at planes in three dimensions. Figure 4.3 (a) shows the three planes (optimal policies) found at the four corners of the parameter space.

The circled point is the next one selected and Figure 4.3 (b) shows the new plane found at that point. If all of the points circled in Figure 4.3 (d) have been checked and no new optimal policies found, then the algorithm terminates.

The next two theorems prove that the underlying idea for this step in the algorithm is correct. The idea is that if a set of points has the same optimal policy, then any point enclosed by those points also has that optimal policy because I am working with linear functions.

Theorem 3 *If two points $\bar{x}$ and $\bar{y}$ in n-dimensional parameter space have the same corresponding optimal policy π, then all points on the line segment between $\bar{x}$ and $\bar{y}$, $f(\alpha) = \bar{x} + \alpha(\bar{y} - \bar{x})$, $0 \leq \alpha \leq 1$, have optimal policy π.*

Proof. Let π be the optimal policy at $\bar{x}$ and $\bar{y}$, $\bar{z}$ be a point on the line between $\bar{x}$ and $\bar{y}$, and π' be the optimal policy at $\bar{z}$:

$$\pi = \text{argmax}_\pi V_\pi^{\bar{x}}(\hat{s}_i^0) = \text{argmax}_\pi V_\pi^{\bar{y}}(\hat{s}_i^0),$$

$$\bar{z} = f(\alpha_0),\ 0 < \alpha_0 < 1, \text{ and}$$

$$\pi' = \text{argmax}_{\pi'} V_{\pi'}^{\bar{z}}(\hat{s}_i^0).$$

Assume that at $\bar{z}$ the value of the optimal policy π' is strictly greater than the value of π, $V_{\pi'}^{\bar{z}}(\hat{s}_i^0) > V_\pi^{\bar{z}}(\hat{s}_i^0)$. I know $V_\pi^{\bar{x}}(\hat{s}_i^0) \geq V_{\pi'}^{\bar{x}}(\hat{s}_i^0)$ because π is optimal at $\bar{x}$. Since $V(\cdot)$ and $f(\cdot)$ are linear functions, I can calculate their value at $f(1) = \bar{y}$ by computing the unit slope.

$$\begin{aligned}
V_\pi^{\bar{y}}(\hat{s}_i^0) &= \frac{V_\pi^{\bar{z}}(\hat{s}_i^0) - V_\pi^{\bar{x}}(\hat{s}_i^0)}{\alpha_0} + V_\pi^{\bar{x}}(\hat{s}_i^0) \\
&= \frac{1}{\alpha_0} V_\pi^{\bar{z}}(\hat{s}_i^0) - \left(\frac{1-\alpha_0}{\alpha_0}\right) V_\pi^{\bar{x}}(\hat{s}_i^0) \\
&\leq \frac{1}{\alpha_0} V_\pi^{\bar{z}}(\hat{s}_i^0) - \left(\frac{1-\alpha_0}{\alpha_0}\right) V_{\pi'}^{\bar{x}}(\hat{s}_i^0) \\
&< \frac{1}{\alpha_0} V_{\pi'}^{\bar{z}}(\hat{s}_i^0) - \left(\frac{1-\alpha_0}{\alpha_0}\right) V_{\pi'}^{\bar{x}}(\hat{s}_i^0) \\
&= \frac{V_{\pi'}^{\bar{z}}(\hat{s}_i^0) - V_{\pi'}^{\bar{x}}(\hat{s}_i^0)}{\alpha_0} + V_{\pi'}^{\bar{x}}(\hat{s}_i^0)
\end{aligned}$$

$$< \quad V_{\pi'}^{\bar{y}}(\hat{s}_i^0)$$

This contradicts that π is optimal at $\bar{y}$, therefore $V_{\pi}^{\bar{z}}(\hat{s}_i^0) = V_{\pi'}^{\bar{z}}(\hat{s}_i^0)$ for any $\bar{z}$ between $\bar{x}$ and $\bar{y}$. □

A bounded polyhedron in n dimensions is composed of a set of faces, which are bounded polyhedra in $n-1$ dimensions. The corners of a bounded polyhedron are the points (polyhedra in 0 dimensions) that the polyhedron recursively reduces to. Theorem 3 shows that for a line segment (polyhedron in 1 dimension) with endpoints with the same optimal policy, every point along that line also has the same optimal policy. This can be inductively extended to higher dimensions because any point in the interior falls on a line segment with endpoints on the edges of the polyhedron.

Corollary 1 *Given a bounded polyhedron in n dimensions whose corners all have the same corresponding optimal policy π_i, any point on the surface or in the interior of that polyhedron also has optimal policy π_i.*

There is some similarity between this step of the algorithm and other algorithms that are trying to find a piecewise-linear and convex function, like dynamic programming for POMDPs. In the POMDP case the function represents the optimal actions over the belief states. In the coverage set algorithm it represents the optimal policies over the parameter space. The primary difference between these two is that here I can efficiently generate the optimal policy for a point in the parameter space, while with a POMDP there is no way to efficiently generate the optimal action for a given belief state. The pruning in dynamic programming for POMDPs is in response to that inefficiency and is not relevant to the coverage set algorithm. Kaelbling et al. [40] give a detailed description of a POMDP and accompanying algorithm.

The part of the algorithm discussed in this section is handled by the function COVERAGE-SET in Figure 4.1. It takes an MDP and a joint reward structure and

returns the optimal coverage set, based on Theorem 1. To illustrate how this works, I will step through a small example.

Consider an instance of the Mars 2-rover problem with just two elements in the joint reward structure: (E_1^1, E_1^2, c_1) and (E_2^1, E_2^2, c_2). The function CSA calls COVERAGE-SET on MDP_1 and ρ. The first thing that COVERAGE-SET does is to create the boundaries of the parameter space. These are the hyperplanes that enclose the parameter space. Since each dimension is a probability, it can range from 0 to 1, so in this case there are 4 boundary lines: $x_1 = 0$, $x_1 = 1$, $x_2 = 0$, $x_2 = 1$. The algorithm then loops until no new planes are found.

In each loop, INTERSECT is called on the set of known boundary and policy hyperplanes. INTERSECT takes a set of hyperplanes and returns a set of points that represent the intersections of those hyperplanes. The simple implementation would just return every intersection point, however many of those points are not useful — those that lie outside the parameter space or lie below a known optimal plane. For example, Figure 4.3(d) has six policy planes and the four boundaries of the parameter space. The total number of points is approximately 84, but only the 14 visible points are necessary to divide up the parameter space into the set of polygons.

After computing the set of points, the augmented MDP for each of those points is created and the optimal policy for each of those augmented MDPs is computed by SOLVE, which can use standard dynamic programming algorithms. The value of a policy and a point in parameter space is

$$\begin{aligned} V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) &= P(E_1^1|\pi_1)P(E_1^2|\pi_2)c_1 + \\ &\quad P(E_2^1|\pi_1)P(E_2^2|\pi_2)c_2 + V_{\pi_1}(\hat{s}_1^0). \end{aligned}$$

For a given π_1, the value function is a plane over the parameter space. The plane for each of the new optimal policies will either be equivalent (different policy but same value) or equal to a plane already in the coverage set, or it will be better than every

other plane in the coverage set at this point in parameter space. If it is the latter case, this new plane is added to the coverage set. If a complete iteration does not find any new planes, then the loop terminates and the current coverage set is returned.

4.3 Selecting the Optimal Joint Policy

Given an optimal coverage set for agent i (considering the joint reward ρ), finding the optimal joint policy is straightforward. From Proposition 1 and Definition 18 I know that one of the policies in the optimal coverage set is a part of an optimal joint policy. Therefore, finding the optimal joint policy reduces to policy search through the optimal coverage set. For each policy in agent i's optimal coverage set, create the corresponding augmented MDP for agent j and find its optimal policy. The optimal joint policy is the pair with the highest global value.

The function GV returns the global value as defined in Definition 14.

Theorem 4 *The coverage set algorithm always returns the optimal value.*

Proof. To prove that the coverage set algorithm always returns the optimal value, I show that the algorithm terminates, it finds the optimal coverage set, and then it returns the optimal joint policy.

1. **Termination** – Three of the four loops in this algorithm iterate over the elements in finite, unmodified sets. The fourth loop is the *do ... while* | *newplanes* |> 0. In every iteration, policies for the MDP are added to *newplanes* only if they have not been added in a previous iteration. Since the set of possible policies is finite, eventually there will be no policies to add and the loop will terminate.

2. **Optimal coverage set is found** – All the planes/policies in the returned set are derived by solving the corresponding MDP using dynamic programming

and are therefore optimal. All the relevant point intersections between the hyperplanes are found. This set of points divides the parameter space into a set of polyhedra. From Theorem 1 if no new optimal policies are found from those points, then the set of optimal policies is the optimal coverage set.

3. **The optimal joint policy is returned** – The set of joint policies created by taking an optimal coverage set and finding the corresponding optimal policy for the other agent includes all Nash equilibria. The algorithm returns the best of those equilibria, which from Proposition 1 is the optimal joint policy.

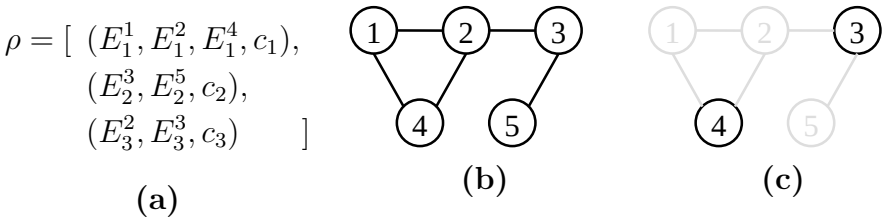
□

4.4 Running Time

An analysis of the algorithm as presented is quite straightforward. Constructing the augmented MDP in the function AUGMENT can be done in polynomial time with efficient representations of ρ and the MDP. The resulting augmented MDP is the same size as the original. SOLVE returns an optimal policy for an MDP, which for dynamic programming is known to be polynomial. INTERSECT finds a set of points. Each point involves solving a system of linear equations, which is polynomial in $|\rho|$. However, the number of points, while polynomial in $|OCS|$, is exponential in the number of dimensions $|\rho|$, $O(|ocs|^{|\rho|+1}/(|\rho|+1)!)$. For each point the optimal policy is found, which is polynomial in the size of the state space.

4.5 Extension to n Agents

Extending the coverage set algorithm to n agents is fairly straightforward. The reason is that the core of the algorithm, finding the optimal coverage set for agent i, does not depend on any particular behavior for agent j. When there are n agents, agent 1 can compute its optimal coverage set using the same algorithm by viewing



(a) **(b)** **(c)**

Figure 4.4. (a) A joint reward structure ρ over five agents. (b) The constraint graph. (c) The constraint graph after agents 1, 2, and 5 have found their optimal coverage set. The remaining agents 3 and 4 are not connected.

the other $n-1$ agents as one other agent. The only real change is that the notation becomes much more difficult to read. A point in the parameter space (Definition 16) for agent 1 becomes

$$\bar{x}_{\pi_2 \ldots \pi_n} = \left[\prod_{k=2}^{|G_1|} P(E_1^{g_{1k}} | \pi_{g_{1k}}), ..., \prod_{k=2}^{|G_{|\rho|}|} P(E_{|\rho|}^{g_{|\rho|k}} | \pi_{g_{|\rho|k}}) \right],$$

assuming that g_{*1} refers to agent 1. With two agents, dimension k represented the probability that the other agent would satisfy its part of constraint k. With n agents, dimension k represents the probability that the other $n-1$ agents will satisfy their parts of constraint k. Since the agents are independent, this is simply the product of the probabilities of each of the other agents satisfying its part of constraint k. The value of each dimension still ranges between 0 and 1, so this change does not affect the computation of the optimal coverage set. The dimensionality of the parameter space for agent i is no longer $|\rho|$, but is only the number of constraints agent i participates in. For example, if agent i is only involved in three constraints then the dimensionality of the search for the optimal coverage set for agent i is three, no matter how many other constraints there are in ρ.

The interesting part of the algorithm that significantly changes is which agents must compute their optimal coverage set. When there were only two agents, I had to

find the optimal coverage set for only one of the two agents. With n agents, I now have to find the optimal coverage set for some subset of the agents, possibly though not necessarily $n-1$ of them. For each constraint in the problem, at most one agent involved in that constraint does not have to find its optimal coverage set. This can easily be demonstrated by a simple reduction to a graph (see Figure 4.4). Let every agent be represented by a vertex. Add an edge between every two vertices that share a constraint (if there is not an edge there already). For example, for constraint 1, $G_1 = \{1, 2, 4\}$, add the edges $E(1, 2)$, $E(1, 4)$, and $E(2, 4)$. The optimal coverage set must be found for a subset of the agents such that when those vertices and all incident edges are removed from the graph, the remaining vertices are completely disconnected (no edges remain). Some valid subsets in this example are $\{1, 2, 5\}$, $\{2, 4, 5\}$, $\{1, 4, 3\}$, among others.

Given a valid subset of agents, the final step in the coverage set algorithm becomes a search process through all combinations of those agents' optimal policies. Given a policy for each in the subset of agents, maximizing the global value reduces to solving the corresponding augmented MDP for each of the remaining agents. As a distributed search process it maps nicely into a DCOP (Distributed Constraint Optimization Problem) [43, 80]. As a DCOP, each agent has one variable, which represents the policy that agent adopts. The domain of that variable is the optimal coverage set for that agent. There is a cost function for each of the agents' local rewards, and one for each constraint in ρ. Solving the DCOP finds the variable assignment that maximizes the sum of the cost functions.

