2-2012

Decision-Theoretic Meta-reasoning in Partially Observable and Decentralized Settings

Alan Scott Carlin
University of Massachusetts Amherst, acarlin@cs.umass.edu

DECISION-THEORETIC META-REASONING IN PARTIALLY OBSERVABLE AND DECENTRALIZED SETTINGS

A Dissertation Presented

by

ALAN CARLIN

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

DOCTOR OF PHILOSOPHY

February 2012

Department of Computer Science

DECISION-THEORETIC META-REASONING IN PARTIALLY OBSERVABLE AND DECENTRALIZED SETTINGS

A Dissertation Presented

by

ALAN CARLIN

Approved as to style and content by:

ACKNOWLEDGMENTS

I would first like to thank my advisor, Shlomo Zilberstein. Shlomo has supplied an endless repository of knowledge about the field and has taught me so much about how to both conduct and present research. Shlomo's guidance has provided the perfect balance of research direction and freedom. In addition, it has been a pleasure to be part of the Resource Bounded Reasoning (RBR) lab at the University of Massachusetts. The time spent in Shlomo's lab has been fun as well as fruitful.

I would also like to thank my previous advisor at Tufts, Jim Schmolze, who was taken from this world too soon. Jim introduced me to the POMDP, taught me LISP, and his enthusiasm for the field was always contagious.

I would like to thank Victor Lesser. Victor's advice has proven invaluable, he has always been able to view research questions in just the right way, or to provide just the right context. I would like to thank Rod Grupen and Senay Solak for serving on my committee and for discussions which have contributed to this thesis.

I would like to thank the friends I have met while at UMass. I didn't come here intending to make friends, it just happened. In addition to being my role model, Chris Amato introduced me to the lab, to Northampton, and to his fantasy baseball league. Marek Petrik helped me move. Marek, I'm sorry that my washing machine almost killed you. Hala Mostafa has been someone with whom I could share both triumphs and tribulations. Hala, if not for you I may have survived my time in Amherst, but I would not have retained my sense of humor. Siddharth Srivastava tried to teach me disc golf but gave up, but then he tried racquetball and succeeded. Thus he has been a true model of persistence. I'd like to thank my travel companion, Akshat Kumar, who it turns out speaks French better than I, and who has a sixth sense for finding pizza and fondue... And William Yeoh – it seems we never really got out as a lab until you arrived.

Post grad school, I would like to thank Nathan Schurr, who taught me how to do industrial research, and who has been my friend as well.

I would like to thank those who have impacted my life before grad school. David Lotierzo, Albert Lamaute, Bob McMurray, Julie Weber, each one of you will always be my friend. I would like to thank my colleagues and friends after college but before grad school; Dennis Roux (who made my first real job fun), Steve Haron (for 2pm coffee), Lisa Basile (my fashion consultant), Ryan Parks (my basketball coach), and Chris Burian (my sci-fi guru).

Finally I would like to thank my family. Stephen and Debbie Carlin, I would like to thank for all of the long drives in the station wagon. And to my parents, Michael and Maxine Carlin, thank you for standing by me for all of these years.

ABSTRACT

DECISION-THEORETIC META-REASONING IN PARTIALLY OBSERVABLE AND DECENTRALIZED SETTINGS

FEBRUARY 2012

ALAN CARLIN

B.A., CORNELL UNIVERSITY

M.S., TUFTS UNIVERSITY

PhD, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

This thesis examines decentralized meta-reasoning. For a single agent or multiple agents, it may not be enough for agents to compute correct decisions if they do not do so in a timely or resource efficient fashion. The utility of agent decisions typically increases with decision quality, but decreases with computation time. The reasoning about one's computation process is referred to as meta-reasoning. Aspects of meta-reasoning considered in this thesis include the reasoning about how to allocate computational resources, including when to stop one type of computation and begin another, and when to stop all computation and report an answer. Given a computational model, this translates into computing how to schedule the basic computations that solve a problem. This thesis constructs meta-reasoning strategies for the purposes of monitoring and control in multi-agent settings, specifically settings that can be modeled by the Decentralized Partially Observable Markov Decision Process (Dec-POMDP). It uses decision theory to optimize computation for efficiency in time and space in communicative and non-communicative decentralized settings. Whereas base-level reasoning describes the optimization of actual agent behaviors, the meta-reasoning strategies produced by this thesis dynamically optimize the computational resources which lead to the selection of base-level behaviors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

This thesis examines decentralized meta-reasoning. For a single agent or multiple agents, it may not be enough for agents to compute correct decisions if they do not do so in a timely or resource efficient fashion. Russell and Wefald describe the situation: “first, real agents have only finite computational power, second, they don’t have all the time in the world... Typically, the utility of an action will be a decreasing function of time” [111]. By the time a decision is made the conditions leading up to it may have changed.

The reasoning about one’s computation process is referred to as “meta-reasoning”. Aspects of meta-reasoning discussed in this thesis include the reasoning about how to allocate computational resources, including when to stop one type of computation and begin another, and when to stop computation and report an answer. Given a computational model, this translates into computing how to schedule the basic computations that solve a problem.

This thesis considers meta-reasoning for the purposes of monitoring and control in multi-agent settings. It examines how computation may be optimized for efficiency in time and space in these settings. Whereas “base-level” reasoning (sometimes referred to as “domain-level” or “object-level” reasoning) describes the optimization of actual agent behaviors, in this thesis meta-reasoning describes the dynamic optimization of computational resources which leads to the selection of domain-level behaviors. Figure 1.1 shows the Cox and Raja model of a meta-reasoning system [39], which is adopted for this thesis. The bottom-left of the figure shows that each agent is capable of Doing a set of actions. At the object level of reasoning, actions are selected and the results of actions are perceived. But the reasoning process itself may affect ground-level performance, for example if reasoning is not performed in a timely fashion. Therefore, at the meta-level, the state of object-level reasoning is monitored, and the meta-reasoning module may control the reasoning process.[1] In a distributed system, the ground-level, reasoning, and meta-reasoning process is happening among many agents at once. Under the Cox and Raja architecture, agents may coordinate their performance at the meta-level.

This thesis contributes to three aspects of this monitoring and control of system components. For each contribution, we apply decision-theory [146] to meta-reasoning.

- The first contribution addresses decentralized meta-reasoning using *performance profiles* [40]. Performance profiles define conditional probabilities of the quality distribution of algorithms over time, and they inform the meta-reasoning components of the system. These profiles are formally defined in Section 2.1.4, where it will be shown that performance profiles have been an area of study for single-agent meta-reasoning. However despite the single-agent work, *there is no current model of decentralized meta-reasoning using performance profiles*. This thesis contributes such a model by extending single-agent models to the multiple agent case. We introduce the Decentralized monitoring problem (DMP). Furthermore this thesis shows the relationship of the resulting model to the decentralized Markov Decision Process (Dec-MDP) model.

[1] If the meta-level process itself requires resources to execute, this may create the need for "meta-meta-level" reasoning, and so on. However, when the computational overhead of meta-reasoning is small, as will be the case in this thesis, there is no need for monitoring and control of the meta-level.

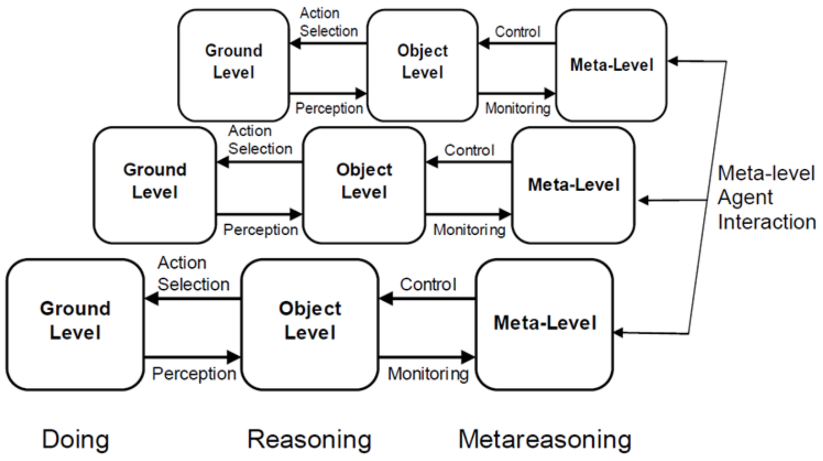


Figure 1.1. Cox and Raja's representation of distributed meta-reasoning [39].

- The second contribution develops a strategy to address the complexity of decentralized reasoning. We examine the Decentralized Partially Observable Decision Process (Dec-POMDP) model, a decentralized model of multi-agent decision making which is NEXP-hard [17]. In Chapter 2 it will be shown that one source of the complexity is that each agent must reason over all possible histories of the other agents, which includes all possible observation sequences that all agents have made about the environment. This thesis introduces a method called *observation compression*, which allows a Dec-POMDP planner to reason over only a subset of the observations while bounding the loss of information due to the compression.

- The third contribution involves communication decisions in decentralized models. Specifically, this thesis examines two models called Dec-MDP-Comm and Dec-POMDP-Comm, in which agents are able to communicate their current state (for Dec-MDP-Comm) or observation histories (for Dec-POMDP-Comm). When deciding on whether to communicate in the Dec-POMDP-Comm model, each agent must reason about a large number of possible decision-observation sequences for the other agents, such reasoning is time and space consuming. This thesis introduces a compression method for reasoning about communication.

  Note: This thesis uses examples and experiments corresponding to two agent models. The underlying notation and algorithms extend to more than two agents, except where noted.

  The next section of this introduction will review domains where agents must perform meta-reasoning. After this, approaches to meta-reasoning will be briefly introduced (and elaborated in Chapter 2). Then, Section 1.3 will briefly review the approaches in artificial intelligence to handle these challenges. Finally, Section

1.4 will summarize the organization of this thesis and present the contributions of this work.

1.1 Illustrative examples

Illustrations of domains that require efficient computation include:

- Games such as chess and Go.
- Medical diagnosis, where timely treatment is necessary.
- Multi-sensor search, where sensors or robots equipped with sensors reason about how to best obtain data.

Chess and Go are games with 10^{47} and 10^{171} states respectively, which have never been solved completely. Typical matches involve a clock and thus iterative deepening-based searches in games reason over both the base-level quality of solutions while also managing time [121]. Medical diagnosis was studied by Horvitz in his work on flexible computation, in order to produce both high quality and timely hypotheses [64]. Additional computation time may result in a better diagnosis, but also a deterioration of patient condition, these two conflicting demands are weighed against one another. One multi-agent extension to medical diagnosis is emergency response, where multiple agents must coordinate to quickly address a disaster [68].

Multi-sensor domains include assistive living scenarios, where sensor networks are being deployed [53]. Other sensor network initiatives include the Collaborative Adaptive Sensing of the Atmosphere (CASA) effort and the Automatic Negotiating Teams (ANTs) project. CASA uses a set of radars to sensing and predicting weather [6, 79, 158], and ANTs uses a set of radars to track targets over a two dimensional space [74]. Pursuit-evasion problems typically address situations where sensors are mobile, such as on robots. Room clearing, in which robots assist humans in searching a building, is an active area of research within pursuit-evasion [25].

We use a multi-robot pursuit-evasion example to illustrate the difference between base-level reasoning and meta-level reasoning. As motivation, we consider an assistive living scenario. Suppose a team of robot assistants searches for a person's wallet and keys, with a separate robot being assigned each task. The search may be *decentralized*, as each robot may not be in full communication with its peers. At the base level, each robot must plan its search, including its movements and its path. Coordination may be required for some parts of the plan, for instance if one robot needs to lift an object so that the other robot can search under it.

At the meta-level, each robot may monitor its own planning process. For example, if planning requires that several areas of several rooms must each be visited, and furthermore if robots plans are independent of each other, this creates an instance of the Traveling Salesman Problem (TSP) for each robot. TSP is known to be NP-complete, and thus optimal solutions become time consuming as the number of locations increases. However anytime solutions are possible, for example the tour improvement algorithm by Lin and Kernighan starts with an initial tour, and then improves it over time [76]. By monitoring the progress of this computation, a robot may decide in the process of planning that its current route is "good enough", and that the benefits of continuing to compute a better route would not justify the additional planning time. Coordination may be beneficial on the meta-level, for cases where the commencement of one robot's plan affects the utility of the other agent's plan. Recalling the example, a wallet and keys may be more useful in conjunction than separately. If so, the fact that one robot has ceased planning to find the wallet and started executing its plan could affect whether the second robot should also cease planning and start executing its search for the set of keys.