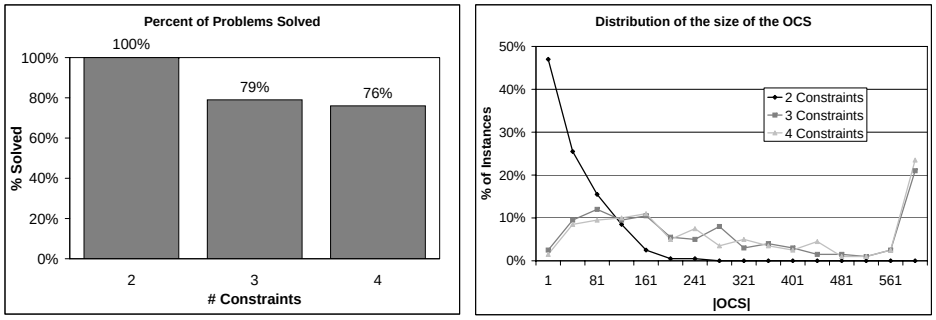
Choosing any subset of agents to find their optimal coverage set is easy, but choosing the best subset is not. In fact, it is not entirely clear what best means in this case. The smallest valid subset is not necessarily the one with the lowest worst case complexity, which is not necessarily the one with the lowest average case complexity, which is not necessarily the one that is easiest to distribute. The complexity of

finding the optimal coverage set is exponential in the number of constraints involved, so choosing a valid subset of agents that minimizes the maximum dimensionality is likely to be better than minimizing the number of agents. However, the complexity also depends polynomially on the number of policies in the optimal coverage set. If a particular local problem could be characterized in a way that indicates the expected number of optimal policies, then that could affect which valid subset should be chosen. Without this information, one possible solution is to work on finding all of the optimal coverage sets in parallel (possibly distributed, with each agent solving their own local problem) until a valid subset has been solved. Examining these issues in more depth remains the topic of future work.

4.6 Experimental Results

I implemented a general version of this algorithm that works for any number of dimensions. The implementation varied slightly from what is presented in Figure 4.1 to improve the efficiency. The primary change is instead of computing all of the intersection points in the function INTERSECT(), I generate one at a time. Many of the points returned by INTERSECT() can be discarded without the need to run SOLVE() on the augmented MDP at that point. This is because many points lie beneath other known optimal policies or because those points are out of bounds (> 1 or < 0 in some dimension of the parameter space).

The results were verified by independently checking that every policy in the optimal coverage set was optimal for some point in the parameter space and by randomly choosing points in the parameter space and verifying that its optimal policy was in the set.



(a) **(b)**

Figure 4.5. (a) The percentage of problems that had fewer than 600 policies in the optimal coverage set. Those with more than 600 were not solved. (b) The distribution over the size of the optimal coverage set for 2, 3 and 4 constraints.

4.6.1 Rover Exploration Problem

I tested this algorithm on problem instances modeled after the Mars rover example used throughout this paper. There are two rovers, each with an ordered set of sites to visit and collect data. The state for each rover is composed of their current task and the current time left in the day. The action for each state is to *skip* the current site or to *collect* data at the current site. If the rover chooses *skip*, then it moves on to the next site without wasting any time. The uncertainty in the problem is the length of time it takes to execute the tasks.

The problem instances generated had a total time of 10 units, and 5 sites for each rover. The original DEC-MDP had 250 world states ($10 \times 5 \times 5$), and both rovers' local decision problem had 50 states (10×5). On average, each rover was able to collect data at only half of its sites.

Some of the sites were shared between the two agents. These sites were assumed to be half redundant, which means that if both rovers collect at that site then the reward received by the system is 1.5 times that received if only one rover does. For example, if rover 1 collects then it receives local reward r. Similarly, if rover 2 collects it receives local reward r. Now if both rover 1 and rover 2 collect at that site, then

they each receive r and the system receives a penalty of $-0.5r$ for a net total global reward of $1.5r$.

Data was collected from 200 randomly generated problems. Each problem was solved three times, once with sites 3 and 4 shared between the agents, once with $2, 3$ and 4, and also with $1, 2, 3$ and 4. Problems that had more than 600 policies in their optimal coverage set were considered hard problems and skipped. Figure 4.5(a) shows the percentage of problems that were solved for the given number of constraints. Figure 4.5(b) shows the distribution of the size of the optimal coverage set. Of the problems solved, the average number of optimal policies in the optimal coverage set was 59, 217 and 230 for 2, 3 and 4 constraints respectively. While only 76% of the problems with four constraints were solved in the available time, this should be viewed in the context that complete policy search would not have been able to solve a single one of these problems in that time.

To get a general idea about the length of time to solve these problems, with four constraints the fastest problem had 23 optimal policies in the optimal coverage set and took about 1.5 seconds on a 2.8GHz Pentium 4. The longest one took about 4 hours and had 597 optimal policies.

There are two measures of work that I use to evaluate the performance. First is the number of times the planes are intersected. Each intersection represents solving a system of linear equations. The second measure is the number of times dynamic programming was used to solve an augmented MDP. The graphs were essentially the same regardless which measure I used, so most will be presented with the number of intersections as the measure of work. Figure 4.6 shows how the amount of work required to solve a problem relates to the size of the optimal coverage set.

The difficulty in solving a particular instance depends on many characteristics of the problem. The characteristic that I focus on here is the magnitude of the constant rewards in ρ, $c_1 \ldots c_x$, as compared to the other rewards in the system. Intuitively, if

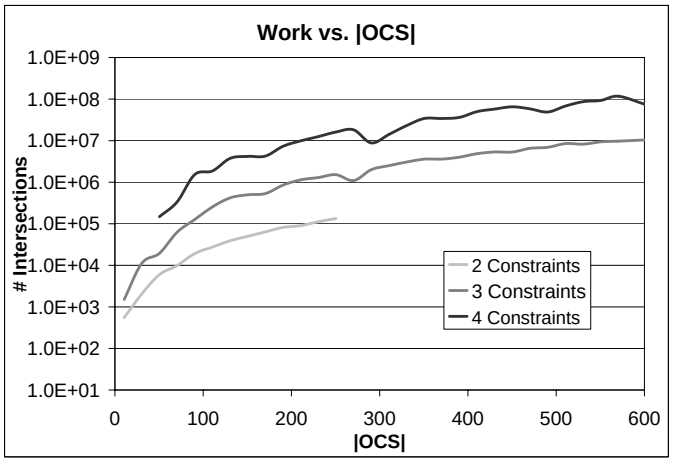


Figure 4.6. The amount of work measured by the number of intersections increases polynomially with the size of the optimal coverage set, $|OCS|$.

the value of a shared task is much greater or much less than the other tasks available to an agent, then the difficulty to decide whether to do the shared task decreases. This idea can be seen in Figure 4.7. When looking along either the x or y dimension, the density of optimal policies is greatest around the middle, and much less at either extreme.

Previously, I collected data on problems that were half redundant. The value of the shared sites ranged from r to $0.5r$, i.e., $r + P \times (-0.5r)$ where P is the probability of the other agent collecting at that site. Instead consider a problem in which the shared tasks were complementary. If either agent collects data it receives $0.5r$, and if they both collect data then the system receives an extra $0.5r$ for a total of $1.5r$. The values in this example range from $0.5r$ to r. The complexity of these two problems is identical because the policies in the optimal coverage set are identical. However, the optimal joint policy will be different because the problems are not identical but mirror images of each other. Similarly, if I increase c then the new optimal coverage set is a superset of the original and therefore more difficult to solve. Figure 4.7 illustrates this relation.

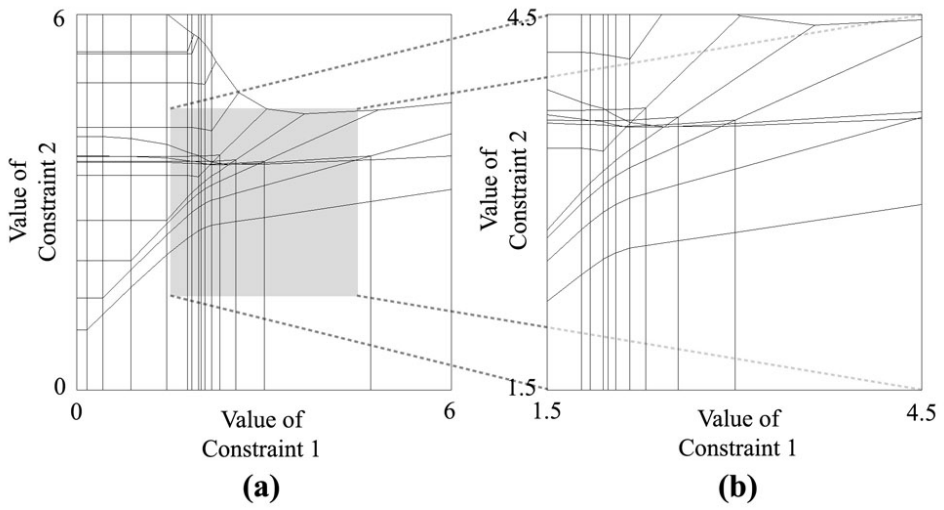


Figure 4.7. Illustration of the upper surface of the optimal coverage set for a 2 constraint problem. Each bounded region corresponds to one policy and the area over which it is optimal. (a) Both constraints have a local reward of 0 and an additional reward of 6. (b) Local reward is 1.5 and additional reward is 3. Notice how this set of policies is equal to the middle area of (a). Problem (b) is a subset of (a).

The focus of the experimental results is primarily on step two of the algorithm (finding the optimal coverage set) because that is where most of the computational complexity lies. That changes, however, as the number of agents increases. Finding the optimal coverage set is internal to each agent and depends on the number of constraints that agent has, not the number of agents in the system. Therefore, as the number of agents increase, the complexity of the third step (searching through the optimal coverage sets for the optimal joint policy) becomes more important. The third step is a distributed constraint optimization problem (DCOP), the evaluation of which is out of the scope of this paper. Mailler and Lesser [45], and Modi et al. [47] present state-of-the-art algorithms for solving DCOPs.

4.6.2 Approximation

While the complexity of transition independent decentralized MDPs is significantly lower than models like the DEC-MDP, it is still intractable for large problems

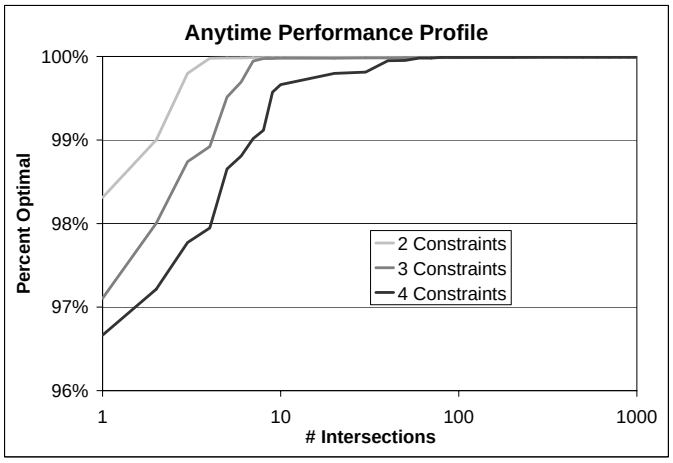
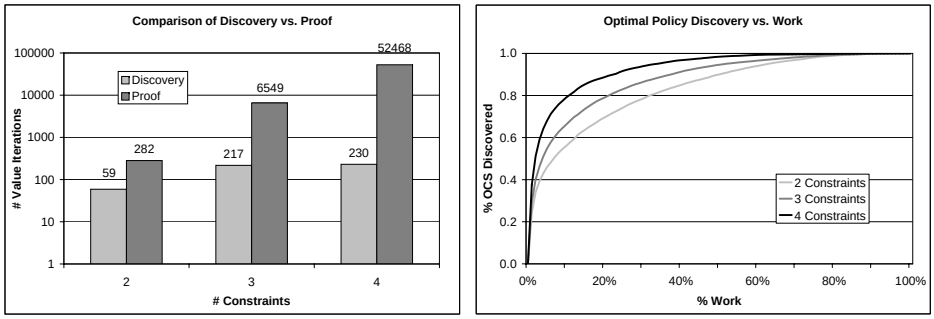


Figure 4.8. The value of the best known joint policy (as a percentage of the optimal policy) as a function of the total work done.

and certain hard instances of small problems. Fortunately, it turns out that the coverage set algorithm is naturally an anytime algorithm with some very nice characteristics.

Figure 4.8 shows the anytime performance profile over the same set of problems discussed earlier. The very first solution generated by the algorithm was better than 90% of optimal in nearly every problem. After only 1000 intersections, 88% of the 4-constraint problems had already found the optimal solution and the others were within 99% of optimal. 1000 intersections is only a small fraction of the total work done to solve one of those problems (average was 18 million).

There are several reasons for the excellent performance. First, each solution generated by the algorithm is a locally optimal solution. This means that agent 1 has a belief about what agent 2 will do and generates its optimal corresponding policy. Agent 2 then generates its optimal policy given agent 1's policy. This pair of policies is not a Nash equilibrium because Agent 1's belief about Agent 2 and Agent 2's actual policy may be different. However, in these randomly generated problems it is usually quite good.



(a) **(b)**

Figure 4.9. (a) As the number of constraints increase, the gap between the work to discover new optimal policies and the work to prove the set is complete increases. (b) The discovery of the optimal planes happens primarily in the beginning.

A second reason is that the amount of work required to discover the policies in the optimal coverage set is significantly less than that required to prove that the set is complete, as shown in Figure 4.9(a). As the number of constraints grows, the gap between the discovery work and the proof work increases. In addition, the discovery is front-loaded as seen in Figure 4.9(b). Finally, the more area over which a policy is optimal, the more likely that policy will be discovered earlier and the more likely that policy will be in the optimal joint policy (for randomly generated problems). Taken together, this explains why the optimal joint policy was usually discovered after only 1000 intersections. For the 4-constraint problems, this is less than one hundredth of one percent of the total work to solve the problem.

While these anytime results are promising, I have not yet addressed the issue of whether they are due to characteristics in the class of problems or the particular domain. To answer this question I plan to implement other domains as well as examine larger problems.

CHAPTER 5

DECENTRALIZED MDPS WITH EVENT-DRIVEN INTERACTIONS

The transition independent DEC-MDP represented domains whose local decision problems are tied only through the reward function. This section will present a new model that allows a limited form of direct interaction between the local problems. The interactions are called *event-driven interactions* and consist of events in one agent that affect the transition probabilities of another agent.

This interaction allows the outcome of actions performed by one agent to depend on the completion of certain tasks by the other agent. This form of interaction has been studied extensively within the multi-agent community [39, 69]. Some instances of this type of interaction that have been previously studied are enables/facilitates interrelationships, whereby one agent executing a task enables the other agent to execute another task, or it may increase the likelihood of success. Another example is a non-consumable resource, which one agent can lock and thus prevent the other agent from using.

To find the optimal solution to this class of problems I show how to construct an augmented MDP for the agents. Using this augmented MDP the Coverage Set Algorithm will find an optimal joint policy.

Intuitively, the special class of problems I focus on involves two agents, each having a "local" decision problem or task that can be modeled as a standard MDP. This local decision problem is fully observable by the agent. The agents interact with each other via a set of structured transition dependencies. That is, some actions taken by one agent may affect the transition function of the other agent.

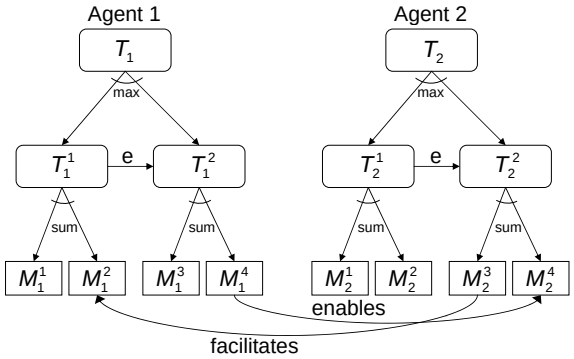


Figure 5.1. An example TAEMS task structure

To illustrate this concretely, I present a problem in TAEMS [15]. TAEMS is a hierarchical task modeling language that has been used successfully in a number of real systems [73]. Figure 5.1 is an example task structure. In it, the two agents each have one task: T_1 for agent 1 and T_2 for agent 2. Both of those tasks can be decomposed into two subtasks, i.e. $T_1 \rightarrow T_1^1$ and T_1^2. Each of the subtasks is decomposed into two methods, i.e. $T_1^1 \rightarrow M_1^1$ and M_1^2. Methods are the atomic units of a task and the agents can execute them. Executing a method takes *time* and produces *quality*, over some distribution. A quality of 0 represents a method that has not been successfully executed. Tasks accumulate quality from their children in many different ways. Two are shown in the example: sum and max. The quality of T_1^1 is the sum of the qualities of its children, and the quality of T_1 is the max quality of its children. The goal of the system is to maximize the sum of the qualities of the highest level tasks in both agents before the deadline.

The two agents do not operate independently, however. TAEMS has three different types of interrelationships between agents, only one of which is used in this example: facilitates/enables. This type of interrelation is a temporal constraint: M_1^4 must be executed successfully *before* M_2^4 is executed for M_2^4 to produce a nonzero quality. Facilitates is similar though less severe. If M_2^3 is successfully executed before M_1^2,

then M_1^2 is *more likely* to produce a higher quality than if it was not facilitated. These are examples of the type of dependencies allowed between agents in our model, though it is not limited to interrelations of this nature.

5.1 Formal Definition of the Model

The formal description of the model borrows many of the definitions from Chapter 3. The model is based on a 2-agent DEC-MDP with additional structure. The state space is factored, and the problem exhibits local full observability and reward independence. The only dependence between the agents is through the transition function.

For example, a TAEMS problem can be represented as follows:

- The local state of each of the agents is composed of the current time and the qualities of each of the methods.
- The actions for the agents are to execute one of their methods.
- The transition function is based on the time/quality distribution for the methods the agents choose to execute, taking into account the facilitates/enables interrelationships.
- The reward is only received in a terminal state, and it represents the sum of the qualities of the highest level tasks at that time.
- Each agent fully observes is own local state. In addition, when an agent attempts to execute a constrained method it learns whether the interrelationship was satisfied.

Next I define the interaction between the two agents as event-driven, meaning that an event in one agent influences an event in the other agent.

5.1.1 Event-Driven Interactions

In this section I will fully define the transition function. The basic idea is that transitions can take two forms. First, many local transitions for one agent are independent of the other agent, which means that they depend only on the local state and action. However, the interrelationships between the agents mean that some transitions depend on the other agent. This interaction is described by a dependency and the change in transitions that result when the dependency is satisfied.

The events used throughout this section are assumed to be proper. An example of an event would be successfully executing M_1^4 before time 10. It would be composed of primitive events of the form $(time < 10, q_1^1, q_1^2, q_1^3, q_1^4 = 0)$, *execute* M_1^4, $(time < 10, q_1^1, q_1^2, q_1^3, q_1^4 > 0)$, where q_i^k is the current quality of method M_i^k.

The event *executing* M_1^4 *before time* 10 is proper because the primitive events that compose the event include the transition from $q_1^4 = 0$ to $q_1^4 > 0$. Since the quality of a task is always nondecreasing, this transition can never occur twice.

The interaction between the agents takes the form of a triggering event in agent i and a set of state-action pairs for agent j that is affected. This interaction is called a dependency.

Definition 19 *A* **dependency** $d_{ij}^k = \langle E_i^k, D_j^k \rangle$*, where* E_i^k *is a proper event defined over primitive events for agent* i*, and* D_j^k *is a set of unique state-action pairs for agent* j*. Unique means* $\neg\exists k, k', \hat{s}_j, a_j \ s.t. \ \langle \hat{s}_j, a_j \rangle \in D_j^k \wedge \langle \hat{s}_j, a_j \rangle \in D_j^{k'} \wedge k \neq k'$.

Definition 20 *A dependency* d_{ij}^k *is* **satisfied** *when* $\Phi_i \models E_i^k$*. Boolean variable* $b_{\hat{s}_j a_j}$ *is* true *if the related dependency is satisfied and* false *if it is not satisfied or there is not a related dependency:*

$$b_{\hat{s}_j a_j} = \begin{cases} \text{true} & \exists k, \ s.t. \ \langle \hat{s}_j, a_j \rangle \in D_j^k \wedge \Phi_i \models E_i^k \\ \text{false} & otherwise \end{cases}$$

Definition 21 *A* **transition function** *for event driven interactions is divided into two functions, P_i and P_j. They define the distribution $P_i(\hat{s}'_i|\hat{s}_i, a_i, b_{\hat{s}_i a_i})$.*

When an agent takes an action that could be influenced by a dependency it learns the status of that dependency, whether or not it was satisfied (i.e. whether the task was facilitated). The idea is that an agent knows why things happened after they do. For example, if an agent attempts to execute a task that has not been enabled, it realizes that it does not have the data necessary for the task when it fails. An argument can be made that the agent should be able to check whether it has the available data before it attempts to execute the task. This can be accomplished by a 'free' action that reveals the status of the dependency. Essentially, the dependency modifies the transition for the free action in addition to facilitating the task.

Dependency $d^{10}_{1,2}$ is the dependency that represents method M^4_2 having been enabled when agent 2 attempts to execute it at time 10. The event E^{10}_1 is the event described earlier where the enabling method M^4_1 has finished executing successfully before time 10. D^{10}_2 contains state-action pairs representing agent 2 attempting to execute M^4_2 at time 10. There is exactly one dependency of this type for each time that agent 2 could attempt to execute method M^4_2. If agent 1 successfully executes M^4_1 at time 6, then all of the dependencies $d^t_{1,2}$ where $t > 6$ will be satisfied (by the same primitive event in agent 1), but each of those satisfied dependencies modify a different set of probabilities in agent 2. Agent 2 can attempt M^4_2 at time 10 and again at time 14, and both times the method will be enabled but through different dependencies ($d^{10}_{1,2}$ and $d^{14}_{1,2}$ respectively).

5.1.2 Defining the Policy

While I have defined a local state space, action space, transition function and reward function for each agent, this unfortunately does not define a local MDP for each agent. The reason is because the local state is not Markov. When an agent

learns the status of a dependency, it changes its belief about the state of the other agent and the impact of future dependencies. This information is contained within the history of an agent, but not in the previously defined local state. Therefore, the local state $S_i \times S_0$ of agent i must be modified to include the dependency history (i.e. at time t dependency d_{ij}^k was satisfied). This modified local state is S_i'. $\langle S_i', A_i, P_i, R_i \rangle$ does define a local MDP, and the local policy for agent i is $\pi_i : S_i' \rightarrow A_i$.

When one of the TAEMS agents attempts a task with an incoming dependency, its next observation includes whether that dependency was satisfied (i.e. was that task enabled). That information is not stored in the local state of the agent that I defined earlier because it was not necessary in the DEC-MDP (transitions in the DEC-MDP are defined over world states not local states). However, that knowledge changes this agent's belief about the other agent's local state, and that could influence a future expectation of a method being enabled.

For outgoing dependencies it is sufficient to add one variable per facilitates/enables interrelationship that represents the time the interrelationship was satisfied. The agent knows the related dependencies before that time were not satisfied and the ones afterward were. For the incoming dependencies, the state needs to store the last time an interrelationship was not satisfied as well as the first time the agent discovered it was satisfied.

In the TAEMS example, the additional variables to the local state is dependent on the number of interrelationships, not the number of dependencies. This is not true for all problems. In the worst case, the state may have to include the status of a dependency and the state in which the agent discovered that status every time the agent gains new information.

5.1.3 Evaluating a Joint Policy

The value function that I am trying to maximize is the original value function for the DEC-MDP. However, evaluating a pair of policies is much easier on a new DEC-MDP constructed from the expanded local states and new transition function because the policies are not defined over the same state space as the original DEC-MDP.

- The states $S' = S'_1 \times S'_2$.
- The actions $A = A_1 \times A_2$.
- The transition function $P = P_1 \times P_2$.
- The reward function $R = R_1 + R_2$.
- The observations for each agent are its local component of the state.

The expected reward the system will receive given a pair of policies can be found by running policy evaluation as if this was an MDP because there is a direct mapping from world state to joint action.

5.2 Problem Complexity

An upper bound on the complexity of the DEC-MDP with event driven interactions can be derived from the complexity of complete policy search.

Theorem 5 *A DEC-MDP with event driven interactions has complexity exponential in $|S|$ and doubly exponential in the number of dependencies.*

Proof In a DEC-MDP with event driven interactions, the number of joint policies is exponential in the number of states because the policy is a mapping from states to actions (unlike the DEC-MDP which is a mapping from sequences of observations to actions). However, the number of states in the new DEC-MDP is exponentially larger in the number of dependencies than the original state space. Therefore, the number

of joint policies is exponential in the original state space and doubly exponential in the number of dependencies.

Evaluating a policy can be done with standard policy evaluation for MDPs because in the new DEC-MDP, the joint policy is a direct mapping from world states to joint actions. Policy evaluation for MDPs is polynomial in the number of states, so this does not raise the complexity.

Therefore, the DEC-MDP with event driven interactions has complexity exponential in $|S|$ and doubly exponential in the number of dependencies. ∎

The reason this class of problems is easier to solve than DEC-MDPs is because the size of the policy space has been reduced. It was reduced by separating the part of the history that is necessary to be memorized (interactions between the agents) from the part that is not necessary (local state). Specific problems may have additional structure that further reduces the complexity. For example, in TAEMS the complete interaction history is not necessary to remember so the complexity is doubly exponential in the number of facilitates/enables, not the number of dependencies. If the problem is such that there is an ordering over the interactions and only the most recent interaction must be memorized then the complexity drops to exponential.

5.3 Solving the Problem

Solving an exponential (or worse) problem using complete policy search is intractable for all problems of moderate or larger size. The reason is that for complete policy search, the worst case complexity is also the best case complexity. That means for every problem, regardless of whether it has characteristics that make it a simple problem, every policy must be evaluated. Suppose in the TAEMS example, M_1^4 and M_2^4 have quality outcomes significantly higher than the other methods. A simple analysis of the problem would indicate that nearly any policy that attempts both M_1^4

and M_2^4 has a higher value than any policy that does not. This significantly reduces the number of policies that must be searched.

While the worst case complexity of the Coverage Set Algorithm may be similar to complete policy search, the best case is only polynomial in the state space and exponential in the number of dependencies. The reason the Coverage Set Algorithm has such a variance in complexity is that it is essentially performing a general analysis of the problem. The more obvious a solution, the faster the algorithm runs. However, in the worst case, no useful information is gained through the analysis and it will perform slower than complete policy search due to extra overhead.

5.3.1 Constructing the Augmented MDP

To show that the CSA can be used for this problem, I must define an augmented MDP.

Let $MDP_i = \langle S'_i, A_i, P_i, R_i \rangle$ represent the local MDP for agent i as defined earlier. Let $MDP_i^{\pi_j} = \langle S'_i, A_i, P'_i, R'_i \rangle$ represent the augmented MDP for a given π_j. The states and actions do not change in the augmented MDP, but the transition function and reward function do. The transition function changes to take into account the incoming dependencies, d_{ji}^k. It is modified by the likelihood that an incoming dependency is satisfied in the other agent and the change in probability that dependency incurs. The reward function is modified to incorporate the changes in expected value the other agent receives when outgoing dependencies are satisfied.