This example illustrates the three challenges within multi-agent meta-reasoning addressed by the contributions of this thesis. First, there is the challenge of representing the progress and predicted performance of the distributed computation. Given performance profiles of TSP, a suitable representation should describe how the two TSP problems relate to each other (e.g., one item may have priority, or one robot may have a more stringent time limit), while allowing for computationally tractable meta-reasoning. Each agent must decide how to perform its anytime computation, and each agent must also decide when to stop computation of its search plan and begin execution of it. Each agent must do this in an environment where the other agents are making similar decisions about computation. Second, there is the challenge of partial observability. It may be the case that each agent is not fully aware of its planning progress. Agents may observe their planning progress, such as by observing the currently computed tour quality. However, the quality of the current tour should be relative to the quality of the optimal tour, which is not known; therefore tour quality is only partially observable. In a multi-agent environment, finding optimal solutions may become complicated, and the second challenge is how to mitigate this complexity with respect to the number of possible observations. The third challenge involves how to reason about the progress of the other agents (and how to reason about the other agents' reasoning process) in a manner that is efficient in both time and space. Optionally, each agent may choose to monitor computation, of both itself and other agents. If it monitors its own computation, this presents a decision of when to do so. If it monitors the computation of others, this presents the challenge of defining a communication mechanism to perform the monitoring, and deciding when to communicate.

1.2 Representing and making meta-reasoning decisions

This thesis studies the problem representation of, and the solution to, meta-reasoning. That is, we divide meta-reasoning into two tasks. First, there is the task of modeling the meta-reasoning process. In advance, a real-world agent should represent its base-level and computational capabilities in the model. Second, there is the task of producing meta-reasoning policies, given the model. At computation time, the agent should query these solution policies to determine how to act next.

Both the modeling and the generation of solutions present challenges, and below we summarize the challenges relevant to this thesis:

- **Model representation**: This thesis studies the representation of both multi-agent and single-agent meta-reasoning. As will be seen, aspects that should be modeled include:
  - *Computation options*: What is meant by computation? What computational options does an agent have? In the model, how should an agent select its computation (or select to stop and decide to act on the current results of its computation)? (Chapter 3)
  - *Monitoring options*: How do we model uncertainty of computational state? This includes both uncertainty of current computational state as well as uncertainty of how this state will change over time. Furthermore how do we model the ability to monitor computational state to reduce that uncertainty? (Chapter 3)
  - *Communication options*: How can agents become aware of the progress of other agent's computations? (Chapters 3 and 5)

  - *Time versus quality trade-off*: How does the model account for the real-world intuition that longer computation time may be undesirable? (Chapters 3 and 4)
  - *Space versus quality trade-off*: How does the model account for the real-world intuition that the agent may have limited space (memory, etc.) to perform computation? (Chapter 4)

- **Solution methods**: This thesis will also study the problem of producing high quality meta-reasoning policies. Issues regarding the solution policies include:
  - *Performance guarantees*: The meta-level policy should be evaluated with respect to optimality of both the resulting base-level decisions and the computation sequence. Russell and Subramanian refer to this evaluation as *bounded optimality* [110]. (Chapter 3)
  - *Resource usage*: It may be the case that a processor has a finite amount of memory to work with, in which case a computation sequence may be evaluated for how it manages this finite memory bound. (Chapter 4)
  - *Myopia of agents*: Each agent must reason in an environment where other agents are reasoning as well. We refer to a *myopic* meta-level policy as a meta-level policy which does not account for the reasoning capabilities of other agents. (Chapters 3 and 5)

1.3 Meta-reasoning approaches in artificial intelligence

This section briefly describes historical approaches to meta-reasoning; we defer a detailed discussion of the state-of-the-art to Chapter 2. The idea that computation itself must be reasoned about was introduced by Herbert Simon in the 1950's [123]. Simon distinguishes "substantive rationality" as studied (according to Simon) by economists, and which accounts only for goals, from “procedural rationality”, as studied by psychologists, and accounts for cognitive processes [124]. For procedural rationality, computing the rational thing to do may require a large amount of overhead, which can become prohibitive. This leads to the concept of satisficing (as opposed to optimizing), or producing solutions that are good enough [123]. I.J. Good distinguished “type I” from “type II” rationality”, the latter of which weighs the trade-off of solution quality for deliberation time [52].

Horvitz, Russell, Dean and Boddy, Zilberstein, and others have formalized aspects of meta-reasoning for single agents in artificial intelligence [111, 110, 63, 65, 19, 156, 55]. The input to these decision theoretic algorithms is a *performance profile*, or a description of how an algorithm is expected to perform given an allotment of time or other computational resources. Computation reflects running the base-level algorithm for a finite amount of time. The meta-reasoning algorithms compute an *expected value of computation*, which specifies the utility of a particular allotment of computational resources to the agent. Utility is expressed as a function of a computational state, a domain-level state, and the time. Typically, utility will decrease with time, so the agent must decide whether to execute its current choice of domain-level action at the current time, or to continue computation, and thus perhaps choose a better domain-level action, but executing it at a later time.

Hansen and Zilberstein have accounted for uncertainty (for a single agent) in the observation of computational progress [55]. They termed their formulation of the meta-reasoning problem as the “monitoring problem”. In this formulation, the agent reasons over a *conditional performance profile* which specifies the probability of a future computational state of progress, given the current state. The agent then decides whether to stop and take its current solution, continue deliberation, or *monitor* its current progress, thus enabling it to exploit the conditional performance profile to make a better decision.

The result is non-optimal and only applies to a single agent, this thesis will show how boundedly optimal solutions may be produced for the single agent, and how the work can be extended to multiple agents.

For multi-agent situations, the single-agent approaches described above may be used to construct agent behavior by having each agent consider the other (external) agents as part of the environment, that is, having each agent reason about a world where the behavior of the other agents is stochastic. Throughout this thesis, we will refer to this approach as the "myopic approach". However, the myopic approach may not produce optimal solutions. External agents do not behave stochastically, instead they reason as well. The myopic approach may miss opportunities for agents to coordinate. In this thesis we will consider coordination through base-level actions, meta-level actions, and also communication.

One approach which does perform meta-reasoning over multiple agents is the work of Raja and Lesser. Cheng, Raja, and Lesser use an Dec-MDP formulation to monitor multiple agents [102, 35]. An MDP is created for each Meteorological Command and Control (MCC) agent of weather radars, to control the heartbeat (time allotment) of radars as well as to handoff radars between agents. A joint utility function is created over all agent tasks. Each agent must reason about how to best control its set of radars, while accounting for the plans of the other agents. A reinforcement learning technique is used to solve the meta-level problem.

Although it regards base-level computation, a multi-agent approach that is similar to ours is the decision-theoretic work of Xuan and Lesser. In one work the authors form contingency plans in a multi-agent environment in which agents make commitments to other agents to perform enabling actions [155]. Since progress of the contingency plan may be monitored, the authors note the relation to meta-reasoning.

A second work by Xuan and Lesser uses decision-theory in order to decompose multi-agent plans at the base-level [154]. Centralized policies are constructed, and the corresponding actions are used as a starting point for decentralized policies without communication. As each agent executes, it accumulates a history (consisting of actions taken and observations received) and it reasons about the set of possible joint histories that are possible, given its own observed history and lack of knowledge of the other agents' histories. When an agent does not know which action is best to take (because the action depends on the history of other agents, which it does not know), the agent considers communication. In order to avoid the complexity involved in tracking the large number of possible histories, the authors outline three strategies for reducing the number of possible histories. These strategies include terminating policies after a certain point in time, merging policy paths together, and localizing policies (that is, only planning around the most probable histories). The work in this thesis differs from the work of Xuan in several ways, including (1) Rather than beginning with the centralized policy, this thesis leverages recent point-based decentralized policies based on the work of Seuken and Zilberstein [120]. (2) This thesis explicitly reasons about bounding the joint utility loss caused by history compression. (3) This thesis explicitly reasons about cost of communication, specifically this thesis reasons about how to construct communication policies whereby an agent defers communication either to another agent or to a future time step (thus, possibly saving itself from overcommunication).

From a high level, the approach used in this thesis is to construct non-communicative policies and then add communication, whereas the work by Xuan and Lesser begins with fully-communicative policies to construct decentralized policies. The work in this thesis is complementary to the work of Xuan and Lesser, it may be possible to use the meta-reasoning contributions of this thesis to augment Xuan and Lesser's works in two ways. First, the observation compression methodology developed in this thesis may be used to augment Xuan and Lesser's merging procedures. Second, the work on Dec-POMDPs with communication in this thesis results in the construction of efficient data structures and algorithms for tracking agent histories, which could be used to augment certain belief updates used in Xuan and Lesser's work.

1.4 This thesis

In this thesis, I utilize formal decision-theoretic models to reason about performance profiles in both partially observable and decentralized settings, and furthermore I utilize meta-reasoning methods to limit resource consumption in both time and space. To do this, I extend the value of computation approach to multi-agent settings in a non-myopic manner. The contributions of this thesis are as follows:

- **Formalization of decentralized monitoring**: I formalize the notion of meta-reasoning in multi-agent settings. I create a problem representation whose inputs are the performance profiles of multiple algorithms as well as the utility of achieving joint qualities at a given point in time. The output is a meta-reasoning policy, so that agents decide when to continue computation, monitor computation, or stop computation.

- **Complexity analysis and solutions to decentralized monitoring**: I analyze the complexity of decentralized monitoring problems where monitoring occurs both locally and globally. Under certain assumptions, decentralized monitoring is NP-complete.

- **Link between conditional performance profiles and Markov Decision Processes**: I show how variants of Markov Decision Processes (MDP, POMDP, Dec-MDP, Dec-POMDP) can be used to represent meta-reasoning problems with performance profiles or conditional performance profiles. Although previous literature has suggested that meta-reasoning may hold the Markov property [55], that Dec-MDPs may be useful for general meta-reasoning [117, 101], and that dynamic programming can be used to solve meta-reasoning problems [55], the formal tie between performance profiles and MDPs has not been fully specified in terms of the MDP state space, the action space, etc. The reduction in this thesis allows state-of-the-art MDP, POMDP, and Dec-MDP solvers to be invoked. Without them, the current algorithms for reasoning about conditional performance profiles are suboptimal when partial observability is added to the MDP framework.

- **Nonmyopic algorithms to perform local and global monitoring**: I show that monitoring problems can be formally encoded and solved using modern decision-theoretic approaches [133, 94], by leveraging recent POMDP algorithms for single-agent monitoring and bilinear programs for two-agent monitoring problems (a formulation is described for extending monitoring problems beyond two agents). For the single-agent case, the new approach outperforms state-of-the-art single-agent monitoring, and for the two agent case, this presents the first decentralized algorithm for reasoning about performance profiles.

- **Observation compression**: I present a meta-reasoning compression method for limiting the space used by decision makers in decentralized settings. Under partial observable circumstances, this limits agent space requirements from exponential with respect to the number of possible observations it can make, to polynomial. The compression-method is "any-space", that is, it can be configured to limit space used to an amount required by the user. The method can also be made lossless, or alternatively it can bound the loss due to the compression to a user-desired amount.

- **Decentralized communication under partial observability**: I present a method whereby decentralized agents can determine when to communicate with other agents in decentralized, partially observable environments with no assumptions made about independence or dependence of agent observations, rewards, and transitions. The resulting algorithm operates by compressing possible non-local histories together based on the expected loss in doing so.

The common thread among the solutions methods is that they each use an expected value of computation approach. Computation is evaluated in terms of the expected value it is expected to produce.

The outline of the thesis is as follows:

In **Chapter 2** I discuss related approaches to meta-reasoning in the literature in further detail. Included in the related approaches are single-agent approaches in which dynamic programming is used to reason about performance profiles, and multi-agent approaches in which reinforcement learning is used. This chapter contextualizes the approach used later in this thesis, in which I perform meta-reasoning in multiple agents using recent decentralized MDP and POMDP-based models.

In **Chapter 3** I formalize a distributed meta-reasoning problem under conditions of both local and global monitoring. The base-level algorithms considered are the class of standard algorithms whose quality increases over time, such as maximum flow or traveling salesperson. I analyze the complexity of both local monitoring (where an agent may monitor the state of its own computation) as well as global monitoring (where an agent may monitor the state of computation of other agents) of such algorithms, and I show that under certain conditions where agent computations are independent of each other, the problem of finding optimal monitoring policies in NP-complete. Then I produce monitoring policies for several variants of the problem, and I empirically evaluate the policies.

In **Chapter 4** I focus on a Decentralized POMDP (Dec-POMDP) planner as the base-level algorithm. Dec-POMDP planners produce coordinated multi-agent policies in partially observable environments without any transition or observation independence. I develop a method of compressing observations together in order to perform more efficient reasoning with respect to space. This reduces the space requirements of Dec-POMDP solvers from exponential (with respect to the number of observations) to polynomial. The compression method computes a loss-bound online, thus compression can be made either lossless, or it can use a user-specified loss bound. Meta-reasoning is used to decide on the trade-off of loss bound for plan quality.

In **Chapter 5** I consider Dec-MDP and Dec-POMDP policies as the base-level algorithms. Each base-level algorithm executes one agent's portion of a plan. Meta-reasoning is used to determine when agents should communicate with each other to replan. In order to determine when to communicate, agents must reason about the possible states and observations of other agents. However, the number of possible histories grows doubly-exponentially with respect to the number possible observations of the other agents. I show how, through efficient data structures, these possible histories can be compressed together, so as to produce non-myopic joint communication policies among the agents.

In **Chapter 6** I briefly consider an application meta-reasoning when one of the base-level decision-makers is autonomous and the other is a human. The application considered is alerting systems for next-generation aircraft. Instead of a stopping and monitoring decision, meta-reasoning with a human in the loop involves selecting a *stage of automation*. Humans are assumed to have different performance profiles under different stages of automation. In order to monitor and predict these performance profiles, a human pilot model is outlined.

	Base-level Algorithm	Meta-level Planner
Chapter 3	Anytime Algorithm	Dec-MDP planner
Chapter 4	Dec-POMDP planner	Observation compression
Chapter 5 part 1	Dec-MDP policy	Communication algorithm
Chapter 5 part 2	Dec-POMDP policy	Communication algorithm
Chapter 6	Human Pilot and flightdeck	TMDP planner

Table 1.1. Base-level algorithms and Meta-level planners used in this thesis.

Table 1.4 summarizes the base-level algorithms and meta-level planners used in this thesis.

1.5 Summary

Distributed meta-reasoning is defined in this thesis as the monitoring and control of distributed systems. This thesis contains three contributions to decentralized meta-reasoning. First, it builds upon work for the single-agent case that uses performance profiles in order to develop monitoring and control policies at the meta-level. Second, it develops an online method (during computation time) of compressing together observations in the Dec-POMDP model, in order to mitigate the complexity of solving these models. Third, in a Dec-POMDP model with communication (Dec-POMDP-Comm-Sync), it develops a method of compressing non-local histories of other agents together, in order to simplify the decision for each agent of when to communicate.

The next chapter will discuss current state-of-the-art approaches for the challenges and models associated with these three contributions. It will review complimentary state-of-the-art approaches, and it will also formalize the approaches that will be extended for this thesis.

CHAPTER 2

RELATED WORK

This chapter provides an overview of the work in meta-reasoning in computer science and other literatures, as well as the relevant models used for this thesis. Surveys of the state of the art in meta-reasoning can be found in [7, 39, 117]. This chapter contains three sections, first on meta-reasoning, next on the Dec-POMDP model, and third on models of communication between agents. On meta-reasoning, 2.1.1, 2.1.2, and 2.1.3 review in detail the history of approaches to meta-reasoning in the single-agent. Section 2.1.4 introduces the work of Zilberstein for single-agent meta-reasoning, which will be expanded to the decentralized case for this thesis. We briefly note other existing state-of-the-art approaches to decentralized meta-reasoning in section 2.1.5 and section 2.1.6. Section 2.2 will then review POMDP-based models. This will include the definition of a Dec-POMDP in 2.2.1 and then solution algorithms from the literature, including top-down, bottom-up, and point-based algorithms. Section 2.3 will provide an overview of work that has taken place regarding the Dec-POMDP-Comm and similar models of communication.

The idea that the cost of decision making must be factored into the decision making process was introduced by Herbert Simon in the 1950's [123]. Concepts originating with Simon include bounded rationality, procedural versus substantive rationality, and satisficing. Bounded rationality refers to limitations in abilities of decision makers, such as cognitive limitations, memory limitations, and time limitations. Procedural rationality refers to systems that compute the rational thing to do, versus substantive rationality which refers to systems that “simply do the rational thing” [110]. Computing the rational thing to do may require a large amount of overhead, which can become prohibitive. This leads to the concept of satisficing (as opposed to optimizing), or producing solutions that are good enough [123].

Russell and Subramanian describe four possible definitions of rational agent behavior [110]:

- *Perfect rationality*: “A perfectly rational agent always acts to maximize expected utility given the information it has acquired from the environment.” This notion corresponds to perfectly rational agents in economics and philosophy.
- *Calculative rationality*: A calculatively rational agent returns the rational choice, given the information it had acquired from the environment before deliberation began.
- *Meta-level rationality*: A meta-level rational agent optimizes over the “object-level computations” which select the actions.
- *Bounded optimality*: A bounded optimal agent behaves “as well as possible given its computational resources”.

Perfectly rational agents, while desirable, may be impractical to achieve given computational constraints. For example, the time required to compute an optimal agent action may exceed the time window in which the action must be taken. Calculatively rational agents reflect typical agents, in fact Russell and Subramanian describe these agents as “the notion of rationality studied in AI”. In practice, such agents may be of limited value, as a calculatively rational chess, poker, or Go program would not complete within a lifetime. Meta-level rationality is useful, but the meta-level computations themselves may take time. Thus, Russell and Subramanian suggest the study of bounded optimality. Bounded optimality specifies “optimal programs rather than optimal actions or optimal computation sequences”, and presents the most comprehensive viewpoint.

This thesis focuses on the latter two levels of rationality. Section 2.1 will describe the larger frameworks for agent meta-level reasoning, which include how the reasoning interacts with the selection of actions at the base level. After that, the details of a particular type of meta-level reasoning, *time dependent planning*, will be described. Subsection 2.1.4 will review single-agent techniques to add monitoring and control to time-dependent plans, this subsection in particular will introduce a notation used later in the thesis. After this, different decentralized meta-reasoning techniques in artificial intelligence and other literatures will be surveyed and the section will conclude.

2.1 Frameworks of meta-level reasoning

Two frameworks which describe the interaction of meta-level reasoning with base-level computation are the meta-reasoning framework of Russell and Wefald [111], and the flexible computation framework of Horvitz [63, 64, 65, 62]. These frameworks were developed in the late 1980’s and early 1990’s. Russell and Wefald’s work studied search algorithms in particular, while Horvitz’s early work was applied to belief networks, theorem proving, and graphics rendering. These works defined many of the underlying concepts which were seen in later works and will be used throughout this thesis, including *time-separable utility* [63, 111], *expected value of computation* [63, 62], and *bounded optimality* itself [111].

2.1.1 Discrete deliberation scheduling

Russell and Wefald referred to their analysis as “Principles of Metareasoning” [111], in order to avoid ambiguity this section will use the term ”Discrete deliberation scheduling” as used by Schut and Wooldridge to describe this work [117].

Russell and Wefald describe three models of deliberation, paraphrased below:

- *External model*: This level analyzes a system as an external observer of the system’s internal and external states. There is a default action, and the goal of computation is to refine it.

- *Estimated utility model*: Agents assign explicit numerical estimates to the utilities of action outcomes, which are often referred to as evaluation functions. Agents select the action whose current *estimate* is the maximum. Deliberation refines the utility estimates.

- *Concrete model*: Agent utility estimates are updated by computation. This provides a specification for how the results of a computation revise the agent’s intended actions.

In order to develop these models, the following notation was used in [111] [2].

- E_i: item i of the set of external actions available to the agent in the current state.
- Γ_j: item j of the set of computational actions available to the agent.
- W_k: A world state labeled k. This includes both external and internal state of the agent.

[2]Some of the symbols representing variable names have been altered from [111], in order to avoid notation conflicts with the rest of this thesis.

- $[X]$: the world state that results when action X is taken in the current state, where the X can either be internal or external.
- $[X, W_k]$: the result of taking action X in the world state W_k.
- $U[W_k]$: the utility of world state W_k. Russell and Wefald note that this typically depends on external state only.
- $\vec{\Gamma}$: a sequence of computational actions. The notation $\vec{T}$ is used to refer to a future sequence of computational actions.
- $\vec{\Gamma}.\Gamma_j$: the sequence of computational actions consisting of the sequence $\vec{\Gamma}$ followed by computational action Γ_j.
- α: the agent's current default external action.

The notation specifies both "external actions" and "computational actions". External actions are the domain-level actions which are referred to as Doing in figure 1.1. Computational actions specify reasoning actions. The default action, α, is an external action. The next sections provide a brief overview of how default actions can be modified through the use of meta-reasoning actions.

2.1.1.1 External model

In discrete deliberation scheduling, the value of a computational action Γ_j is the resulting increase in utility over the default action α that would have been taken.

$$V(\Gamma_j) = U([\Gamma_j]) - U([\alpha]) \tag{2.1}$$

Russell and Wefald refer to a "full computation" as a computation which leads to an internal state which yields an action choice, and a "partial computation" as a computation which results in an internal state which does not yet yield an action choice. If Γ_j is a complete computation and α_{Γ_j} is the action taken as the result of it, then $U[\alpha_{\Gamma_j}, [\Gamma_j]])$ is substituted for $U([\Gamma_j])$.

$$U([\Gamma_j]) = U[\alpha_{\Gamma_j}, [\Gamma_j]]) \tag{2.2}$$

If Γ_j is a partial computation, its utility is defined by the expected utility of the base-level actions which may follow it. Let $P(\vec{T})$ define the probability that the agent will perform the computation sequence $\vec{T}$ after Γ_j.

$$U([\Gamma_j]) = \sum_{\vec{T}} P(\vec{T}) U[\alpha_{\vec{T}}, [\Gamma_j.\vec{T}]]) \tag{2.3}$$

The above says that the value of a computation is the expectation of the base level action over the possible finishing computations that select it. Using the two expressions above, an ideal control algorithm would perform two steps in a loop. First, it would select computations with the highest expected value, until no computation had a positive expected value. Then it would perform the action α selected by the last computation.

2.1.1.2 Estimated utility model

The model of the previous subsection can be refined to estimate utility. Let $\vec{\Gamma}$ represent a series of computations. Since true utility is unknowable and can only be estimated, use $\hat{U}$ instead of U to reflect a utility estimate, and use a superscript when helpful to denote that a computation has been performed. Value of computation from equation 2.1 becomes the following

$$\hat{V}(\Gamma_j) = \hat{U}^{\vec{\Gamma}.\Gamma_j}([\Gamma_j]) - \hat{U}^{\vec{\Gamma}.\Gamma_j}([\alpha]) \tag{2.4}$$

In this thesis, we examine a special case of utility functions, specifically utility functions $\hat{U}$ of a computation and action that can be split into an intrinsic utility of the action U_I and a cost function C.

$$\hat{U}([E_i, [\Gamma_j]]) = \hat{U}_I([E_i]) - C(E_i, \Gamma_j) \tag{2.5}$$

More specifically, we examine the case where the intrinsic value is due to the action, and the cost is due to the *time cost* TC of computation. This concept is relevant to flexible computation of Horvitz, as well as throughout this thesis.

$$\hat{U}([E_i, [\Gamma_j]]) = \hat{U}_I([E_i]) - TC(|\Gamma_j|) \tag{2.6}$$

where $|\Gamma_j|$ represents the elapsed time for performing computation Γ_j. In the above, the cost of performing a computation Γ_j and then performing an external action E_i at the end of it is decomposed into the delay associated with Γ_j, and the utility associated with E_i.

The value of a computation can then be decomposed:

$$\begin{aligned}\hat{V}([\Gamma_j]) &= \hat{U}^{\vec{\Gamma}.\Gamma_j}([\alpha_{\Gamma_j}, [\Gamma_j]]) - \hat{U}^{\vec{\Gamma}.\Gamma_j}([\alpha]) \\ &= \hat{U}_I^{\vec{\Gamma}.\Gamma_j}([\alpha_{\Gamma_j}]) - \hat{U}_I^{\vec{\Gamma}.\Gamma_j}([\alpha]) - TC(|\Gamma_j|) \\ &= \Delta(\Gamma_j) - TC(|\Gamma_j|)\end{aligned} \tag{2.7}$$

where

$$\Delta(\Gamma_j) = \hat{U}_I^{\vec{\Gamma}.\Gamma_j}([\alpha_{\Gamma_j}]) - \hat{U}_I^{\vec{\Gamma}.\Gamma_j}([\alpha])$$

The equality in the last step of 2.7 represents the expected increase of U_I (which is the estimate of the value of the revised action minus the value of the default action α), minus the expected decrease due to TC.

2.1.1.3 Summary of discrete deliberation scheduling

In summary, discrete deliberation scheduling provides a formal notation and system of meta-reasoning over how computation affects base-level decisions. The value of computation is formalized as having two components, the increase in expected value of the action, and the negative cost of time. The expected value of the action, in turn, is specified as the expectation of the new action versus the expectation of the default action. For partial computations, the expectation of the new action depends on the resulting choice of action when the computation finishes.

Discrete deliberation scheduling models the base level computation as well as the meta-reasoning process which supervises it. As a result, the framework is exceptionally descriptive, but it often becomes computationally intractable to optimize both types of computations simultaneously. The approach in this thesis will only allow the agent to select from meta-level computation options, achieving bounded optimal solutions for this smaller decision space. Subsection 2.1.3 will show how base-level reasoning can be abstracted into a module that produces an output whose quality depends on its runtime. The role of meta-reasoning in this thesis is to alter the runtime of this module, but not to alter its internals.

2.1.2 Flexible computation

The Protos (PROject on computational resources and Tradeoffs) system was designed to explore the use of meta-reasoning to control inference approximation methods [64]. The resulting work provided many of the foundations of meta-reasoning in the artificial intelligence community, such as expected value of computation (which was also seen in discrete deliberation scheduling), separable utility functions, and decision-theoretic meta-reasoning. These will be described below. Much of the early work on the Protos system regarded medical scenarios, specifically the use of belief networks in order to perform medical diagnosis. The problem of probabilistic inference with belief networks is NP-hard, and becomes intractable as problems become large [38].