Definition 22 *$P'_i(\hat{s}'_i|\hat{s}_i, a_i)$ is the transition function for the augmented MDP. For the probabilities not altered by an incoming dependency, $P'_i(\hat{s}'_i|\hat{s}_i, a_i) = P_i(\hat{s}'_i|\hat{s}_i, a_i, false)$. For the others,*

$$\forall k, \langle \hat{s}_i, a_i \rangle \in D_i^k, P'_i(\hat{s}'_i|\hat{s}_i, a_i) = P_i(\hat{s}'_i|\hat{s}_i, a_i, false) +$$

$$P(d^k_{ji}|\hat{s}_i)\left[P_i(\hat{s}'_i|\hat{s}_i, a_i, true) - P_i(\hat{s}'_i|\hat{s}_i, a_i, false)\right],$$

where $P(d^k_{ji}|\hat{s}_i)$ *is the probability that dependency* d^k_{ji} *is satisfied given that agent* i *is in state* $\hat{s}_i$*.*

Definition 23 $R'_i(\hat{s}_i, a_i, \hat{s}'_i)$ *is the reward function for the augmented MDP. For each primitive event* $e = \langle \hat{s}_i, a_i, \hat{s}'_i \rangle$ *that does not satisfy an outgoing dependency,* $R'_i(\hat{s}_i, a_i, \hat{s}'_i) = R_i(\hat{s}_i, a_i, \hat{s}'_i)$*. For the others,*

$$\forall k, \langle \hat{s}_i, a_i, \hat{s}'_i \rangle \in E^k_i, R'_i(\hat{s}_i, a_i, \hat{s}'_i) =$$

$$R_i(\hat{s}_i, a_i, \hat{s}'_i) + V^{\hat{s}'_i}_{\pi_j}(\hat{s}^0_j) - V^{\hat{s}_i}_{\pi_j}(\hat{s}^0_j),$$

where $V^{\hat{s}_i}_{\pi_j}(\hat{s}^0_j)$ *is the expected value of the start state of agent* j*'s local MDP given the policy* π_j *and the dependency history contained in* $\hat{s}_i$*.*

The parameters from agent j's policy and MDP take two forms, $P(d^k_{ji}|\hat{s}_i)$ and $V^{\hat{s}_i}_{\pi_j}(\hat{s}^0_j)$. Neither of these is difficult to compute, and they can be derived at the same time. The value V can be obtained by running policy evaluation on the MDP obtained by applying the dependencies satisfied in $\hat{s}_i$. At the same time, the probability of reaching each state can be computed. $P(d^k_{ji}|\hat{s}_i)$ is the sum of the probabilities of all primitive events in E^k_j that have a consistent dependency history with $\hat{s}_i$.

A value function for an MDP can be represented as the sum over all primitive events in the MDP, the probability of that primitive event occurring times the reward received at that primitive event. This function can be converted to the form $V = c_0 + c_1x_1 + c_2x_2 + ...$ where x_n is the product of one or more parameters i.e., $P(d^3_{ji}|\hat{s}^6)P(d^5_{ji}|\hat{s}^{13})P(d^5_{ji}|\hat{s}^{16})$. Partially symbolic policy evaluation would generate a value function with this form. While this function is not linear in the parameters, it is linear in these combinations of parameters. Having a linear value function allows the use of the Coverage Set Algorithm.

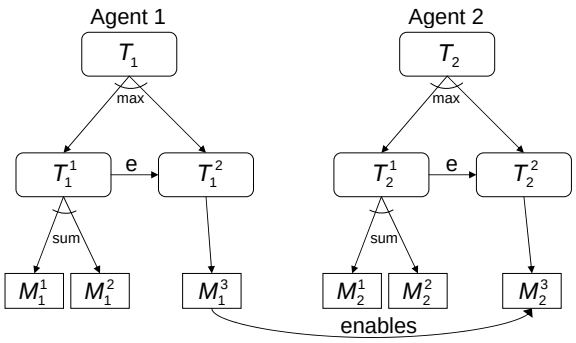


Figure 5.2. The TAEMS problem structure for the experiments.

5.4 Experimental Results

To test the algorithm on this class of problems, I implemented the example problem shown in Figure 5.2. The agents had 6 time steps to execute their methods. Each method took between 1 and 3 time steps to complete. Methods M^1 and M^2 produced an integer quality between 0 and 2, while M^3 produced a quality of 0, 2 or 4. The method M_2^3 took 1 time step and produced quality 0 if not enabled. After executing M_2^3 agent 2 knows whether it was enabled or not. The agents received a reward after the time ran out equal to the final quality of their task. The global reward being maximized is the sum of the local rewards: $Max(q_1^1 + q_1^2, q_1^3) + Max(q_2^1 + q_2^2, q_2^3)$. This section will examine a typical instance of this problem in detail.

The dimensionality of the search was different for the two agents because they are on different sides of the dependencies. Agent 1, being the enabler, had parameters that represented the expected value for agent 2 given that agent 1 enabled at time t. There were four different times that agent 1 could enable agent 2 and another parameter for when agent 2 was never enabled leading to a parameter space of size five. Agent 2, being on the receiving end, depends on the probability that it was enabled given the current time and the last time it learned it was not enabled. There were ten probabilities, but the parameter space depended on combinations of those

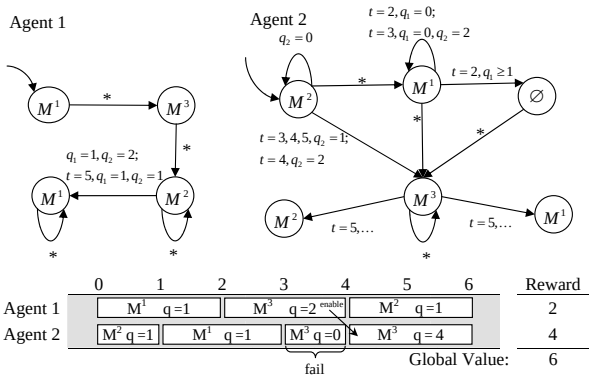


Figure 5.3. The optimal joint policy and an example execution.

probabilities and was much higher. Since agent 1 had a much lower dimensionality, I chose to find its optimal coverage set.

Figure 5.3 shows an FSM representation of the optimal joint policy for one instance of the problem. The optimal coverage set for agent 1 contained 141 policies and took only a few hours to find on a modern desktop computer.

While the complexity of these decentralized MDPs with event-driven interactions is significantly easier than models like the DEC-MDP, it is still intractable for large problems and certain hard small problems. Fortunately, it turns out that the coverage set algorithm is naturally an anytime algorithm with some very nice characteristics. Finding an optimal or near optimal solution usually takes very little time. Proving that the solution is optimal takes the majority of the computation. For example, the expected value of the optimal joint policy in Figure 5.3 is 5.8289. The first joint policy found had a value of 5.6062, or 96.2% of optimal. The optimal joint policy was discovered after only 0.004% of the total computation. The result is a good anytime solution to problems too large to solve optimally.

5.5 Summary

The DEC-MDP framework has been proposed to model cooperative multi-agent systems in which agents receive only partial information about the world. Computing the optimal solution to the general class is NEXP-complete, and with the exception of [33] the only known algorithm is brute force policy search. I have identified an interesting subset of problems that allows for interaction between the two agents in a fixed, structured way. This interaction could take the form of restricted communication. For this class of problems I have identified that the complexity reduces from being doubly exponential in the state space to doubly exponential in the level of interaction between the agents and only exponential in the state space. Since many problems have a level of interaction significantly lower than the number of states, the savings can be quite substantial.

I also provided a mapping to an algorithm that runs much faster than complete policy search. While the high complexity still makes it intractable for large problems, this work does facilitate finding optimal solutions for larger problems that have only a limited amount of interaction between the agents. That can be useful in establishing a baseline for evaluation of approximation algorithms.

The augmented MDP enables a simple yet powerful hill-climbing algorithm that converges on a local optimum. In addition, the coverage set algorithm is also naturally an anytime algorithm with some promising characteristics. Using these two approximations should facilitate finding good solutions to much larger problems. This remains the subject of future work.

CHAPTER 6

COMMUNICATION

The work presented so far is just beginning to address the tradeoff between computational complexity and general applicability of decision-theoretic multi-agent frameworks. The transition independent DEC-MDP verifies the intuition that significant computational savings can be gained when the local problems are largely independent. When the interaction between the agents is strictly through the global value function being maximized then the complexity drops to NP-complete. However, not all forms of limited interaction are easy to solve. An upper bound for the event-driven interactions puts the complexity at doubly exponential in the level of interaction and exponential in the state space. This is a significant savings over the DEC-MDP, however, it still represents an intractable problem. While both of the models presented so far represent useful and interesting classes of problems, there is one common characteristic of multi-agent systems that is not represented—communication.

Deciding when to communicate is a fundamental challenge in multi-agent systems, and finding the optimal communication policy is usually intractable in decentralized problems when communication has a cost, ranging from NP-complete to NEXP-complete [28, 59]. This decision can be formulated as a value of information problem. The value of the information collected and disseminated can be measured by the difference between the improvement in the agents' performance and the costs associated with communication, regardless whether communication takes the form of state information, intentions or commitments. The optimal communication policy

involves the agents choosing the communicative act at each step that maximizes the expected future utility, much like choosing an optimal action in an MDP.

Information value theory [37] is an important component of decision making, and it has been used to calculate the value of information in different settings, for example the expected value of computation [35]. However, even in the single-agent contexts where information value theory has been extensively used, finding the exact value is very difficult. The typical approach to dealing with this complexity is to approximate it with two myopic assumptions: each source of information is evaluated in isolation and they are evaluated with a 1-step horizon [57].

Others [26, 27, 69] have extended these myopic assumptions to multiple agents in order to generate communication policies. Frequently, however, the exact assumptions being made are not clearly stated. Additionally, a careful analysis of the impact of these assumptions on the quality of the resulting communication policy has not been made. While the myopic assumptions may be an appropriate way to approximate the value of information in the single-agent case, it is not obvious that they remain so for the multi-agent case.

This work attempts to improve the understanding of communication in multi-agent systems by examining the implications of the myopic assumptions. First, I clearly state the basic myopic assumptions and formally show how to compute the optimal communication policy given these assumptions. I then identify and describe two facets of the assumptions that introduce error, and I provide an improved way to compute the communication policy that compensates for this bias.

I perform the analysis of communication using the Transition Independent Decentralized MDP [5] as the underlying multi-agent framework extended to allow communication between the agents. I chose this model for several reasons. First, decision-theoretic models are a formal way of describing a problem and have natural definitions of optimality. Second, in this framework each agent has a different, local

view of the world. Their actions are based on their own local views, so an agent does not know the actions of the other agents even if it knows the other agents' policies. Centralized models, like the MMDP [9], are not capable of (naturally) representing this decentralized view. Third, this model also has a known algorithm to find the optimal joint policy assuming zero communication. This guarantees that the results of the analysis are not due to interactions between two heuristics.

This model also isolates the effect communication has on the expected value of a problem by imposing a strict separation between domain-level actions and communicative acts. Most other decision-theoretic, multi-agent models allow domain-level actions to include implicit forms of communication [6, 27, 59], which make analysis difficult. Implicit communication occurs when one agent gains information about another agent's state through a non-communicative act. This communication is often represented within the transition function and is difficult to quantify. For example, a robot attempts to move forward and fails. The failure could be caused by the wheels spinning in place or by another robot sitting in front of it. Therefore, its failure to move forward changes its belief about the location of the other robot.

Many other researchers have studied different aspects of communication. Some have worked with algorithms not based on myopic assumptions, like Reinforcement Learning [24]. The advantage of using RL is that they do not need a complete model of the problem, but they do their learning online and potentially make very bad decisions until they learn better ones. Others have addressed different questions, like **what** the agents should communicate [63] instead of **when**.

Xuan and Lesser [76] have worked toward understanding communication as a way to reduce uncertainty. This work compliments and builds on their understanding of communication by using the value of information as a quantitative measure of the benefit of reducing uncertainty.

6.1 Problem Description

In this section I will present an extension to the Transition Independent DEC-MDP that adds communication. First, however, I will discuss some of the design decisions.

Communicative acts can be represented explicitly or implicitly. For example, the DEC-POMDP has an implicit representation of communication where an action taken by one agent could be a communicative action, a non-communicative (domain) action, or a mixture of the two. This is handled by the transition function, which is a function of world states and joint actions. Xuan et al. [77], on the other hand, use an explicit representation of communication where each step of the decision process consists of a domain action and a communication action. The COM-MTDP [59] takes a similar approach.

While there is no expressive difference between the DEC-POMDP and the COM-MTDP, Xuan et al. demonstrates one advantage of explicit communication: the domain actions may have very different characteristics from the communication actions, which can be exploited to simplify the problem. If communication is added implicitly to the model through generalizing the transition function, then any structural difference between the two types of actions is lost. The explicit communication also allows them to focus on the effects of different communication policies. For example, the model I used without communication is a special case where the communication policy is to never communicate. The communication policy of always communicate complete information transforms the problem into an MMDP. In between these two extremes are many other interesting policies for communication.

Adding communication to the Transition Independent DEC-MDP violates some of the independence relationships. First, observation independence is violated because part of the observations is the message that the agent receives, which is dependent on an action taken by the sending agent. The second effect is that the policy is no

longer a mapping of local states to actions unless the messages received are added to the state, which would violate transition independence. However, with explicit communication, the domain actions can maintain their independence properties and that can be exploited to simplify the problem even though the agents themselves are no longer independent.