2.1.2.1 Protos architecture and bounded optimality

Protos contains both a reasoning and a meta-reasoning system. The reasoning system decides on actions to take at the base level. For example, the authors consider a medical scenario involving a hypothesis (designated H_j) and an action to take (representing a treatment and designated A_i). Hypotheses may correspond to different diseases, and actions correspond to different treatments. There is uncertainty in the hypothesis. The expected utility of a treatment is therefore represented by:

$$eu(A_i, t) = \sum_{j=1}^{n} p(H_j|E, \xi) u(A_i H_j, t) \tag{2.8}$$

where eu represents the expected utility of the treatment, the function $u(A_i H_j, t)$ represents the utility of the treatment on the disease at time t, and the probability function $p(H_j|E, \xi)$ represents the probability of the disease being present, based on a model of the world represented in E and ξ.

The result of this analysis produces utility functions such as in figure 2.1. The endpoints of the solid lines represent the utilities of the treatments on the disease, and the lines represent the utilities over probability distributions. In the figure shown, we assume $\sum p(H_j|E, \xi) = 1$. In this case, treatment A_2 should be undertaken when $P(H_1|E, \xi) < p^*$, and treatment A_1 should be taken otherwise at time t.

However, time t may have an effect on utility. In a similar manner to the work on discrete deliberation scheduling, Horvitz separates value into two components, the *object-related* value described above, and *inference-related value* which depends on time [63]. The latter is described as "the expected disutility intrinsically associated with computation, such as the cost a physician might attribute to the delay of a decision". As

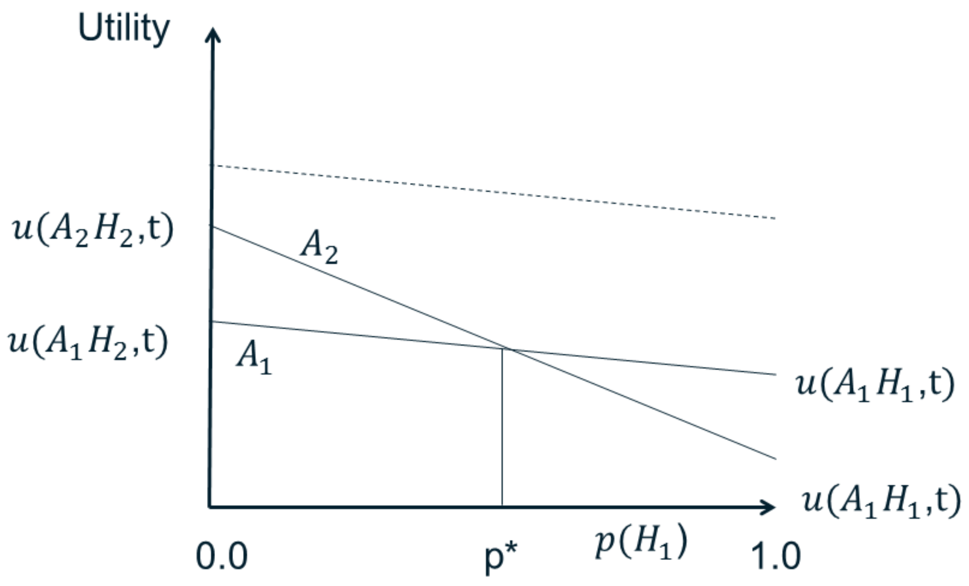


Figure 2.1. Flexible computation vectors

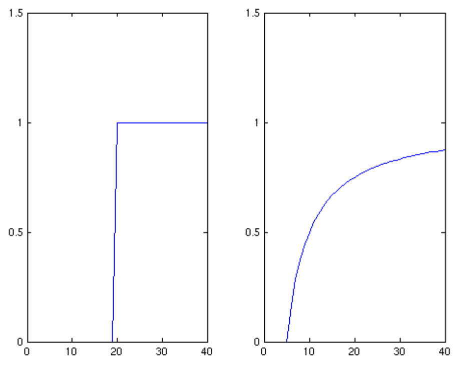


Figure 2.2. Performance profiles. The horizontal axis is computation time and the vertical axis is performance quality. On the left is the profile of an algorithm in the one-shot improvement class. On the right is an algorithm that shows monotonic improvement.

an example, view the dotted line in figure 2.1. This is the $u(A_1H_i, t_0)$, where t_0 is less than t. As shown, the value of taking action A_1 is higher at an earlier time step.

There is a trade-off of time for decision quality. As discussed above, actions have higher utility when taken sooner, however, it may be the case that the probability model $p(H_j|E, \xi)$ will become more accurate with time, leading to better decisions. Horvitz formally analyzes the trade-off by computing the *Expected Value of Computation (EVC)*, which is the expected utility of the decision at the current time step subtracted from the expected utility at a future time step. The optimization problem of computational utility given the assumptions about the problems and the resource constraints is referred to as *bounded optimality* [63].

2.1.3 Time dependent planning

Work on time dependent planning began contemporaneously to the development of flexible computation. However, instead of deriving from the problem of the refinement of hypothesis fidelity, early work on time dependent planning focused on how to best schedule computationally intense responses to varying events on a single processor.

Dean and Boddy used the term “anytime algorithm” in the 1980’s to describe a class of algorithms that have three properties [40]:

- They can be preempted with negligible overhead.
- Any algorithm that is preempted at any time will nevertheless return some answer.
- The quality of the returned answers improves as a function of time.

Figure 2.2 portrays performance profiles for different classes of algorithms. For the one-shot improvement class (which had been the subject of study before the work of Dean and Boddy), shown on the left, the algorithm will run for a period of time, and after some amount of time its performance will jump to a high level of quality. By contrast, for the anytime algorithm shown on the right, performance will improve in a piecewise monotonically increasing linear manner, and if the algorithm is interrupted, an intermediate quality will be reported.

Time dependent planning presents a method for scheduling several such anytime algorithms on a single processor. Time dependent planning problems are formalized as follows [19]

- A set of time points, T.
- A set of event (or condition) types, C. Each event type is associated with a time point in T.
- A set of action (or response) types, A.
- A set of decision procedures, D.
- A value function, $V(a \in A | c \in C)$.

The agent knows about a set of pending events in C and the time for each event. Dean and Boddy define a response function for each event $c \in C$ which is simply denoted Response(c), and maps to an action $a \in A$. They also define a value function V which maps the response to a real value. The total value of the agent's responses is:

$$\sum_{c \in C} V(\text{Response}(c)|c) \tag{2.9}$$

For each event type c there is one decision procedure $dp(c) \in D$ along with a function $\mu_c : \mathbb{R} \rightarrow \mathbb{R}$ which inputs the amount of time δ to run the decision procedure, and returns the expected value of the response to c generated by $dp(c)$ for δ time units. The decision procedure is run prior to the event for which it computes a response. Let alloc$(\delta, dp(c))$ be the allocation of δ time units to $dp(c)$. We can then specify

$$V(\text{Response}(c)|c, \text{alloc}(\delta, dp(c)) \tag{2.10}$$

as the value of the response given the event c and the allocation of δ time units to $dp(c)$ to calculate Response(c). We can compute an expectation of the above value function:

$$\mu_c(\delta) = E(V(\text{Response}(c)|c, \text{alloc}(\delta, dp(c)))) \tag{2.11}$$

This framework defines a deliberation scheduling problem, the problem of which decision procedure to invoke at any given time. For the case where slopes of consecutive line segments of the quality function are decreasing (i.e. "diminishing returns"), the scheduling problem can be solved using an algorithm such as in [19], which is summarized in Algorithm 1.

The algorithm works backwards from the time of the last event. The algorithm is invoked with $\hat{t}$ as the current time and the variable t is initialized to be the time of the last event.

Algorithm 1: Deliberation scheduling procedure for time dependent planning

```
begin
    for $i = 1$ to $n$ do
        $\delta_i \leftarrow 0$;
    $\hat{t} \leftarrow$ current time;
    $t \leftarrow$ the (future) time of the last pending event in $C$;
    while $t \neq \hat{t}$ do
        $\Delta \leftarrow \min\{t - \hat{t}, t - \text{last}(t), minalloc(\delta_i)\}$;
        $i \leftarrow \text{argmax}\{\gamma_i(\delta_i) | c_i \in \Lambda(t)\}$;
        $\delta_i \leftarrow \delta_i + \Delta$ ;
        $t \leftarrow t - \Delta$;
    $t \leftarrow \hat{t}$;
    for $i = 1$ to $n$ do
        run the ith decision procedure from $t$ until $t + \delta_i$.;
        $t \leftarrow t + \delta_i$;
end
```

Let $C = \{c_1, ...c_n\}$ be the set events, and let μ_i be the function describing the performance profile for the ith decision procedure.

The algorithm starts from the time of the last event and works backwards. Let $\Lambda(t)$ be the set of events that occur after time t, that is

$$\Lambda(t) = \{c | (c \in C) \ \ \& \ \ (time(c) \geq t)\} \tag{2.12}$$

Create a set of variables δ_i reflecting the amount of allocation to decision procedure i, and initialize each of these allocations to zero. Also let $last(t)$ be the first time before t that an event in C occurs that is not in $\Lambda(t)$. Thus, the algorithm initializes t to the last event. The algorithm loops backwards until $t = \hat{t}$, that is, the time being scheduled equals the current time. In each iteration of the loop, it first computes Δ, the time slice to be scheduled in this iteration. Δ represents the amount of time that is currently being scheduled (thus, in each iteration, the time slice being scheduled is the interval between $t - \Delta$ and t). Δ is the minimum of three quantities: (1) The difference between the current time and t (if Δ were to exceed this, the algorithm would be scheduling in the past). (2) The difference between t and the last event that occurs before t. This assures that no event occurs in the time slice chosen. (3) The minimum length of the "next" piece of the piecewise linear function, over each of the decision procedures, based on their allocations thus far. This assures that no decision procedure changes slope in the time slice chosen.

Having selected a time slice Δ, the best decision procedure is chosen for the time slice between $t - \Delta$ and t. Let $\gamma_i(x)$ represent the slope of the performance profile of decision procedure i at x. Thus, $\gamma_i(\delta_i)$ represents the slope of decision procedure i given its current allocation. The decision procedure chosen is the one with the greatest slope. Next, δ_i and t are updated to reflect the choice over the time slice, and the loop iterates.

After the main loop completes, the last loop of the algorithm reschedules the decision allotments into contiguous segments.

2.1.4 Conditional performance profiles

Zilberstein and Russell formalize the use of conditional performance profiles. The formalization requires successive definitions, beginning with the performance profiles from time-dependent planning. The following definitions are quoted from [156].

Definition 2.1. *The performance distribution profile of an algorithm A is a function $D_A : \mathbb{R}^+ \rightarrow Pr(\mathbb{R})$ that maps computation time to a probability distribution over the quality of the results.*

Definition 2.2. *The Conditional Performance Profile (CPP) of an algorithm A is a function $C_A : \mathbb{R} \times \mathbb{R}^+ \rightarrow Pr(\mathbb{R})$ that maps quality and computation time to a probability distribution over the quality of the results.*

Throughout the thesis, we will express the CPP as an expression $Pr(\hat{q}|q,t)$, which is the probability of getting a solution of quality $\hat{q}$ by running an algorithm for t time units.

2.1.4.1 Monitoring performance

Hansen and Zilberstein consider the same time-dependent utility functions of Horvitz [62] and Russell and Wefald [111].

Definition 2.3. *A time-dependent utility function U(q,t) represents the utility of a solution of quality q at time t.*

This function can then be used to find an optimal allocation for time. Three approaches are presented. The simplest approach is to find a fixed running time at the start of the problem. In the remainder of this section, let Q be a set of possible quality levels, let $q \in Q$, and let $q^t \in Q$ be a variable representing a quality level at a time t.

Definition 2.4. *Given a time-dependent utility function and a performance profile, an optimal fixed allocation, t^*, is defined by:*

$$t^* = argmax_t \sum_{q \in Q} Pr(q|t)U(q,t)$$

where q represents a quality level. The second approach is termed the *myopic estimate of the expected value of computation (MEVC)*. In this approach, the agent keeps computing until the value of computation is negative.

Definition 2.5. *The myopic estimate of the expected value of computation (MEVC) at time t is:*

$$MEVC(\Delta t) = \sum_{\hat{q} \in Q} Pr(\hat{q}|q,\Delta t)U(\hat{q},t+\Delta t) - U(q,t)$$

where q and $\hat{q}$ are quality levels. This myopic approach is optimal when two conditions both hold. First, the MEVC computation itself takes a negligible amount of time. Second, every time t and quality q with non-positive $MEVC$ implies that $MEVC$ will also be non-positive for every time $t' > t$.

The third approach is to consider the sequential nature of the stopping problem, and construct a monitoring policy.

Definition 2.6. *A monitoring policy $\pi(q,t)$ is a mapping from time step* t *and quality level* q *to a decision whether to continue the algorithm or stop and act on the currently available solution.*

Monitoring policies in discrete time steps can be constructed through dynamic programming, using the following value term with Δt representing a single time step:

$$V(q,t) = \max_{d} \begin{cases} \text{if } d = \text{stop:} \\ U(q,t), \\ \text{if } d = \text{continue:} \\ \sum_{q^{\ t+\Delta t}} Pr(q^{\ t+\Delta t} | q^{t}, \Delta t) V(q^{t+\Delta t}, t+\Delta t) \end{cases} \tag{2.13}$$

to determine the following policy:

$$\pi(q,t) = \text{argmax}_d \begin{cases} \text{if } d = \text{stop:} \\ U(q,t), \\ \text{if } d = \text{continue:} \\ \sum_{q^{\ t+\Delta t}} Pr(q^{\ t+\Delta t} | q^{t}, \Delta t) V(q^{t+\Delta t}, t+\Delta t) \end{cases}$$

However, monitoring may involve overhead, for example gauging the quality of a solution may require context switching on the part of the processor, queries to disk drives, or computation. It can be modeled as having a monitoring cost C.

Definition 2.7. *A cost-sensitive monitoring policy $\pi_c(q,t)$ is a mapping from time step t and quality level q into a monitoring decision ($\Delta t, m$) such that Δt represents the additional amount of time to allocate to the anytime algorithm, and m is a binary variable that represents whether to monitor at the end of the time allocation or step without monitoring.*

The value function can be modified to account for monitoring.

$$V_c(q,t) = \max_{\Delta t, m} \begin{cases} \text{if } m = \text{stop:} \\ \sum_{q^{t+\Delta t}} Pr(q^{t+\Delta t}|q^t, \Delta t) U(q^{t+\Delta t}, t+\Delta t), \\ \text{if } m = \text{monitor:} \\ \sum_{q^{t+\Delta t}} Pr(q^{t+\Delta t}|q^t, \Delta t) V_c(q^{t+\Delta t}, t+\Delta t) - C \end{cases} \tag{2.14}$$

The "continue" action is implicitly represented within this function by selecting $\Delta t > 0$.

2.1.4.2 The compilation Problem

A related class of problems which utilize performance profiles has been coined by Zilberstein and Russell as *the compilation problem* [156]. This class of problems is characterized by the presence of multiple anytime algorithms, in which the output of some of the algorithms may serve as the input to other algorithms. The authors provide the example of a diagnosis and treatment system, where the output of the diagnosis anytime algorithms serves as the input to the treatment anytime algorithm.

Formally, the problem is defined in terms of anytime functions. Let $\mathcal{F}$ be a set of anytime functions and $\mathcal{I}$ be a set of input variables. Zilberstein and Russell define functional expressions as follows [156]:

Definition 2.8. *A functional expression over $\mathcal{F}$ with input $\mathcal{I}$ is:*

- *An input variable $i_j \in \mathcal{I}$, or*
- *An expression $f(g_1, \ldots, g_n)$ where $f \in \mathcal{F}$ and each g_j is a functional expression.*

Each functional expression represents a module with a conditional performance profile operating on a set of inputs, which may be input variables or else recursively may be the output of other functional expressions. The recursive relationship between functional expressions can be represented by either a tree or a directed acyclic graph (DAG). Solving the compilation problem involves finding an optimal schedule, which is a conditional allocation of time to each of the component algorithms. The problem is NP-complete, but the authors use the mechanism of *local compilation*, that is, solving one programming structure at a time, to produce solutions that are optimal given certain graph structures and monotonicity assumptions.

2.1.5 Decentralized meta-reasoning

The approaches explored so far have involved meta-reasoning over a single agent. Other work considers meta-reasoning over multiple agents. Design-to-time scheduling is an approach to solving problems in domains in which there are soft and hard real-time deadlines and in which there are multiple solution methods available [47]. It has been used to schedule discrete methods in a TAEMS (Task Analysis, Environment Modeling, and Simulation) environment [41] "with the goal of maximizing the value of the scheduled computation" [48]. Raja and Lesser also reason about TAEMS, and suggest a Dec-MDP representation of multiagent meta-reasoning [103] to support the scheduling of tasks in the network, making the following assumptions:

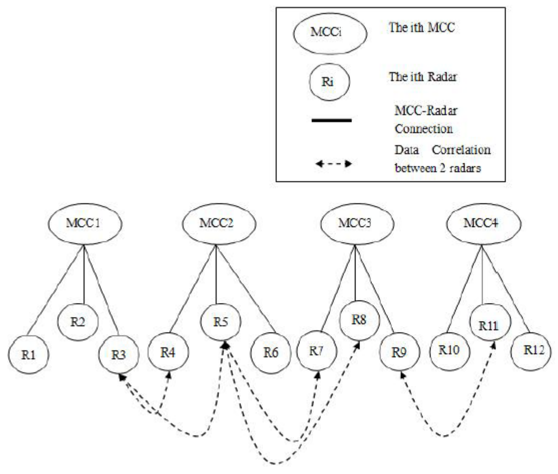


Fig. 1. An Example NetRads Topology

Figure 2.3. NetRads topology. Figure extracted from [35]

- Agents are cooperative.
- Agents have goals which have deadlines.
- High-level goals may be partly achieved.
- There is a finite time horizon

The authors use meta-level control to support "decisions on when to accept, delay, or reject a new task; when it is appropriate to negotiate with another agent; whether to renegotiate when a negotiation task fails; how much effort to put into scheduling when reasoning about a new task; and whether to reschedule when actual performance deviates from expected performance". These higher level task scheduling and negotiation concerns complement the finer-grained work on conditional performance profiles seen in previous sections of this chapter as well as in the rest of this thesis.

The authors define a set of features, which together define the system *state*. Meta-reasoning decisions take the form of *actions*, which consist of information gathering actions, planning and scheduling actions, and coordination actions.

The text below represents the formal model in [103], for each agent:

- S is the set of states of the agent. The state space consists of the state of a set of abstract features, which include task status, an environmental model specifying probability of task arrival, schedule slack and other influences on action choice, and performance characteristics.
- A is the set of possible control actions available to the agent.
- $P(s_j|s_i, a)$ is the transition table, which represents probability that the agent transitions to state s_j as a result of taking action a in state s_i.
- A policy π is a description of the behavior of the agent. It is a mapping $\pi : S \to A$ which specifies a control action to be taken for each state.
- $R(s_i, a, s_j)$ is the reward obtained in state s_j as a consequence of taking control action a in state s_i and then executing the domain actions that follow a.
- $U_\pi(s_i)$ is the utility of state s_i under policy π.

The Dec-MDP problem is solved through reinforcement learning. Each agent reasons as an MDP and reasons about the Q-values of state-actions.

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max Q(s', a') \tag{2.15}$$

The framework is used for analysis of how to best coordinate a TAEMS network. The authors note three features of their approach. First, state features are identified to approximate the system state. This prevents the problem from becoming intractable. Second, heuristics are evaluated as a form of meta-level control. Third, the meta-level Markov Decision Process is solved using reinforcement learning. Although none of these features are in common with this thesis, the view of meta-reasoning as Dec-MDP provides the foundations of the decision theoretic approach used for this thesis.

One successive work by Cheng, Raja, and Lesser used a Dec-MDP representation and reinforcement learning to control a group of radars [35]. Figure 2.3 shows the radars as well as their controller agents. The goal is to jointly scan for weather phenomena by accomplishing tasks. Tasks have types which include *storm, rotation, reflectivity, or velocity*, and are also classified as either *pinpointing or non-pinpointing*. Each task has a degree of correlation attached to it, reflecting the interdependence of tasks. Radars can be neighbors if they share overlapping scanning regions. Figure 2.3 shows the network of radars. Radars connected by a dotted line hold overlapping scanning regions.

The authors created a Dec-MDP where the state space contained features, including information about self, information about neighbors, and degree of data correlation. Nodes could take actions to move tasks among the radars or control the heartbeat. The resulting Dec-MDP was solved using Abdallah and Lesser's WPL algorithm [1]. The algorithm moves each policy along the gradient of its Q-value, and it learns faster when the Q values are higher than the total average reward, and slower if Q values are below.

2.1.6 Decentralized Wald framework

Similar work in decentralized systems has taken place in the communications field. Although this related work is not the subject of study in this thesis, we briefly use this section to note the relation. The solution techniques that will be developed in this thesis, especially in Chapter 3, may be applicable to the communications field, and vice versa.

The Decentralized Wald problem is formalized as follows [139]. Informally, this problem can be described as involving two sensors, and an object which has two possible classifications. At each time step, both sensors make an observation about the object, but each sensor is not aware of the observation of the other. At each step, each sensor may choose to continue for another time step (at a cost), or to stop observing and classify the object. The formal representation follows:

There are two detectors, numbered 1 and 2. The ith detector's observation y_i at time t is described by

$$y_i(t) = f_i(h, w_t^i) \tag{2.16}$$

where w_t^i for $i = 1, 2$ are mutually independent i.i.d. sequences and h is a hypothesis. Define u_i as

$$u_i(t) = \gamma_i(y_i^t) \tag{2.17}$$

where γ_i represents the decision made by detector i and

$$y_i^t = (y_i(1)...y_i(t))$$

and

$$u_i = 0, 1$$

Also define a joint cost function $J(u_1, u_2, h)$, with h being the true hypothesis, with the following rules

$$\begin{aligned} J(0, u_2, h_1) &\geq J(1, u_2, h_1) \\ J(1, u_2, h_0) &\geq J(1, u_2, h_1) \\ J(1, u_2, h_0) &\geq J(0, u_2, h_0) \\ J(0, u_2, h_1) &\geq J(0, u_2, h_0) \end{aligned} \tag{2.18}$$

and similar rules for u_1. Let τ_i denote the stopping time of the ith agent. The decentralized Wald problem is

$$\text{Minimize } E(c\tau_1(\gamma_1) + c\tau_2(\gamma_2) + J(\gamma_1(y_1^{\gamma_1}), \gamma_2(y_2^{\gamma_2}), h) \tag{2.19}$$

The problem is solved approximately, using results from statistical sequential analysis [147].

2.2 POMDP-based frameworks

In this thesis we study the use of the Partially Observable Markov Decision Process (POMDP) and its extensions, for meta-level reasoning. The POMDP is an extension of the single-agent MDP, which is a well-known framework in agent literature [14]. In an MDP model, an agent is modeled as an entity which exists in a state. The Markov assumption is made, that state captures all necessary information about the agent's history. One special state is designated the *start state*. Over a series of discrete time steps, the agent is permitted to select an item from a set of actions at each step. Taking an action from a state probabilistically results in a change of state. In addition, a reward is defined for each state-action pair. In a finite-horizon MDP, a small number of steps is specified, whereas in an infinite-horizon MDP, a discount factor $\gamma \in (0, 1)$ is specified, and reward is multiplied by γ^t, where t is the time-step of the rewarded state and action. An infinite-horizon MDP *policy* is defined as a mapping from agent state to an action. A finite-horizon policy maps time and state to an action. For the rest of this proposal, we will restrict our attention to finite-horizon cases.

The POMDP extends the MDP to account for uncertainty of state [10, 126]. In a POMDP, the agent receives an observation at each time step, each possible observation has a given probability. Thus, state is partially observable. A finite-horizon POMDP policy can be viewed as a mapping from the time step and a belief distribution over states, to an action. An alternative representation is as a mapping from a starting distribution and a subsequent agent history to an action. The two representations are related, as knowledge of an agent's history defines the time step and a belief distribution over states.

Work in the late 1990's and early 2000's focused on extending the POMDP to multiple agents. Early work was conducted by Xuan and Lesser, who focused on the construction of decentralized plans [154], and by Bernstein and Zilberstein, who focused on complexity analysis of several model variations [17]. The **Dec-POMDP** or Decentralized Markov Decision Process extends the POMDP to multiple agents [17]. Instead of single actions at each step, joint actions are taken instead, one for each agent. Instead of single observations, joint observations are received. However, each agent's actions and observations are not necessarily known to the other agents. We will elaborate on the model and on solution techniques to Decentralized POMDPs in the next subsection.

Bernstein and Zilberstein proved that the Dec-POMDP model is NEXP-hard [17]. This result holds even for the Dec-MDP, a related model in which joint observations uniquely identify the state. The complexity can be reduced to NP-complete if agents are transition independent, observation independent, and can fully observe their local state [51].