The other issue is the language of communication. There are two types of information that could be beneficial for the agents to communicate: observations and policy. In this work I generate complete policies for all agents so it is not necessary for the agents to transmit policy information. This leaves observations, which in the case of the TI-DEC-MDP is just the local states of the agents. Goldman and Zilberstein [28] showed that for a fixed communication cost, communicating their current local state is sufficient to maximize the global value.

Shen et al. [63] take a different approach to communication. Instead of assuming that the agents transmit their state information at a fixed cost, they allow the agents to communicate components of their state information at reduced cost. The problem they are solving, then, is not *when* to communicate but *what* to communicate.

The remainder of this chapter will use the Transition Independent DEC-MDP framework extended with a synchronizing communication protocol. The model is composed of n cooperative agents. Each agent i works on its own local subproblem that is described by an MDP, $\langle S_i, A_i, P_i, R_i \rangle$. The local subproblem for agent i is completely independent of the local subproblems for the other agents, and completely observable only by agent i. This means that at each step agent i takes action $a_i \in A_i$ and transitions from state $s_i \in S_i$ to $s'_i \in S_i$ with probability $P_i(s'_i|s_i, a_i)$ and receives reward $R_i(s'_i)$. The state of the world is just the collective local states of all of the agents.

At each time step each agent first performs a domain-level action (one that affects its local MDP) and then a communication action. The communication actions

are simply *communicate* or *not communicate*. If at least one agent chooses to communicate, then **every** agent broadcasts its local state to every other agent. This synchronizes the world view of the agents, providing each agent complete information about the current world state. The cost of communication is $\mathcal{C}$ if at least one agent initiates it, and it is treated as a negative reward. An optimal joint policy for this problem is composed of a local policy for each agent. Each local policy is a mapping from the current local state $s_i \in S_i$, the last synchronized world state $\langle s_1 ... s_n \rangle \in \langle S_1 ... S_n \rangle$, and the time T since the last synchronization to a domain-level action and a communication action, $\pi_i : S_i \times \langle S_1 ... S_n \rangle \times T \rightarrow A_i \times \{yes, no\}$. I will occasionally refer to domain-level policies and communication policies as separate entities, which is just a mapping to A_i and $\{yes, no\}$ respectively.

In addition to the individual agents accruing rewards from their local subproblems, the system also receives reward based on the joint states of the agents. This is captured in the global reward function $R : S_1 \times ... S_n \rightarrow \Re$. To the extent that the global reward function depends on past history it must be included in the local states of the agents just like the local rewards. The goal is to find a joint policy $\langle \pi_1 ... \pi_n \rangle$ that maximizes the global value function V, which is the sum of the expected rewards from the local subproblems and the expected reward the system receives from the global reward function.

Definition 24 *The global value function* $V(s_1 ... s_n) =$

$$\sum_{s'_1 ... s'_n} \prod_{i=1}^{n} P_i(s'_i | s_i, a_i) \left[\sum_{i=1}^{n} R_i(s'_i) + R(s'_1 ... s'_n) + V(s'_1 ... s'_n) \right] \tag{6.1}$$

To summarize, the class of problems I am dealing with can be defined by n MDPs, a global reward function R, and synchronizing communication between the agents with a fixed cost $\mathcal{C}$.

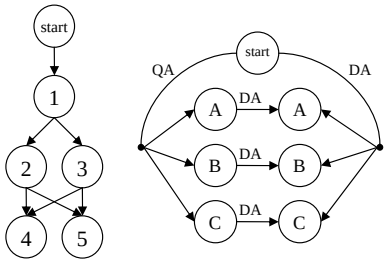


Figure 6.1. Graphical depiction of an example decision problem. (left) A partially ordered list of 5 sites. (right) A decision problem for one site with three potential classes.

The complexity of the related decision problem to this class is NP-complete [28], which is low for a decentralized problem with communication. A key structure in the model that keeps the complexity at NP-complete is the synchronizing communication protocol. When any information is transferred between the agents it is complete information so only the last communication must be remembered. Without this, the agents must remember the entire history of communication to make correct decisions, which results in an exponential increase in the size of the policies and a doubly-exponential increase in the solution time.

6.2 Example Application

I illustrate this class of problems with the following multi-agent data collection example. This example can be viewed as an abstraction of many different types of data collection problems, though I will present it as a rover exploration problem. Consider n rovers exploring a landscape and collecting data. Each rover has its own partially ordered list of sites it can visit, see Figure 6.1 (left). Each site contains a particular class of information. This class is not known *a priori*, instead the rover has a distribution over the classes for each site. See Figure 6.1 (right) for an example decision problem of a site with three classes of information. Each site has a similar decision problem associated with it. For example, the site could be an interesting rock

formation. With 70% probability it could be (A) a sedimentary rock, 25% (B) an igneous rock, and 5% (C) a fossil. The value of discovering and collecting data from a fossil may be significantly higher than collecting data from yet another sedimentary rock.

When a rover arrives at a site it has two choices. First, it can gather the information through a Detailed Analysis (DA) without knowing what class of information it is collecting. Alternatively, the rover can perform a Quick Analysis (QA) to determine the class of information available at the site before choosing whether to collect the information. The rover is restricted from collecting information at every site due to limited resources, like time and battery power.

The value of a DA comes from the information collected. The value of a QA is that it consumes fewer resources than a DA and allows the rover to make a more informed decision. The system receives reward based on the total information collected by all of the rovers. Each class of information has a base value. If the information in a particular class is redundant then the total value for collecting that class more than once may be only slightly higher than the base value. Alternatively, a class could be complementary, in which case the value for two pieces of information may be greater than twice the base value. The values of the information are captured in the global reward function.

6.3 Basic Myopic Approach

Using a myopic algorithm is a common way of dealing with the complexity inherent in finding an optimal solution. I present an algorithm for determining when the agents should communicate. This algorithm is optimal assuming that it must be initiated by the current agent (agent i in the following description) and that the current step is the only time communication is possible. For clarity the equations are presented

for two agents i and j, but the approach easily extends to n agents. The complexity results still include all n agents.

While the problems I am solving are distributed in nature (each agent chooses an action based on its own local view) the algorithm I present here computes offline the policies for each agent in a centralized location with a fully specified model of the problem, and the individual policies are given to the agents to follow. This does not trivialize the problem, nor does it reduce it to a single MDP since the solution found is still a decentralized solution. I chose this approach for two reasons. First, individual agents often lack the computational resources necessary to generate high quality solutions. Second, individual agents often lack a global view of the problem, which while not strictly necessary does simplify the solution process and reduces the communication between the agents (which has a cost).

The basic idea is that each agent follows the optimal policy assuming no future communication, which is obtained using the Coverage Set Algorithm (CSA) [5]. At each state, the agent chooses whether to communicate by computing the **Value of Communication** (VoC). If the VoC > 0 then the agent initiates communication causing all of the agents to broadcast their local state. This synchronizes the local views of all of the agents to the world state. The agents then compute a new optimal policy assuming no future communication, using their synchronized world state as the starting state. The domain-level actions the agents take always come from this zero-communication policy.

The VoC from agent i's perspective depends on i's current local state s_i, the previous synchronized world state (or original starting state) $\langle s_i^0, s_j^0 \rangle$, and the time since the last synchronization t. It also implicitly depends on the optimal joint policy assuming zero communication that the agents have been following since the previous synchronization, $\langle \pi_i^0, \pi_j^0 \rangle$.

Definition 25 *The Value of Communication (VoC) is the difference between the expected value when communicating and the expected value for remaining silent.*

$$\text{VoC}\left(s_i, \langle s_i^0, s_j^0 \rangle, t\right) = \sum_{s_j} P(s_j | s_j^0, t, \pi_j^0) \left[V^*(s_i, s_j) - \mathcal{C} - V(s_i, s_j) \right], \tag{6.2}$$

where $P(s_j|s_j^0, t, \pi_j^0)$ is agent i's belief about agent j's current local state, $V(s_i, s_j)$ is the expected value for following the current local policy, and $V^*(s_i, s_j) - \mathcal{C}$ is the expected value if the agents communicate now and follow a new zero communication policy after synchronizing.

The complexity of the VoC depends on the size of the local state space as well as the number of agents.

Theorem 6 *Computing the Value of Communication can be done in time polynomial in the number of local states and exponential in the number of agents.*

Proof. There are four components to computing the VoC that add to the complexity:

- $P(s_j|s_j^0, t, \pi_j^0)$ is the t-step transition function for agent j. Given the assumption that j will never initiate communication,

$$P(s_j|s_j^0, t, \pi_j^0) = \sum_{s_j'} P(s_j'|s_j^0, t-1, \pi_j^0) P(s_j|s_j, \pi_j^0). \tag{6.3}$$

  This takes $O(|S_j|)$ if the values from $t-1$ were cached from a previous call to VoC and $O(|S_j|^2)$ to compute from scratch.

- $V(s_i, s_j)$ and $V^*(s_i, s_j)$ are both expected values (see Definition 24). The only difference is that they assume different domain-level policies. With dynamic programming they can be solved in time polynomial in the number of world states, which is exponential in the number of agents, $O(|S_i|^n)$.

- The difficult part of computing the VoC is finding the new optimal joint policy with no communication for the different possible world states. I observed that the CSA does not need to be run in its entirety each time. Instead, most of the computation can be cached and only the final step of the algorithm must be rerun for each world state. That step involves searching through a small set of policies for each agent for the optimal joint policy. This step takes time exponential in the number of agents.

- When there are $n > 2$ agents the summation in the VoC is over all possible local states of the other agents. The loop, therefore, must be repeated $O(|S_j|^{n-1})$ times. However, it is useful to note that $V^*(s_i, s_j) - V(s_i, s_j) \geq 0$ and therefore the summation can terminate as soon as it becomes greater than $\mathcal{C}$ instead of looping through all possible next states.

The net result is a complexity polynomial in the number of local states for the agents and exponential in the number of agents. □

A final point about the complexity is the number of times VoC must be executed to generate the joint communication policy. While the worst case appears to be quite large, $O(n|S|^{n+2})$, in practice it is not nearly that bad. The reason is that many of the combinations of variables are not reachable. For example, if communication is frequent, then the time since the last communication, t, will remain low. If communication is infrequent then the number of reachable synchronized world states $\langle s_i^0, s_j^0 \rangle$ remains low because the world state is only synchronized through communicating. Additionally, there will be substantial overlap in computation between calls to VoC and caching can greatly reduce the running time in practice.

6.4 Implications of the Myopic Assumption

The myopic assumption allows a simple, straightforward computation of the value of communication. While this may be a good assumption for the single agent case,

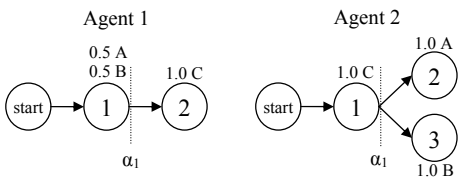


Figure 6.2. A simple example that illustrates how a simple model for the other agent introduces error.

there are additional implications that may not be readily apparent in a multi-agent setting. I examine these implications by identifying and analyzing two sources of error in the basic myopic approach, and for each I illustrate it with a simple example.

6.4.1 Modelling the Other Agents

The *Basic* myopic approach (Definition 25) assumes the simplest of models for the other agents–they never initiate communication. However, since every agent is following a communication policy based on computing the value of communication this is an inaccurate model. The first implication of an accurate model of the other agents is that not communicating itself becomes a form of communication. The distribution of states agent j can be in after t steps, $P(s_j|s_j^0, t, \pi_j^0)$, changes because j is known to not have passed through states in which it would have communicated.

The second implication is that at the current step, agent i may not need to initiate communication to acquire valuable information from agent j if j is already planning to initiate if it has the information. Figure 6.2 illustrates this with a simple example where agent 1 collects information valuable to agent 2. At site 1, agent 1 has an equal chance of collecting an A or a B. If both agents collect A's or B's the system receives reward 10. The system also receives a reward of 1 every time class C is collected. α_1 is the communication point of interest.

The initial zero-communication policy is for agent 2 to collect data from site 2. The only reason to communicate is if agent 1 collects a B, agent 2 needs to change

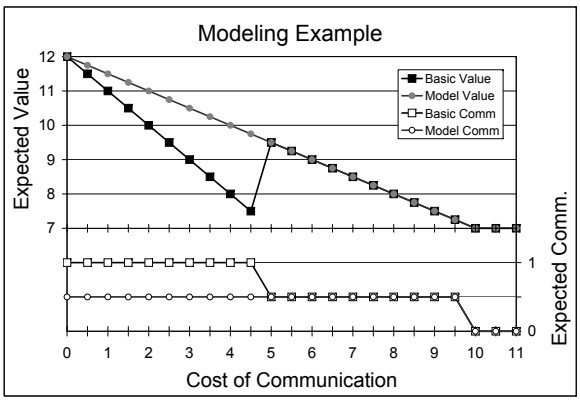


Figure 6.3. Performance comparison of the *Basic* and *Model* approaches.

its policy to go to site 3. Based on the initial policy, 50% of the time the agents will receive the maximum reward of 12 and 50% the minimum reward of 2. When agent 1 collects a B, its VoC $= -\mathcal{C} + 1.0[12 - 2] = -\mathcal{C} + 10$. As long as the cost $\mathcal{C} < 10$, agent 1 will initiate communication. Agent 2 does not know what agent 1 has collected, so its VoC $= -\mathcal{C} + 0.5[12 - 12] + 0.5[12 - 2] = -\mathcal{C} + 5$. When the cost of communication $\mathcal{C} < 5$ agent 2 will communicate because its VoC > 0. Half of the time this communication is unnecessary because agent 1 had collected an A. When $\mathcal{C} \geq 5$ it is no longer valuable for agent 2 to initiate the communication and their communication policies are optimal.