There are three other models of note that are similar to the Dec-POMDP, but which are not described elsewhere in this thesis:

- **POSG**: The POSG model is the same as the Dec-POMDP model, except each agent may have its own reward function.
- **I-POMDP** [49]: This model is similar to the POSG model in that different agents may have different reward functions. Also, for each agent, the beliefs over other agent types are included in the state space. Solutions are the optimal policies which deduce an action, given a belief state about the state and the belief state of

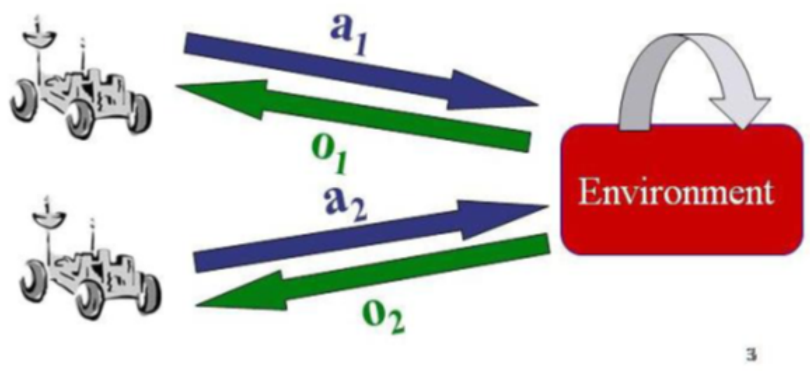


Figure 2.4. Illustration of the Dec-POMDP model. At each step, each agent takes an action, the environment transitions, and each agent receives its own observation.

the other agents. An advantage of the I-POMDP over Dec-POMDPs is that they allow game-theoretic solutions for adversarial games, whereas Dec-POMDPs are limited to cooperative games. A disadvantage is the increased size of the state space.

- **MTDP**, or Multiagent Team Decision Problem [99]: This is a similar model to Dec-POMDP. Elements represented include a set of agents, a set of world states, a set of actions for each agent, a probabilistic distribution over successor states, a joint observation function, a reward function, and a horizon. MTDP is equivalent to Dec-POMDP, assuming all agents can recall their past histories [118].

2.2.1 Dec-POMDP model

We select the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) as a model to study in this thesis. The Dec-POMDP model is broad enough to capture the aspects of multi-agent planning and execution listed earlier. In particular, a Dec-POMDP is a sequential, multi-agent problem where individual agents receive observations which correspond to beliefs about the environment. Agents are not aware of each other's observations unless communication takes place. A Dec-POMDP is defined as follows:

- A set of agents numbered $1..n$
- S: the set of domain states.
- $b_0 \in \Delta S$: the initial belief state distribution.
- $A = \times A_i$: the set of joint actions, where A_i is the set of actions available to agent i. At each time step, agents take one joint action $\mathbf{a} = \langle a_1, .., a_n \rangle$.
- P: the transition function. $\forall s, s' \in S, \mathbf{a} \in \mathbf{A}$, $P(s'|s, \mathbf{a})$ is the probability of transitioning to state s' given the previous state was s and joint action $\mathbf{a}$ was taken by the agents.
- R: the reward function. $\forall s \in S, \mathbf{a} \in A, R(s, \mathbf{a})$ is the immediate reward for taking joint action $\mathbf{a}$ in state s.
- $\Omega = \times \Omega_i$: the sets of joint observations. Each agent i receives only its own observation $o_i \in \Omega_i$ at each time step. The vector of received observations is denoted $\mathbf{o} = \langle o_1, .., o_n \rangle$.
- O, the observation function. It specifies joint observation probability $O(\mathbf{o}|s', \mathbf{a})$, the joint probability that agents see corresponding observation $\mathbf{o} \in \Omega$ after the agents took joint action $\mathbf{a}$ causing a state transition to s'. Occasionally in this thesis, the observation function may be specified in terms of the previous state and the successive state, $O(\mathbf{o}|s, \mathbf{a}, s')$.
- T, the horizon, or number of steps, in the problem.

The model unfolds over T stages. At each stage, all agents simultaneously select an action and receive a reward and observation. The objective is to maximize the expected sum of joint rewards received.

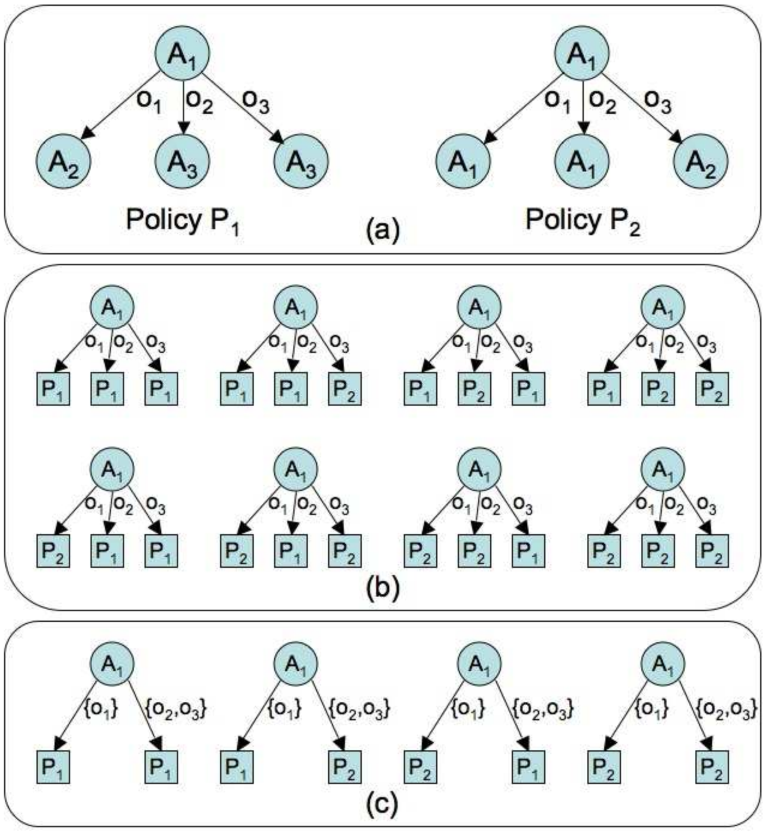


Figure 2.5. (a) Two policy trees. (b) The result of a full-backup of the policy trees shown in part (a) is 24 new trees. Only the trees rooted in A_1 are shown. (c) (Referenced in Chapter 4): The result of running *CompressObs* and then a partial backup. Here $maxObs = 2$ and subpolicies of o_2 and o_3 are forced to be the same.

2.2.2 POMDP concepts

The following two concepts are used in solution approaches.

belief state: A belief state is a probability distribution over the states. The distribution must sum to one. Typically, we use the notation b to represent a belief distribution over states and $b(s)$ to refer to the belief probability of a specific state. b_0 is sometimes used to refer to a belief distribution before the first time step.

policy tree: As stated earlier, an agent policy can be viewed as a mapping from an action and observation history, to an action. Agent policies will be denoted with π. A tree representation of a finite-horizon policy is shown in Figure 2.5 (a). The nodes in the tree represent actions, and the edges represent observations. A policy is executed by starting at the root node, and proceeding downwards according to the received observation.

2.2.3 Single agent case

For a single-agent, the **Value** of a policy tree π in state s, denoted $V(\pi, s)$, is the expected reward that will be achieved by executing the policy. It can be recursively defined as the immediate reward plus the expected reward on future steps. The expected value on future steps is decomposed into the probability of receiving each observation from each possible next state, times the value of the policy that will be taken given that observation. Formally, it is expressed as:

$$V(\pi, s) = R(s, a_\pi) + \sum_{s'} P(s'|s, a_\pi) \sum_{o} O(o|s, a_\pi, s')V(\pi_o, s') \tag{2.20}$$

where a_π is the action defined at the root of policy π, and π_o is the subpolicy of π taken after observation o.

For a single agent, the function of optimal value over belief state is convex (for an illustration, see Figure 2.6(a)). Modern single-agent solution approaches are based on this fact, which can be proved in two steps. First, the value of each policy can be shown to be a hyperplane over belief state. This follows from the fact that if the value of a policy is known for two belief states, the value of that policy at belief points between these two states is a linear combination. Second, the optimal value at a belief state is value of the maximal policy at that state. Maximization of affine functions is a convex function.

Point-based algorithms [95, 133] for single-agent POMDPs leverage convexity. A point-based algorithm proceeds over a series of T steps, at each step the algorithm starts

```
Given DEC-POMDP, horizon $T$
 t=1
 Generate horizon-1 policies for each agent.
 While $t < T$
     backup each agent to generate policies
     prune each agent's policies
     increment $t$
```

Table 2.1. Basic bottom-up algorithm

with a set of policies. At the first step this set of policies is the set of actions. Given a set of policies at step t and a belief state $b(s)$, point-based algorithms find a set of policies at step $t+1$, each of which contains the step t policies as sub-policies. The distinguishing characteristic of point-based algorithms is that the maximum value policy is identified at select belief points, and all other policies are discarded as candidate sub-policies. For a single-point, the best policy can be found in time $|A||\Omega||S||\Pi|$, where $|\Pi|$ is the set of subpolicies retained from the previous step. The loss due to discarding the remaining policies is bounded due to convexity.

2.2.4 Dec-POMDP bottom-up approach

For the multi-agent case, Hansen *et al.* have developed an optimal dynamic programming algorithm that prunes many useless policies [54].

Table 2.1 illustrates the basic algorithm. First, a set of horizon-1 policies are constructed, to represent the last step of the policy. This is simply the set of actions A_i. From these horizon-1 policies, horizon-2 policies are constructed, with the horizon-1 policies as subtrees. In bottom-up planning, horizon $t+1$ trees are trees whose subtrees are horizon t trees.

A key part of the planning process is to evaluate policies. For the two agent case, a joint policy value for policies π_i and π_j at a given state can be written recursively:

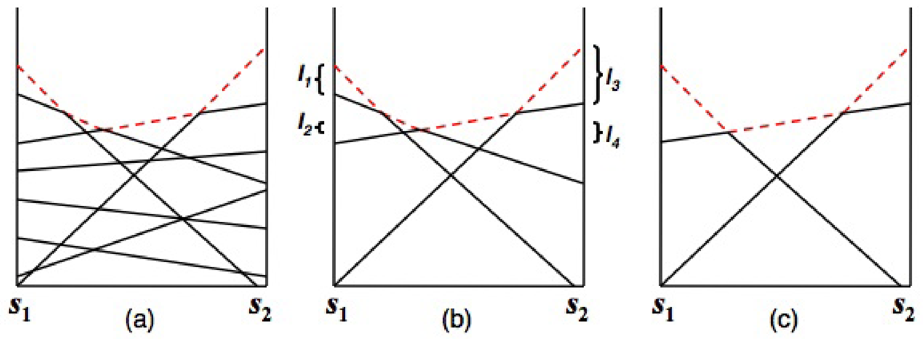


Figure 2.6. Illustration of pruning. (a) The value of a set of vectors at two states s_1 and s_2. (b) The result of pruning with $\epsilon = 0$. Vectors that do not contribute at any belief state are pruned. (c) Pruning for $\epsilon > 0$, vectors that do not contribute more than ϵ are pruned.

$$V(\pi_i, \pi_j, s) = R(s, a_i, a_j) + \sum_{s'} P(s'|s, a_i, a_j) \sum_{o_1,o_2} O(o_1, o_2|s', a_i, a_j) V(\pi_{i,o_1}, \pi_{j,o_2}, s') \tag{2.21}$$

where a_i and a_j represent the root actions taken in the policy trees of each agent and $V(\pi_{i,o_1}, \pi_{j,o_2}, s')$ is the value of continuing in the policy trees based on the observation that was seen and the resulting state.

During the planning process, each agent must choose policies that have high value based on *any* policy the other agent may use. To formalize this, define belief state, $b(s, \pi_j)$, for an agent as the probability that the system state is s and the other agent will use policy π_j. The value of an agent's belief is then:

$$V(b) = \max_{\pi_i} \sum_{s,\pi_j} V(\pi_i, \pi_j, s) b(s, \pi_j) \tag{2.22}$$

which represents the maximum the agent can achieve given its set of policies weighted by the probability that the true state is s and the other agent uses policy π_j. Because

Given a vector to be pruned, α and an undominated set U
For variables: $b(s, \pi_j)$ and δ
Maximize δ
Such that:

$$\forall i \sum_{s,\pi_j} b(s, \pi_j) \left[\alpha(s, \pi_j) - U_i(s, \pi_j)\right] \geq \delta$$

And probability constraint:

$$\sum_{s,\pi_j} b(s, \pi_j) = 1$$

Table 2.2. The two agent linear program for pruning a single vector, α, when compared to the current undominated set, U. $b(s, \pi_j)$ represents the probability of state s and policy π_j for the other agent. $\alpha(s, \pi_j)$ and $U_i(s, \pi_j)$ represent values of these vectors given state s and policy π_j for the other agent.

the probability distribution of the underlying state depends on the policies of the other agent and we assume no prior knowledge about the other agent, all policies that may contribute to this maximum for some policy of the other agent must be retained. Policies that do not contribute to the maximum may be removed, or *pruned*. A linear program can be used to prune policies, and such a linear program is shown in Table 2.2. The program is visually depicted in Figure 2.6(b). A set U is constructed of policies that will be retained. This set is initialized to empty. For each policy that is evaluated for pruning, we associate a hyperplane α with its value function (over states and policies of other agents), and then an associated value δ is found. δ represents the maximum contribution of that policy. The linear program seeks to find a belief point for which the policy represented by α contributes value more than any other member of the set U. Thus, if $\delta \leq 0$, the policy does not contribute and can be pruned.

While this method guarantees that an optimal tree will be found, the set of trees may be very large and many unnecessary trees may be generated along the way. The pruning may be unhelpful, as even if a single policy of the other agent allows a policy to minimally contribute, it will not be pruned. Because of this, many Dec-POMDP solution techniques require an intractable amount of memory for any but the smallest problems.

Figure 2.5(b) shows the result of a full-backup where pruning did not eliminate any policies. Policy Trees P_1 and P_2 are represented within the squares. A full backup will generate every possible action and observation sequence. Note that only the new policies rooted in action A_1 are shown. In this case, with 3 possible observations, 3 actions, and two previous policies, a full backup would generate $3 * 2^3$ or 24 new policies.

Amato, Carlin, and Zilberstein prune additional policy trees through a technique called epsilon pruning [4]. The technique is shown visually in 2.6(c). At each step, a full backup is performed just as in [54]. Then the undominated set is iteratively constructed. A policy is added to the undominated set if there exists a belief state as well as a policy for the other agents, such that the new policy exceeds the value of the current undominated set by more than epsilon. The same linear program in Table 2.2 is used for this step, but the policy is only pruned if $\delta < \epsilon$. The epsilon pruning technique thus produces solutions with an error bound of $(n)(\epsilon)(T)$, where ϵ is a user specified parameter and n is the number of times that epsilon pruning is performed per horizon step. This works well in domains with a small number of observations. But in domains with a large number of observations, the number of new policies generated before pruning will be exponential in the number of observations. This motivates the development of additional pruning that specifically targets the number of observations.

2.2.4.1 JESP approach

The *JESP* (Joint Equilibrium Search) algorithm finds an equilibrium of the single-agent policies [85]. It does so by fixing each of the policies except for one. Then, the final policy is solved as a POMDP, while considering the other fixed policies as part of the environment. This solution may achieve poor results, as the equilibrium point found may be a local but not global optimum.

2.2.5 Top-down approaches

Another approach is to construct trees from the top-down, starting with the first step. In the MAA* algorithm, an A* search is constructed of joint policies [138]. The basic A* search algorithm is typically used in shortest-path problems in which an agent must plan a route; it must traverse a set of nodes on its way towards a goal, and it is given distances between pairs of nodes. A* decomposes $f(n)$, the estimated distance from each node to the goal, into two elements:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost of a solution path from the start to node n, and $h(n)$ is a heuristic estimate of the completion of that function to the goal. The A* algorithm works if the heuristic never overestimates the distance to the goal. That is, the heuristic is always either correct or overly optimistic.

In the case of a Dec-POMDP, $g(n)$ can be assigned the value of the first steps of a policy. This value is computed, thus it is known and exact. Next, a heuristic function $h(n)$ is constructed which is overly optimistic. For example, the *MDP heuristic* views the problem as an MDP, centralized and completely observable. The *POMDP heuristic* views the problem as centralized but not completely observable.

Though the heuristics can be computed in a tractable amount of time, one difficulty with this approach is that the search space, through the set of all possible joint policies, is large.

Algorithm 2: The *MBDP* algorithm

```
begin
    $MaxTrees \leftarrow$ max number of trees before backup;
    $H \leftarrow$ pre-compute heuristic policies for each $h \in H$;
    $\Pi_i^1, \Pi_j^1 \leftarrow$ initialize all 1-step policy trees;
    for $t=1$ to $T$ do
        $\Pi_i^{t+1}, \Pi_j^{t+1} \leftarrow fullbackup(\Pi_i^t), fullbackup(\Pi_j^t)$;
        $Sel_i^{t+1}, Sel_j^{t+1} \leftarrow empty$;
        for $k = 1$ to $maxTrees$ do
            choose $h \in H$ and generate belief state $b$;
            for each $\pi_i \in \Pi_i^{t+1}$ do do
                evaluate each pair $(\pi_i, \pi_j)$ with respect to $b$;
            add best policy trees to $Sel_i^{t+1}$ and $Sel_j^{t+1}$;
            delete those policy trees from $\Pi_i^{t+1}$ and $\Pi_j^{t+1}$;
        $\Pi_i^{t+1}, \Pi_j^{t+1} \leftarrow Sel_i^{t+1}, Sel_j^{t+1}$;
    select best joint policy tree $\delta^T$ from $\{\Pi_i^T, \Pi_j^T\}$;
    return $\delta^T$;
end
```

2.2.6 *MBDP* -based planners

The *MBDP* algorithm combines elements of top-down and bottom-up planning [120]. The algorithm is somewhat analogous to the single-agent point-based algorithms. In *MBDP* , joint policies are generated from the bottom-up. A set of belief points are selected, just as in the single-agent planners. However, instead of retaining all possible unpruned policies at each horizon step, only one joint policy is retained for each selected belief state. The belief states are selected by a similar set of heuristics used in MAA*.

The number of belief points selected at each step is set to a predefined constant $MaxTrees$. At each step, a full backup produces $|A|MaxTrees^{|\Omega|}$ new policies. Then for each selected belief point, the joint policy with the highest value for that belief point is retained. Policies not retained are pruned. Thus, the number of policies for each agent is pruned to $MaxTrees$ before the next full backup is performed.

Algorithm 2 shows the *MBDP* algorithm. Sets of policy trees are represented by π_i and π_j. The sets are initialized to the one-step policy trees. Then, at each step, a set of new policy trees are generated, based on the trees retained from the previous step. This set of trees is then pruned. One joint policy is retained for each point considered in the point-based backup. The process then repeats, up to horizon T.

2.2.6.1 IMBDP

The *MBDP* algorithm is still exponential in $|\Omega|$ due to the large number of policies generated with each backup. The *IMBDP* planner attacks this problem by limiting the number of possible observations to a constant $MaxObs$ [119]. *IMBDP* selects the most likely observations from heuristically determined belief states, and its backup step only generates subpolicy combinations under those observations. This process is termed a *partial backup*. Complete policies are then ”filled up” with the missing observations, by selecting the best available policies for these observations. Thus, for instance if a partial backup were run for the policies shown in part A of Figure 2.5, and a $MaxObs$ of two were used, only 12 new policies would be generated, 4 for each root action. Although this process is quick compared to a full backup, the error is only bounded for the heuristically selected belief state. The error is given by:

$$T^2(1-\epsilon)(R_{max} - R_{min}) \tag{2.23}$$

where T is the horizon of the problem, ϵ is the probability of receiving one of the remaining observations, and R is the reward function. In *IMBDP*, the probabilities of these observations are taken into account, but not their values.

2.2.6.2 Recent work

Recent extensions to *MBDP* and *IMBDP* involve quicker searching for optimal joint policies from each belief point. *PBIP* [42] involves a search for a distributed subpolicy from each belief point, starting from the underlying MMDP (that is, the equivalent problem if the Dec-POMDP were fully observable and centralized), and then conducting a search to resolve conflicts. Other recent work involves finding the best joint policy for each belief point by solving a constraint satisfaction problem [71].

There has been some work on formulating a Dec-POMDP as an integer program [9]. From the start state, a joint policy is found which maximizes value, which can be expressed as the sum of the values of immediate reward and expected value of subpolicies. Variables are the policies used by each agent. However, the choice of policies is subject to a number of constraints, agents must select only one subpolicy for each observation, for example.

A recent approach of Wu et al. combines *MBDP* , integer programming, and equilibrium search [152]. The *PBPG* algorithm performs a point-based backup and heuristically estimates belief points, as in *MBDP* . However, it does not perform a space-consuming full-backup as in *MBDP*, and it does not discard observations as in *IMBDP*. Instead, it finds a policy equilibrium for the point, as in *JESP* . It fixes policies of all the agents except for one, and finds the resulting best policy of the remaining agent by forming solving a linear program. Because the equilibrium is found at each step, for a variety of points, the algorithm avoids the accumulation of error over multiple steps as in *JESP* . Other recent approaches include incremental policy generation which combines the *PBIP* algorithm with a state-pruning measure, which eliminates unreachable states from evaluation as backup is performed [5]. The *GMAA* *-Cluster algorithm performs an A* search while clustering on action-observation histories to speed up the search.

2.3 Related work in Dec-POMDP communication

There are numerous communication strategies in the literature, here we provide an overview. Xuan and Lesser start with Centralized Policies (CPs), which assumes communication, and use these as the basis to form Decentralized Policies where unnecessary communication is eliminated [153]. To decide upon communication, an agent must reason about whether its own history is ambiguous to other agents. Roth et al. use this form as reasoning as well for the *tell* model of communication [106].

Nair et al. integrate communication strategies with the *JESP* algorithm, and maintain computational tractability by forcing communication to occur at least every K steps [86]. Mostafa and Lesser analyze domains where agent interactions are limited, and exploit this structure to create a more tractable problem with limited communication points [83]. Williamson et al. use KL-divergence of beliefs as an estimate of when communication is necessary, and use this to shape the reward function [150]. Shen et al. generate abstractions of information, and use the abstractions to limit the number of messages that need to be sent [122].

Goldman and Zilberstein introduce the Decentralized Partially Observable Markov Decision Process with Communication (Dec-POMDP-Comm) model [50]. This extends the Dec-POMDP model with the following augmentations.

- Σ is an alphabet of messages. $\sigma_i \in \Sigma$ denotes an atomic message sent by agent i. There is also a special null message ϵ.
- $C_\Sigma : \Sigma \rightarrow \Re$ is the cost of transmitting an atomic message. The cost of sending the null message is zero.

Agents in this model keep two policies, a local policy and a communication policy. The local policy is a mapping from observation history and received message history to an action. The communication policy is a mapping from observation history and received message history to a communication choice from the alphabet.

Another model of communication, used in subsequent chapters, is the sync model, where each message is not specified, but rather a single, special sync capability is assumed, which allows agents to mutually exchange histories. Communication is instantaneous; a message is received without delay as soon as it is sent. Also defined is C, a fixed cost on each step of communicating these synchronization messages. The fixed cost of C is incurred if *any* number of agents choose to communicate. Otherwise, if no agent communicates, they incur no penalty.

Using the sync model, one decision theoretic means of making a communication decision is to use a Value of Information (VoI) approach. In this type of approach, each agent individual finds an expectation over the value of joint policies after communication, versus an expectation over the value without communication. However, as we will see in future sections one weakness of these policies is that they are myopic. These policies can be perspective of other agents and to make joint communication decisions. They can also be improved by attaching a value to deferred communication, and weighing that against the value of immediate communication [13].

Becker et al. use a value of information approach under conditions of full local observability and transition independence, each agent finds an expected value of communication which is computed as the difference between the value of the resulting joint policy after communication and the value of continuing without communication [13]. These values are used to find a Nash equilibrium among one-step communication policies. From the Nash equilibrium, multi-step communication policies are constructed by comparing the value of communicating at the current step, to the value of deferring communication or not communicating at all.

CHAPTER 3

MONITORING AND CONTROL OF ANYTIME PROBLEM SOLVING WITH MULTIPLE AGENTS

This chapter considers multiagent settings in which a group of agents is engaged in collaborative decision making. Specifically it considers settings with multiple base-level algorithms, and where each base-level algorithm solves a component of an overall problem using an anytime algorithm, as defined in Chapter 2.1.3. A meta-reasoning agent supervises each base-level computation. This chapter extends the work of Hansen and Zilberstein as described in Chapter 2.1.4 from the single-agent to the multi-agent case. When the separate anytime algorithms that compose multi-agent base-level reasoning are independent of each other, i.e. each meta-reasoning agent can only affect the utility of local base-level computation, the single-agent methods extend to multiple agents without modification. In this case each agent governs an anytime algorithm, and meta-reasoning can be achieved through the single-agent monitoring and control algorithms of Hansen and Zilberstein [55].

This chapter specifically addresses the non-trivial case where utilities are joint utilities. However it assumes computation quality of each base-level algorithm remains independent of the status of the other algorithms (this notion will be formalized in the next section). This chapter analyzes two aspects of meta-reasoning, the ability to stop local computation and begin execution, and the ability to monitor computation.

The contributions of this chapter are as follows:

- *Formalization of the problem of monitoring and control of multiple anytime algorithms*. In the formalization, meta-reasoning agents monitor base-level algorithm quality and control when to stop base-level reasoning and begin execution. The formulation depends on whether meta-reasoning agents may monitor performance of a local algorithm or whether agents may monitor global performance. The formulation also depends on whether monitoring is relatively inexpensive, or whether it has cost. Section 3.1 outlines a taxonomy of problems related to all of these cases.

- *Complexity analysis* of several variants of the decentralized monitoring problem. Section 3.2.1 shows that decentralized monitoring is NP-complete.
- *Development of solution algorithms* for the decentralized monitoring problem. Section 3.2.2.2 maps decentralized monitoring to a bilinear formulation, and utilizes the bilinear programming technique of Petrik and Zilberstein in order to provide solutions [94].
- *Analysis of decentralized monitoring algorithms.* Section 3.4 provides an empirical analysis of the performance of the resulting algorithm. We find that optimal algorithms outperform naive algorithms under circumstances where either the difference in expected value between stopping and non-stopping is small, or the utility function is asymmetric.