The *Basic* line in Figure 6.3 shows the performance of the basic myopic strategy. As the cost of communication increases from 4.5 to 5 it exhibits a jump in value. This undesirable behavior is caused by error introduced into the VoC by not accounting for the other agent's communication policy. This error can be removed from the approximation by computing an optimal **joint** communication policy for each step (assuming no future communication) instead of an optimal **local** communication policy.

To compute the optimal joint communication policy for the current step, the agents must maximize the expected value over all possible world states they could be in. They do this by creating a table M with rows representing the possible states of

	s_2^1	s_2^2	s_2^3	π_{1c}	VoC
s_1^1	**-1**	0	-1	*no*	-2
s_1^2	**4**	**-1**	**-1**	***yes***	**2**
s_1^3	**-2**	-1	1	*no*	-2
π_{2c}	***yes***	*no*	*no*		
VoC	**1**	-2	-1		

Figure 6.4. A Table M showing the expected gain in value for communicating for each world state.

agent 1 and columns representing states of agent 2 for the current step (see Figure 6.4).[1] The elements in the table are the value of communicating in that world state weighted by the probability that it is the current world state,

$$M_{xy} = P\left(s_1^x | s_1^0, t, \pi_1^0\right) P\left(s_2^y | s_2^0, t, \pi_2^0\right) \left[V^*\left(s_1^x, s_2^y\right) - \mathcal{C} - V\left(s_1^x, s_2^y\right)\right] \tag{6.4}$$

The *Basic* approach (π_{1c} and π_{2c} in Figure 6.4) represents building a communication policy for each agent by checking if the sum of a row or column is greater than 0. This strategy double counts certain elements in the table and can result in choosing a communication policy worse than not communicating at all! The expected value of a joint communication policy for one step is the sum of all entries in the table where communication happens (an entry is only counted once, even if both agents initiate communication). In the example table, the *Basic* policy given has a value of -1 (sum of the bold entries) because the valuable state $M_{2,1}$ was counted twice for determining the policies (once for each policy), but only once for determining the value of the table. If agent 2 did not communicate in s_1 then the value would be 2. Never communicating ($\pi_{ic} = \{no, no, no\}$) will always have a value of 0.

[1] This table does not correspond to the problem in Figures 6.2 and 6.3.

The optimal joint communication policy is the joint policy that maximizes the value of this table. Finding the optimal joint policy is exponential in the size of the table, while a simple hill-climbing algorithm can find a Nash equilibrium in polynomial time. The line labeled *Model* in Figure 6.3 optimizes this table to eliminate the error, resulting in the optimal policy for this example.

Creating the table costs no more than the original approach since each entry represents a reachable world state.

6.4.2 Myopic View of the Future

The second facet of the myopic assumption is that no agent will communicate in the future. This approximates the true value of communication by introducing error in two ways. The first is due to the greedy nature of the algorithm. When communicating immediately has a positive value, VoC > 0, the agent communicates without considering whether the expected value would be even higher if it waited to communicate until a future step. To compensate, the agents can compute the value of (possibly) communicating after a 1-step delay:

$$\mathrm{VoC}_{delay}\left(s_i, \langle s_i^0, s_j^0\rangle, t\right) =$$

$$\sum_{s_i'} P(s_i'|s_i, \pi_i^0) \times \max\left(0, \mathrm{VoC}\left(s_i', \langle s_i^0, s_j^0\rangle, t+1\right)\right).$$

The agent will initiate communication when its VoC $>$ VoC$_{delay}$. This does not imply that the agent really will initiate communication in the next step because the same comparison will be made at that time to later steps. As long as the expected value for delaying one step is greater than the value of communicating immediately, the agent will delay communication.

Figure 6.5 illustrates this with a simple example. If agent 1 collects A at site 1 then agent 2 should go to site 3, otherwise agent 2 should go to site 4. Similarly with

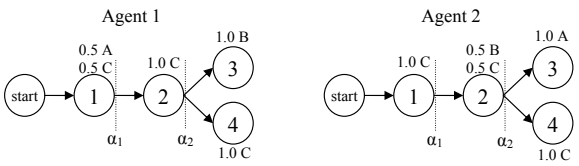


Figure 6.5. A simple example that illustrates how delaying communication can improve the expected value.

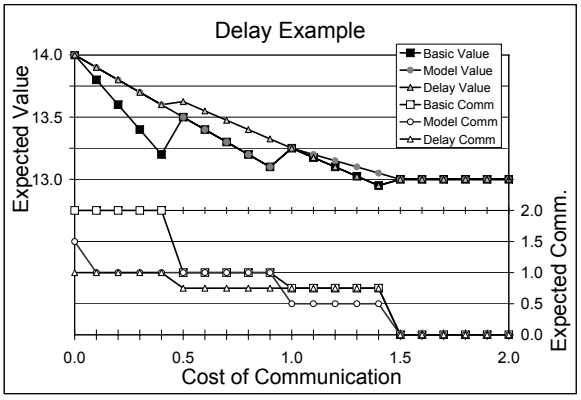


Figure 6.6. The expected value and expected amount of communication as a function of cost.

agent 2 collecting B at site 2. Like the previous example, two A's or two B's have a reward of 10, and each C adds a reward of 1. α_1 and α_2 are the two communication points. The *Basic* approach will always communicate at both α_1 and α_2 for low communication cost (See Figure 6.6). When the cost increases to 0.5, the agents will only communicate when they have valuable information. Agent 1 will initiate communication 50% of the time at α_1 and agent 2 will 50% of the time at α_2, for a total expected communication of $0.5 + 0.5 = 1.0$. The *Delay* policy, on the other hand, recognizes that waiting a step is beneficial and will only communicate at α_2, which reduces the communication without decreasing the expected reward, yielding a higher expected value.

When the cost goes above 1, the *Model* approach realizes that it is more efficient to have only one agent initiate communication when it has valuable information. This

illustrates that the *Model* and *Delay* approaches address different sources of error and neither dominates the other.

A second source of error in the assumption of no future communication is built in to the policies generated by the CSA. These policies may avoid situations which are valuable only when close coordination is possible. The optimal solution can exploit the possibility of future communication, while the domain-level policies generated here always assume no future communication. When the cost of communication is high enough, this solution is optimal. It is our belief that as the cost decreases, the solutions generated by this approach decline in quality compared to optimal. This source of error can also be partially compensated for by extending the 1-step delay to consider h-steps into the future.

6.5 Model-Lookahead Approach

This section demonstrates how the *Model* approach of 6.4.1 and the *Delay* approach of 6.4.2 can be merged together and extended to consider further into the future. The basic idea is an algorithm that makes optimal communication decisions within a horizon h given fixed domain-level policies based on zero communication.

To start I introduce two new value functions. $V^h(s_i, s_j)$ is the expected value of not communicating in the current step, following an optimal communication policy for the next h steps, and then not communicating again after h steps. $V^{*h}(s_i, s_j) - \mathcal{C}$ is similar but starts with an immediate communication. When the horizon is 0 these value functions are equivalent to the single-step value functions from Definition 25, $V^0(\cdot) = V(\cdot)$, $V^{*0}(\cdot) = V^*(\cdot)$.

$$V^h(s_i, s_j) = \tag{6.5}$$

$$\sum_{s'_i,s'_j \in \text{Comm}} P(s'_i|s_i,\pi^0_i)P(s'_j|s_j,\pi^0_j)\left[\mathcal{R}(s'_i,s'_j)+V^{*h-1}(s'_i,s'_j)-\mathcal{C}\right]$$
$$+\sum_{s'_i,s'_j \in \neg\text{Comm}} P(s'_i|s_i,\pi^0_i)P(s'_j|s_j,\pi^0_j)\left[\mathcal{R}(s'_i,s'_j)+V^{h-1}(s'_i,s'_j)\right]$$

where $\mathcal{R}$ is the sum of the reward functions, $\mathcal{R}(s'_i,s'_j) = R_i(s'_i)+R_j(s'_j)+R(s'_i,s'_j)$. Comm is the set of states in which communication will take place. How it is computed is not clear until I transform the equation. The derivation of Equations 6.5 and 6.6 can be found in Appendix A.

$$V^h(s_i,s_j) = V(s_i,s_j) \tag{6.6}$$

$$+\sum_{s'_i,s'_j \in \text{Comm}} P(s'_i|s_i,\pi_i)P(s'_j|s_j,\pi_j)\left[V^{*h-1}(s'_i,s'_j)-\mathcal{C}-V^{h-1}(s'_i,s'_j)\right]$$
$$+\sum_{s'_i,s'_j} P(s'_i|s_i,\pi_i)P(s'_j|s_j,\pi_j)\left[V^{h-1}(s'_i,s'_j)-V(s'_i,s'_j)\right]$$

The agents must find the set of communication states for the next step that maximizes $V^h(s_i,s_j)$. The next step communication policy only affects the second line of Equation (6.6), which bears a remarkable similarity to Equation (6.4), except that this is a recursive function. The same table algorithm can be applied to generate optimal communication policies over the horizon.

Figure 6.7 illustrates the performance of this approach on a larger problem. The two agents each had a local decision problem with 6 steps and more than 10,000 states. The *Model-Lookahead* approach performs significantly better than the original *Basic* approach and demonstrates a smooth reduction of the expected value as the cost for communication increases.

Figure 6.8 shows the running time of *Model-Lookahead* compared to *Basic*. The *Basic* approach took about 11 seconds to generate the entire policy while *Model-Lookahead* took 50% longer with a horizon of 0 due to the added cost of finding the optimal communication policies of the tables. The worst case complexity of

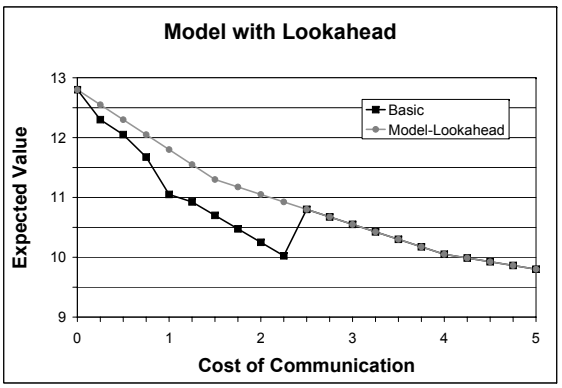


Figure 6.7. Performance of the *Model-Lookahead* Approach with horizon 2.

Model-Lookahead is exponential in the size of the horizon, but due to caching and the structure of the problem, in practice this is not always the case. In this example, the running time started out with an exponential curve but that changed as the horizon approached the number of steps in the problem.

This approach does have its limitations. Even when the horizon is equal to the number of steps in the decision problem, the policy generated is not the optimal joint policy. This is because the domain-level actions taken by the agents are generated assuming no future communication. This is effectively a horizon of 0 for choosing domain-level actions. Future work will include extending this algorithm to a larger domain-level action horizon.

6.6 Summary

This chapter addresses the problem of choosing when to communicate in a multi-agent system. I formulate a condition for communication based on the value of information. The standard assumption used to efficiently generate communication policies is that communication is only possible at the present time. This is based on the myopic assumption from information value theory.

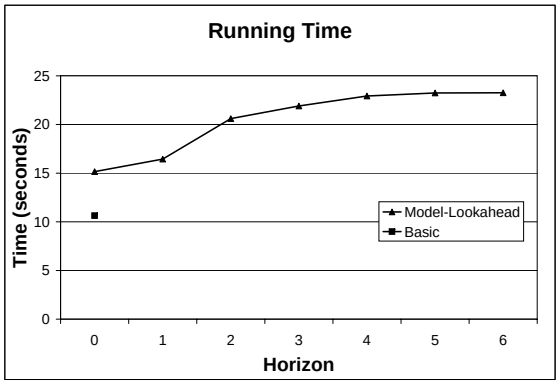


Figure 6.8. Comparison of the time to compute the policy for the *Basic* approach versus the *Model-Lookahead* approach of various depths.

I show how to generate optimal joint policies under the myopic communication assumption. I also examine the implications of the assumption and show that it can lead to poor agent behavior. I identify two sources of error and provide modifications to the original algorithm to address these problems. Together, these modifications result in an improved algorithm for generating a decentralized joint policy. Moreover, the computational overhead of our modifications is small compared to the original algorithm, for a small horizon, and increasing the horizon adds flexibility in the tradeoff between solution quality and computation time.

While the sources of error that I identify and the general approach to addressing them are common to many multi-agent systems, the equations and specific algorithms I present do rely on certain structure being present in the problem. The key structure in the model that reduces the complexity to NP-complete is the synchronizing communication protocol. Without this, the agents must remember the entire history of communication to make correct decisions, which results in an exponential increase in the size of the policies and a doubly-exponential increase in the solution time.

There are two components that together allow the use of synchronizing communication as an exact model. First is the fixed cost of communication. If the agents can

send partial state information at a reduced cost then the optimal solution may include communication that does not synchronize the agents' view of the world. Second is the transition and observation independence between the domain-level actions. If the agents are able to take domain-level actions that affect the observations or transitions of another agent then the agents have a form of implicit communication and must memorize the history to make correct decisions.

If a problem does not have synchronizing communication it can be added and the algorithm presented here can be used as an approximation. I also hope that identifying the sources of error common to many myopic approaches and the general approach I took to address them will help others design better communication algorithms.

CHAPTER 7

CONCLUSION

Cooperative multi-agent systems play an important role in solving many types of problems. However, the complexity inherent to problems with a decentralized view of the world have led researchers to focus on ways to approximate both the representation of the space of coordination behaviors and the search through those behaviors. These approximate approaches have shown themselves capable of handling large real-world problems, but evaluating the quality of the solution is often difficult and frequently relies on comparisons to other approximate solutions.

Recently, a number of researchers have been looking at the success of formal, decision-theoretic models in single agent systems like the Markov Decision Process (MDP) and are attempting to migrate those ideas to systems with two or more agents. The major advantage of the decision-theoretic models is that they have the potential to lead to significantly better solutions due to both a much more accurate representation of the space of coordination behaviors and because the solutions are quantitatively represented and searched as an optimization problem.

Within the multi-agent decision-theoretic community, the research has been in two primary directions. On one side is a direct attempt to extend the MDP to multiple agents. A classic example of this is the Multi-agent Markov Decision Process (MMDP). The assumption used in the MMDP is that each agent observes the entire state of the world. Solving this problem is no more difficult than solving an MDP, but it has the disadvantage of being restricted to solving only fully observable problems or problems with free communication.

The other direction uses the much more general assumption that each agent does not have complete knowledge of the world state. This is more of an extension of Partially Observable Markov Decision Processes (POMDPs) to multiple agents, and the many models in this area are collectively known as distributed POMDPs. This assumption is much more widely applicable to multi-agent systems, but it also significantly increases the complexity of finding an optimal solution, NEXP-complete.

These two dichotomies (approximate vs. decision-theoretic, and fully observable vs. partially observable) are central to understanding the importance of this work. Most of the existing work is found at the extremes of these dichotomies. Heuristic models have the advantage of being easily solved and widely applicable but at the cost of no guarantee on the quality of the solution. The decision-theoretic models often have quality guarantees, but the fully observable models are not widely applicable and the partially observable models are not tractable. The work presented in this dissertation identified several models and algorithms that fall in the middle of these dichotomies and incorporate some of the advantages of each.

The first class of problems I presented is called Transition Independent DEC-MDP (TI-DEC-MDP). This problem is more general than the MMDP because the agents have different and incomplete views of the world. It is also more specific than the DEC-MDP because the interaction between the agents is only through the reward function. The complexity of problems in this class is NP-complete, which is also harder than the MMDP but much easier than the general DEC-MDP. This model naturally represents information collecting problems where the agents do not directly interact, but collect information whose value could be complementary or redundant.

I also investigated other types of interactions between agents. Instead of allowing the agents to interact through the reward function, the agents interact through the transition function using a mechanism I call event-driven interactions. The global reward being maximized is just the sum of the rewards accumulated by each agent

individually, but the actions taken by one agent can affect actions taken by another agent. For example, one rover may carry equipment needed by another and it could deposit the equipment at a prespecified location. The other rover can detect the existence of the equipment when it arrives at that location. This model is called Decentralized MDPs with Event-Driven Interactions.

Identifying a class of problems by itself is not useful unless there exists a good algorithm to find solutions. To this end, I developed an algorithm called the Coverage Set Algorithm (CSA), which returns an optimal solution to the TI-DEC-MDP and the DEC-MDP with Event Driven Interactions. A simple hill-climbing algorithm for these types of problems would converge to a local maxima. The CSA gets around this problem by efficiently searching for all of the local maxima. In general, the number of local maxima is significantly fewer than the number of policies so the algorithm can prune a majority of the policies. The algorithm can also return the best of the local maxima found at any point, making it an anytime algorithm. It has performed very well in experimental work both as an exact algorithm and as an anytime solution.

Neither of the two models I have discussed allow a general form of communication between the agents. This is because finding optimal solutions with communication is significantly more difficult than without, even though certain communication protocols have the same worst-case complexity. I extended the TI-DEC-MDP to allow communication between the agents. I then illustrated how the common myopic assumptions used to deal with communication can lead to undesirable behavior in the agents. Finally, I presented a new algorithm that addressed these sources of error and produced significantly better results. This new algorithm finds the optimal communication policy for fixed domain-level policies.

The work presented here affects the state-of-the-art in multi-agent systems in four primary ways. First, the three classes of problems expand the body of knowledge about multi-agent systems in an area that is currently quite scarce. It identifies

structure in many real multi-agent problems that can be exploited to reduce the complexity of finding optimal solutions, for example the transition and observation independence in Chapter 3. It also grounds problems that were previously solved heuristically in a more formal decision-theoretic framework.

The second effect is that the Coverage Set Algorithm is one of the first practical algorithms to solve optimally a general class of decentralized multi-agent systems. This leads the way for new and innovative solutions for optimal control of multiple agents. Chapter 4.6 shows the results of running the CSA on 600 problems that could not be solved with any other algorithm. 85% of these problems took less than four hours to solve.

The third impact this work will have is to further the work on approximation algorithms for multi-agent systems. While all heuristic algorithms for multi-agent systems are approximations, approximate algorithms grounded in decision theory are quite new and have the potential to significantly outperform the ad-hoc, heuristic ones. However, very few existing approximate algorithms for multi-agent systems provide any guarantees on the solution outcome. The CSA has the distinction of guaranteeing convergence on the optimal solution.

Finally, I presented an analysis of two ways a myopic assumption commonly used to deal with communication can negatively affect the agents' behavior. The Model-Lookahead algorithm I presented in Chapter 6 to address those errors finds the optimal communication policy given a fixed domain policy over a fixed horizon. This is the first algorithm that can efficiently find optimal communication policies for problems of this size and complexity. My experimental results show that in some cases it can provide a significant improvement in solution quality without also taking significantly longer to compute.

There are a number of interesting future directions I plan to pursue with this work. First, I presented two different algorithms, one that finds optimal domain

policies given no communication and another to find optimal communication policies given fixed domain policies. Ideally, these two algorithms could be merged to find globally optimal domain and communication policies, though a way to do that is not immediately apparent. A first step may be to extend the CSA to allow arbitrary communication policies instead of restricting it to no communication.

It is also time to start generating a testbed of problems designed to facilitate comparing the various optimal and approximate distributed POMDP algorithms. This testbed should include a range of problem types from transition and observation independent, to structured agent interactions like event driven interactions, to general distributed POMDPs that are not the topic of this dissertation. It should also include a range of problem sizes, from toy problems just small enough to solve optimally (perhaps with weeks of processing) to real-world sized problems with no hope of an optimal solution to compare to.

There are two comparisons that are particularly interesting to pursue. First is the recent work by Nair et al. [52] on Networked Distributed POMDPs. In that work they present an approximate algorithm that combines their distributed POMDP algorithm JESP with a DCOP algorithm. They also present an optimal algorithm that essentially performs brute force policy search. It would be interesting to implement and compare the CSA/DCOP combination mentioned in Chapter 4 to their approach.

The second comparison is the Model-Lookahead method of generating communication policies to work done by Xuan et al. [77, 78]. These approaches tackle the problem from different perspectives. Model-Lookahead starts with no communication and adds it in, while they start with complete communication and pare it down. My sense is that these two approaches will each perform better on problems where the globally optimal solution is closer to their starting points.

Perhaps the best use of distributed POMDPs in real world problems will come from a merger with the more traditional approaches. In a hybrid system the tra-

ditional approaches can handle the high complexity of the real world problem and the distributed POMDPs can be used to optimize a small yet critical piece of the whole. Existing frameworks like GPGP/TAEMS [41] and STEAM [69] already use MDPs to optimize components of the coordination process, and RMTDP [49] mixes a BDI approach with POMDPs. Perhaps these models can be extended to incorporate *distributed*, decision-theoretic components to improve their performance.

APPENDIX

DERIVING THE MODEL-LOOKAHEAD EQUATION

This Appendix shows how Equation 6.6 was derived. This is for two agents, i and j. The current local states for the agents are s_i and s_j. I always use s'_i and s'_j for the next states. The previous synchronized world state is $\langle s_i^0, s_j^0 \rangle$, which happened t steps earlier. When the agents communicate in s_i, s_j, the new synchronized world state becomes $\langle s_i, s_j \rangle$, and $t = 0$. The agents take domain-level actions based on an optimal policy assuming no future communication, $\langle \pi_i^{\langle s_i^0, s_j^0 \rangle}, \pi_j^{\langle s_i^0, s_j^0 \rangle} \rangle$. C is the cost for communicating. Comm$'$ is the set of world states in which the agents will communicate. I will mention how this is computed at the end.

The global value function assuming no communication is:

$$V(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t) = \sum_{s'_i, s'_j} P(s'_i | s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s'_j | s_j, \pi_i^{\langle s_i^0, s_j^0 \rangle}) \left[R_i(s'_i) + R_j(s'_j) + R(s'_i, s'_j) + V(s'_i, s'_j, \langle s_i^0, s_j^0 \rangle, t+1) \right]$$

Note: The derivation is the same if you include a discount factor.

A superscript next to V represents the horizon in which communication is considered:

- $V^0(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t)$ is the expected value of never communicating.
- $V^0(s_i, s_j, \langle s_i, s_j \rangle, 0) - C$ is the expected value of communicating immediately and never again.

- $V^1(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t)$ is the expected value of not communicating in the current step, allowing communication if appropriate in the next step, and never communicating after the next step.

- $V^1(s_i, s_j, \langle s_i, s_j \rangle, 0) - C$ is the expected value of communicating immediately, allowing communication if appropriate in the next step, and never communicating after the next step.

- $V^h(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t)$ is the expected value of not communicating in the current step, allowing communication where appropriate for the next h steps, and never communicating after that.

- $V^h(s_i, s_j, \langle s_i, s_j \rangle, 0) - C$ is the expected value of communicating immediately, allowing communication where appropriate for the next h steps, and never communicating after that.

I will give an inductive definition of the value allowing communication over a horizon. First is the base case, $h = 0$. $V^0(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t) = V(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t)$. This is just the global value function defined above.

Assume that $V^{h-1}(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t)$ is the expected value of not communicating in the current step, allowing communication for the next $h - 1$ steps, and never communicating after that. Also assume that $V^{h-1}(s_i, s_j, \langle s_i, s_j \rangle, 0) - C$ is the expected value of communicating immediately, allowing communication for the next $h-1$ steps, and never communicating after that.

To compute $V^h(\cdot)$ we divide the next possible world states into two categories, those in which the agents would choose to communicate, Comm$'$, and those in which they would not, $\neg$Comm$'$. For both cases, it is the sum of the probability that the agents transition to that world state times the immediate rewards plus the expected value allowing future communication up to the original horizon.

$$V^h(s_i, s_j, \langle s_i^0, s_j^0 \rangle, t) =$$

$$\sum_{s_i', s_j' \in \text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[R_i(s_i') + R_j(s_j') + R(s_i', s_j') + V^{h-1}(s_i', s_j', \langle s_i', s_j' \rangle, 0) - C \right]$$

$$+ \sum_{s_i', s_j' \in \neg\text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[R_i(s_i') + R_j(s_j') + R(s_i', s_j') + V^{h-1}(s_i', s_j', \langle s_i^0, s_j^0 \rangle, t+1) \right].$$

In the next several steps I transform the equation. First I separate the rewards from the expected values:

$$= \sum_{s_i', s_j' \in \text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[R_i(s_i') + R_j(s_j') + R(s_i', s_j') \right]$$

$$+ \sum_{s_i', s_j' \in \text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[V^{h-1}(s_i', s_j', \langle s_i', s_j' \rangle, 0) - C \right]$$

$$+ \sum_{s_i', s_j' \in \neg\text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[R_i(s_i') + R_j(s_j') + R(s_i', s_j') \right]$$

$$+ \sum_{s_i', s_j' \in \neg\text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[V^{h-1}(s_i', s_j', \langle s_i^0, s_j^0 \rangle, t+1) \right].$$

Then I combine the rewards and add/subtract two new components, (A.4)/(A.5) and (A.6)/(A.7):

$$= \sum_{s_i', s_j'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[R_i(s_i') + R_j(s_j') + R(s_i', s_j') \right] \quad \text{(A.1)}$$

$$+ \sum_{s_i', s_j' \in \text{Comm}'} P(s_i'|s_i, \pi_i^{\langle s_i^0, s_j^0 \rangle}) P(s_j'|s_j, \pi_j^{\langle s_i^0, s_j^0 \rangle}) \left[V^{h-1}(s_i', s_j', \langle s_i', s_j' \rangle, 0) - C \right] \quad \text{(A.2)}$$

$$+ \sum_{s'_i, s'_j \in \neg\text{Comm}'} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.3)}$$

$$+ \sum_{s'_i, s'_j} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.4)}$$

$$- \sum_{s'_i, s'_j} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.5)}$$

$$+ \sum_{s'_i, s'_j \in \text{Comm}'} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.6)}$$

$$- \sum_{s'_i, s'_j \in \text{Comm}'} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right]. \quad \text{(A.7)}$$

Next I combine (A.1) with (A.4), (A.2) with (A.7), and (A.3) with (A.6) and then with (A.5):

$$= \sum_{s'_i, s'_j} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[R_i(s'_i) + R_j(s'_j) + R(s'_i, s'_j) + V(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.8)}$$

$$+ \sum_{s'_i, s'_j \in \text{Comm}'} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s'_i, s'_j\rangle, 0) - C - V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right]$$

$$\sum_{s'_i, s'_j} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1) - V(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right]$$

Line (A.8) is simply the expected value with zero communication.

$$= V(s_i, s_j, \langle s_i^0, s_j^0\rangle, t) \quad \text{(A.9)}$$

$$+ \sum_{s'_i, s'_j \in \text{Comm}'} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s'_i, s'_j\rangle, 0) - C - V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.10)}$$

$$+ \sum_{s'_i, s'_j} P(s'_i|s_i, \pi_i^{\langle s_i^0, s_j^0\rangle}) P(s'_j|s_j, \pi_j^{\langle s_i^0, s_j^0\rangle}) \left[V^{h-1}(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1) - V(s'_i, s'_j, \langle s_i^0, s_j^0\rangle, t+1)\right] \quad \text{(A.11)}$$

This is Equation 6.6. I transformed the equation in this way because it is much easier to use. This transformed equation also demonstrates how to compute the communication policy for the next step. We want to find a joint communication policy for the next step that maximizes this value function. Lines (A.9) and (A.11) do not depend on the communication policy for the next step, so we just need to maximize line (A.10). This is what the *Model* approach does.

BIBLIOGRAPHY

[1] Abdallah, Sherief, and Lesser, Victor. Learning the game of distributed task allocation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems* (2006).

[2] Barto, Andrew G., and Mahadevan, Sridhar. Recent advances in hierarhical reinforcement learning. *Special Issue on RL: Discrete Event Systems Journal 13* (2003), 41–77.

[3] Becker, Raphen, Zilberstein, Shlomo, and Lesser, Victor. Decentralized Markov decision processes with event-driven interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (New York City, New York, July 2004), vol. 1, ACM Press, pp. 302–309.

[4] Becker, Raphen, Zilberstein, Shlomo, Lesser, Victor, and Goldman, Claudia V. Transition-independent decentralized Markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems* (Melbourne, Australia, July 2003), ACM Press, pp. 41–48.

[5] Becker, Raphen, Zilberstein, Shlomo, Lesser, Victor, and Goldman, Claudia V. Solving transition independent decentralized MDPs. *Journal of Artificial Intelligence Research 22* (2004), 423–455.

[6] Bernstein, Daniel S., Givan, Robert, Immerman, Neil, and Zilberstein, Shlomo. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research 27*, 4 (November 2002), 819–840.

[7] Beynier, Aurelie, and Mouaddib, Abdel-Illah. A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems* (2005).

[8] Beynier, Aurelie, Mouaddib, Abdel-Illah, and Zilberstein, Shlomo. Solving efficiently DEC-MDPs with temporal constraints. In *PDMIA* (2005).

[9] Boutilier, Craig. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (1999), pp. 478–485.

[10] Bowling, Michael. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17*, Lawrence K. Saul, Yair Weiss, and Léon Bottou, Eds. MIT Press, Cambridge, MA, 2005, pp. 209–216.

[11] Bratman, M. E. *Intentions, Plans, and Practical Reason.* Harvard University Press, 1987.

[12] Chades, Iadine, Scherrer, Bruno, and Charpillet, Francois. A heuristic approach for solving decentralized-POMDP: Assessment on the pursuit problem. In *SAC* (2002).

[13] Chia, Mike H., Neiman, Daniel E., and Lesser, Victor R. Poaching and distraction in asynchronous agent activities. In *Proceedings of the Third International Conference on Multi-Agent Systems* (1998), pp. 88–95.

[14] Cohen, Philip R., and Levesque, Hector J. Intention is choice with commitment. *Artificial Intelligence* (1990), 213–261.

[15] Decker, Keith, and Lesser, Victor. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour 2* (January 1993), 215–234.

[16] Denardo, Eric V. On linear programming in a Markov decision problem. *Management Science 16*, 5 (1970), 281–288.

[17] Dolgov, Dmitri, and Durfee, Edmund. Graphical models in local, asymmetric multi-agent Markov decision processes. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (2004).

[18] Doshi, Prashant, and Gmytrasiewicz, Piotr J. A particle filtering based approach to approximating interactive POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (2005).

[19] Durfee, Edmund. Practically coordinating. *AI Magazine 20*, 1 (Spring 1999), 99–116.

[20] Emery-Montemerlo, Rosemary, Gordon, Geof, and Schneider, Jeff. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (2004).

[21] Emery-Montemerlo, Rosemary, Gordon, Geof, and Schneider, Jeff. Game theoretic control for robot teams. In *ICRA* (2005).

[22] Erman, L.D., Hayes-Roth, F., Lesser, V.R., and Reddy, D.R. The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys 12*, 2 (June 1980), 213–253.

[23] Ghavamzadeh, Mohammad, and Mahadevan, Sridhar. A multiagent reinforcement learning algorithm by dynamically merging Markov decision processes. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems* (Bologna, Italy, 2002), ACM Press.

[24] Ghavamzadeh, Mohammad, and Mahadevan, Sridhar. Learning to communicate and act in cooperative multiagent systems using hierarchical reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (New York, July 2004), ACM Press, pp. 1114–1121.

[25] Gmytrasiewicz, Piotr J., and Doshi, Prashant. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research* (2005).

[26] Gmytrasiewicz, Piotr J., and Durfee, Edmund H. Rational communication in multi-agent environments. *Autonomous Agents and Multi Agent Systems Journal 4*, 3 (2001), 233–272.

[27] Goldman, Claudia V., and Zilberstein, Shlomo. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems* (Melbourne, Australia, July 2003), ACM Press, pp. 137–144.

[28] Goldman, Claudia V., and Zilberstein, Shlomo. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research 22* (2004), 143–174.

[29] Grosz, Barbara J., and Kraus, Sarit. Collaborative plans for complex group action. *Artificial Intelligence 86*, 2 (1996), 269–357.

[30] Guestrin, Carlos, and Gordon, Geoffrey. Distributed planning in hierarchical factored MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence* (2002).

[31] Guestrin, Carlos, Koller, Daphne, Parr, Ronald, and Venkataraman, Shobha. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research 19* (2003), 399–468.

[32] Guestrin, Carlos, Venkataraman, Shobha, and Koller, Daphne. Context specific multiagent coordination and planning with factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (Edmonton, Canada, July 2002), pp. 253–259.

[33] Hansen, Eric, Bernstein, Daniel S., and Zilberstein, Shlomo. Dynamic programming for partially observable stochastic games. *Proceedings of the Ninteenth National Conference on Artificial Intelligence* (July 2004), 709–715.

[34] Hansen, Eric, and Zilberstein, Shlomo. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence 129* (2001), 35–62.

[35] Horvitz, Eric J. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence* (Minneapolis, MN, August 1988), Morgan Kaufmann, pp. 111–116.

[36] Howard, Ronald A. *Dynamic Programming and Markov Processes*. MIT Press and Wiley, New York, 1960.

[37] Howard, Ronald A. Information value theory. *IEEE Transactions on Systems Science and Cybernetics SSC–2*, 1 (1966), 22–26.

[38] Hsu, Kai, and Marcus, Steven I. Decentralized control of finite state Markov processes. *IEEE Transactions on Automatic Control 27*, 2 (1982), 426–431.

[39] Jennings, Nicholas R. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence 75*, 2 (1995), 195–240.

[40] Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence 101* (1998), 99–134.

[41] Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., NagendraPrasad, M., Raja, A., Vincent, R., Xuan, P., and Zhang, X.Q. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems 9* (2004), 87–143.

[42] Levesque, Hector J., Cohen, Philip R., and Nunes, Jose H. T. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (1990), pp. 94–99.

[43] Liu, JyiShane, and Sycara, Katia P. Exploiting problem structure for distributed constraint optimization. In *Proceedings of the First International Conference on Multi-Agent Systems* (1995), pp. 246–253.

[44] Madani, Omid, Hanks, Steve, and Condon, Anne. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (1999), pp. 541–548.

[45] Mailler, Roger, and Lesser, Victor. Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (2004), 438–445.

[46] Manne, Alan S. Linear programming and sequential decisions. *Management Science 6* (1960), 259–268.

[47] Modi, Pragnesh Jay, Shen, Wei-Min, Tambe, Milind, and Yokoo, Makoto. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems* (2003), pp. 161–168.

[48] Mundhenk, Martin, Goldsmith, Judy, Lusena, Christopher, and Allender, Eric. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM 47*, 4 (2000), 681–720.

[49] Nair, Ranjit, and Tambe, Milind. Hybrid BDI-POMDP framework for multiagent teaming. *Journal of Artificial Intelligence Research* (2005), 367–420.

[50] Nair, Ranjit, Tambe, Milind, Roth, Maayan, and Yokoo, Makoto. Communication for improving policy computation in distributed POMDPs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (2004).

[51] Nair, Ranjit, Tambe, Milind, Yokoo, Makoto, Pynadath, David, and Marsella, Stacy. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (2003).

[52] Nair, Ranjit, Varakantham, Pradeep, Tambe, Milind, and Yokoo, Makoto. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (2005), pp. 133–139.

[53] Ooi, James M., and Wornell, Gregory W. Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proceedings of the 35th Conference on Decision and Control* (1996), pp. 293–298.

[54] Papadimitriou, Christos H., and Tsitsiklis, John. On the complexity of designing distributed protocols. *Information and Control 53* (1982), 211–218.

[55] Papadimitriou, Christos H., and Tsitsiklis, John. Intractable problems in control theory. *SIAM Journal on Control and Optimization 24*, 4 (1986), 639–654.

[56] Papadimitriou, Christos H., and Tsitsiklis, John. The complexity of Markov decision processes. *Mathematics of Operations Research 12*, 3 (1987), 441–450.

[57] Pearl, Judea. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, second ed. Morgan Kaufmann, 1988.

[58] Peshkin, Leonid, Kim, Kee-Eung, Meuleau, Nicolas, and Kaelbling, Leslie P. Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, 2000), Morgan Kaufmann, pp. 489–496.

[59] Pynadath, David V., and Tambe, Milind. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research 16* (2002), 389–423.

[60] Roth, Maayan, Simmons, Reid, and Velosa, Manuela. Decentralzied communication strategies for coordinated multi-agent policies. In *MRS* (2005).

[61] Seuken, Sven, and Zilberstein, Shlomo. Formal models and algorithms for decentralized control of multiple agents. *University of Massachusetts, Amherst Technical Report UM-CS-2005-068* (2005).

[62] Shen, Jiaying, Becker, Raphen, and Lesser, Victor. Agent interaction in distributed POMDPs and its implications on complexity. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems* (2006).

[63] Shen, Jiaying, Lesser, Victor, and Carver, Norman. Minimizing communication cost in a distributed Bayesian network using a decentralized MDP. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems* (Melbourne, Australia, July 2003), ACM Press, pp. 678–685.

[64] Shen, Jiaying, Zhang, Xiaoqin, and Lesser, Victor. Degree of local cooperation and its implication on global utility. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (New York, New York, July 2004).

[65] Shoham, Yoav, Powers, Rob, and Grenager, Trond. If multi-agent learning is the answer, what is the questions? To appear in *Special Issue of Artificial Intelligence* (2006).

[66] Smith, Reid G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers C-29*, 12 (December 1980), 1104–1113.

[67] Szer, D., Charpillet, F., and Zilberstein, S. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence* (2005).

[68] Tambe, M., Bowring, E., Jung, H., Kaminka, G., Maheswaran, R., Marecki, J., Modi, P.J., Nair, R., Okamoto, S., Pearce, J.P., Paruchuri, P., Pynadath, D., Scerri, P., Schurr, N., and Varakantham, P. Conflicts in teamwork: Hybrids to the rescue. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems* (2005).

[69] Tambe, Milind. Towards flexible teamwork. *Journal of Artificial Intelligence Research 7* (1997), 83–124.

[70] Tambe, Milind, and Zhang, Weixiong. Towards flexible teamwork in persistent teams: Extended report. *Journal of Autonomous Agents and Multi-agent Systems 3* (2000), 159–183.

[71] Theocharous, Georgios, and Mahadevan, Sridhar. Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation* (Washington, D.C., May 2002).

[72] Varakantham, Pradeep, Nair, Ranjit, Tambe, Milind, and Yokoo, Makoto. Winning back the CUP for distributed POMDPs: Planning over continuous belief spaces. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems* (2005).

[73] Wagner, Thomas, Guralnik, Valerie, and Phelps, John. TAEMS agents: Enabling dynamic distributed supply chain management. *Journal of Electronic Commerce Research and Applications* (2003).

[74] Wang, X., and Sandholm, T. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Advances in Neural Information Processing Systems* (2002).

[75] Washington, Rich, Golden, K., Bresina, J., Smith, D., Anderson, C., and Smith, T. Autonomous rovers for Mars exploration. In *Proceedings of the IEEE Aerospace Conference* (1999).

[76] Xuan, Ping, and Lesser, Victor. Multi-agent policies: From centralized ones to decentralized ones. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems* (2002), ACM Press, pp. 1098–1105.

[77] Xuan, Ping, Lesser, Victor, and Zilberstein, Shlomo. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents* (Montreal, Canada, January 2001), ACM Press, pp. 616–623.

[78] Xuan, Ping, Lesser, Victor, and Zilberstein, Shlomo. Modeling cooperative multiagent problem solving as decentralized decision processes. *Submitted for publication* (2004).

[79] Xuan, Ping, and Lesser, Victor R. Handling uncertainty in multi-agent commitments. In *University of Massachusetts, Amherst, Technical Report 1999-05* (1999).

[80] Yokoo, Makoto, and Durfee, Edmund H. Distributed constraint optimization as a formal model of partially adversarial cooperation. Tech. Rep. CSE-TR-101-91, The University of Michigan, Ann Arbor, Michigan, 1991.

[81] Zilberstein, Shlomo, Washington, Rich, Bernstein, Daniel S., and Mouaddib, A. I. Decision-theoretic control of planetary rovers. In *Plan-Based Control of Robotic Agents*, M. Beetz et al., Eds., no. 2466, in Lecture Notes in Artificial Intelligence. Springer, 2002, pp. 270–289.