The theoretical work presented this chapter was first published at AAMAS 2011 [32]. It was also summarized in a talk at the NIPS 2010 workshop "Decision Making with Multiple Imperfect Decision Makers", where the talk focused on the relevance of these concepts to prescriptive decision theory with imperfect participants [157]. The organizers of this workshop have published a related book, titled "Decision Making with Imperfect Decision Makers", which contains extended theory and experiments [33]. The notation used in these works has been modified to conform with this thesis.

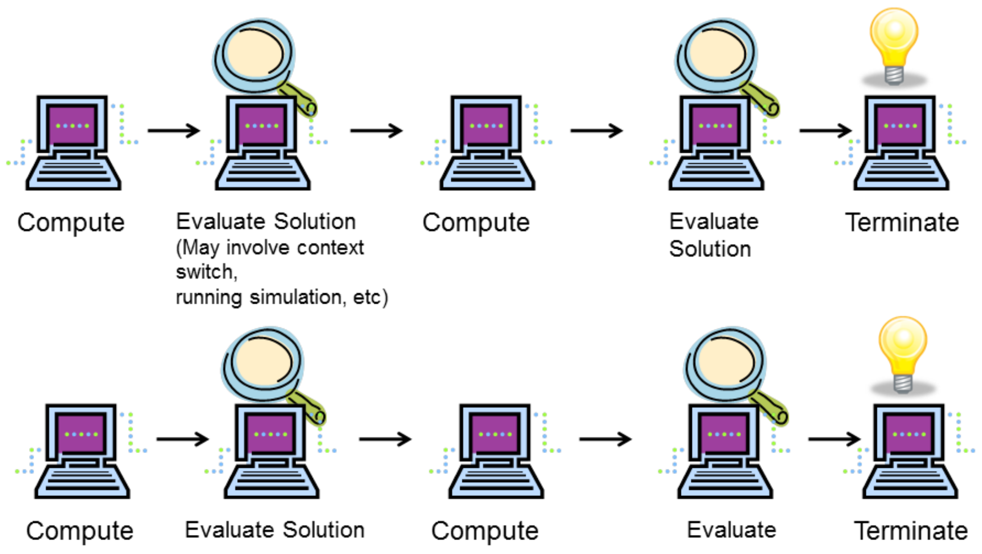


Figure 3.1. Illustration of a joint computation sequence involving local monitoring.

The outline of the rest of this chapter is as follows: Section 3.1 will formalize the notion of decentralized monitoring for the cases of both local monitoring (in which a meta-reasoning agent can monitor a local computation) as well as global monitoring (in which meta-reasoning agents share the status of all base-level algorithms). Next, in Section 3.2 the complexity of decentralized monitoring will be analyzed. It will then be shown how expected value of computation, such as the methods described in Section 2.1, can be used to provide sub-optimal solutions. In Section 3.2.2.2, the bilinear programming technique of Petrik and Zilberstein will be utilized to form optimal solutions for the case of local monitoring. After that, in Section 3.3 the case of global monitoring will be analyzed. Finally, in Section 3.4 empirical results will be provided, and the algorithms in this chapter will be characterized through the empirical results.

3.1 The Decentralized monitoring problem

Definition 3.1. *The* Decentralized Monitoring Problem *(DMP) is defined by a tuple* $< Ag, Q, A, T, P, U, C_L, C_G >$ *such that:*

- Ag *is a set of agents. Each agent supervises an anytime algorithm.*
- Q *is a set* $Q_1 \times Q_2 \times ... \times Q_n$*, where* Q_i *is a set of discrete quality levels for agent* i *(we will use the term "quality level", or just "quality", interchangeably). At each step* t*, we denote the vector of agent qualities by* $\vec{q}^t$*, or more simply by* $\vec{q}$*, whose components are* $q_i \in Q_i$*. Components of* $\vec{q}^t$ *are qualities for individual agents. We denote the quality for agent* i *at time* t *by* q_i^t*.*
- $\vec{q}^0 \in Q$ *is a joint quality at the initial step, known to all agents.*
- $A = \{$*continue, stop, monitorL, monitorG*$\}$ *is a set of metalevel actions available to each agent. The actions monitorL and monitorG represent local and global monitoring, respectively.*
- T *is a finite horizon representing the maximum number of time steps in the problem.*
- P_i *is the quality transition model for the "continue" action for agent* i*. For all* $i, t \in \{0..T-2\}$*,* $q_i^t \in Q_i$*, and* $q_i^{t+1} \in Q_i$*,* $P_i(q_i^{t+1}|q_i^t) \in [0,1]$*. Furthermore,* $\sum_{q_i^{t+1} \in Q_i} P_i(q_i^{t+1}|q_i^t) = 1$*. We assume that the quality transition models of any two agents* i *and* j *are independent of each other, that is,* $P_i(q_i^{t+1}|q_i^t, q_j^t) = P_i(q_i^{t+1}|q_i^t)$*.*
- $U(\vec{q}, t) : Q \to \Re$ *is a utility function that maps the value of solving the overall problem with quality vector* $\vec{q}$ *at time* t *to a real number.*
- C_L *and* C_G *are positive costs of the local monitoring and global monitoring actions respectively.*

Each agent solves a component of the overall problem using an anytime algorithm. Unless a "stop" action is taken by one of the agents, all the agents continue to deliberate for up to T time steps. Agents must decide whether to accept the current solution quality or continue deliberation, which results in a higher expected solution quality for typical transition models, but protracted deliberation results in decreased utility under typical utility models where longer computation time is assumed to be less desirable.

At each time step, agents decide which option to take, to continue, stop, or monitor globally or locally. If all the agents choose to continue, then the time step is incremented and solution quality transitions according to P_i. However, agents are unaware of the new quality state determined by the stochastic transitions, unless they monitor. If any agent chooses to "stop", then all agents are instructed to cease computation before the next time step, and the utility $U(\vec{q}, t)$ of the current solution is taken as the final utility[1]. If an agent chooses to monitor locally, then a cost of C_L is subtracted from the utility $U(\vec{q}, t)$ (for each agent that chooses monitorL) and the agent becomes aware of its local quality at the current time step. If any agent chooses to monitor globally, a single cost of C_G is subtracted from the utility and all agents become aware of all qualities at the time step. The time step is not incremented after a monitoring action. After an agent chooses to monitor, it must then choose whether to continue or stop, at the same time step.

Agents are assumed to know the initial quality vector $\vec{q}^0$. As stated above, an agent has no knowledge about quality after the initial step, unless a monitoring action is taken. The "monitorL" action monitors the local quality; when agent i takes the "monitorL" action at time t it obtains the value of q_i^t. However, it still does not know any compo-

[1]One likely implementation would be to send a system-wide interrupt message to begin execution, and allow each agent to service this event. Although this stopping criterion may seem restrictive, we note that the bilinear formulation developed in this chapter can be easily modified to represent other types of stopping criteria depending on user needs.

Case	Mon. Cost	Global/ Local	Number of DMs	Complexity	Model	Comments
1	0	Global	1	Poly	MDP	Centralized
2	0	Global	> 1	Poly	MMDP	Centralized
3	0	Local	1	Poly	MDP	Actions are continue/stop
4	C_L	Local	1	Poly	MDP/POMDP	Section 3.2.2.1
5	0	Local	> 1	NP-c	TI Dec-MDP	Theorem 3.1
6	C_L	Local	> 1	NP-c	TI Dec-MDP	Theorem 3.1
7	C_G	Global	> 1	NP-c	Dec-MDP-Comm-Sync	Theorem 3.1

Table 3.1. Different variants of the Distributed Monitoring Problem (DMP). Global/Local refers to whether the monitoring decision monitors anytime algorithms running on the local agent, or on all other non-local (global) agents. Number of DM refers to the number of decision makers. Complexity is denoted with respect to number of quality levels, for a constant number of agents. Model refers to the model used to solve the problem, in the section referenced in the comments column.

nent of the qualities of the other agents (denoted $\vec{q}^{\,t}_{-i}$). A "monitorG" action results in communication among *all* the agents, after which they all obtain the joint quality $\vec{q}^{\,t}$.

Since a "stop" action on the part of any individual agent ceases computation for all agents, this would appear to imply coordination of agent decisions, which in turn may imply that optimal joint decisions are computationally complex. This motivates an analysis of complexity of the model. In the next section, we will summarize complexity results under various conditions, and outline how the complexity can be reduced.

The complexity of the DMP model can vary according to the number of agents performing meta-reasoning, termed decision making agents (DMs), whether monitoring is local or global, and whether there is a cost attached to monitoring. Table 3.1 summarizes how the problem can be varied along these axes, each case corresponds to a combination of these parameters. Monitoring cost is considered as either a constant or zero. When monitoring is local, the cost of global monitoring is assumed to be infinity, and when monitoring is global, the cost of local monitoring is assumed to be 0. The number of decision makers refers to the number of agents which may make stopping and monitoring decisions. When there is one decision maker, multiple anytime algorithms run, although only one is permitted to supervise stopping and monitoring decisions. As the table shows, this often results in a simpler problem, we will see later in the chapter that these cases have simpler complexity because agents do not need to coordinate their meta-reasoning.

The first two cases in the table represent cases where global awareness of the quality level of other agents can be achieved at no cost. In this case, decision making can be made in a centralized fashion by applying the technique of Section 2.1.4 [55]. The third and fourth cases represent problems where agents may monitor their own quality, but cannot monitor the other agents. Section 3.2.2.1 will show that these cases can be handled through dynamic programming. The fifth, sixth, and seventh cases contain more than one decision maker. We will show that these problems are reducible to the Dec-MDP model [17]. Finally, the seventh row contains global monitoring, which we will show is reducible to Dec-MDP-Comm-Sync [13].

3.1.1 Transition-Independent decentralized MDP

A transition-independent decentralized MDP (sometimes referred to as TI Dec-MDP) is composed of n cooperative agents, each agent i working on its own local subproblem that is described by an MDP, $\langle S_i, A_i, \mathcal{P}_i, R_i, T \rangle$ [1] We deliberately use a notation that overlaps the DMP definition, because as we will see, many of the components of the TI Dec-MDP tuple will directly correspond to components of the DMP

[1]A transition-independent decentralized MDP can be related to the Dec-POMDP model of section 2.2.1, by assuming (1) The problem is fully observable given the joint observations. (2) The state, action, and transition probabilities are factored by agent and transition independent, as described in the text. For more detail on the relationship of TI Dec-MDP to Dec-POMDP, see [118]

definition. In TI Dec-MDP, the local subproblem for agent i is independent of local subproblems for the other agents. It is also completely observable, but only by agent i. At each step agent i takes action $a_i \in A_i$, transitions from state $s_i \in S_i$ to $s'_i \in S_i$ with probability $\mathcal{P}_i(s'_i|s_i, a_i)$, and receives reward $R_i(s'_i)$. All agents are aware of the transition models of all the other agents (but not necessarily their states or choice of actions at runtime). The *global* state of the domain is composed of the local states of all the agents. T is a finite number of horizon steps of the problem.

A communicative variant of this problem is referred to as Dec-MDP-Comm-Sync. At each time step, each agent first performs a domain-level action (one that affects its local MDP) and then a communication choice. The communication choices 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 corresponds to the sync model of communication [154], as it synchronizes the world view of the agents, providing each agent complete information about the current world state. The cost of communication is a constant C if at least one agent chooses to communicate, 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 $\langle s_{1 \ldots} s_n \rangle \in \langle S_{1 \ldots} S_n \rangle$, and the time drawn from a set $\mathcal{T}$ (whose elements are between 0 and T) since the last synchronization, to a domain-level action and a communication choice, $\pi_i : S_i \times \langle S_{1 \ldots} S_n \rangle \times \mathcal{T} \rightarrow A_i \times \{\text{yes}, \text{no}\}$, where yes and no refer to decisions to communicate and not to communicate respectively. We will occasionally refer to domain-level policies and communication policies as separate entities, which are the mappings 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 $R : S_1 \times ... \times S_n \to \Re$. To the extent that the global reward function depends on past history, the relevant information must be included in the local states of the agents just as with 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.

Let a_i denote the domain-level action selected by $\pi(s_i)$. The global value function is:

$$V(s_1 ... s_n) = \sum_{i=1}^{n} R_i(s_i) + R(s_1 ... s_n) + \sum_{s'_1 ... s'_n} \mathcal{P}(s'_1..s'_n | s_1...s_n, a_1..a_n) V(s'_1 ... s'_n) \quad (3.1)$$

Transitions on the MDPs are independent of each other; we will therefore assume that without communication $\mathcal{P}(s'_1, ...s'_n | s_1..s_n, a_1..a_n) = \prod_{i=1}^{n} \mathcal{P}_i(s'_i | s_i, a_i)$.

The complexity of finding optimal policies for both Dec-MDP with transition independence, and the Dec-MDP-Comm-Sync classes of problems has been shown to be NP-complete [51], which is lower than the doubly exponential complexity (NEXP-hard) of general decentralized decision making. In the next section, we will show how a DMP problem with local monitoring can be formulated as a transition independent Dec-MDP, thus maintaining NP-completeness.

3.2 Local monitoring

In this section, we examine the concept of local monitoring and control. That is, each meta-level agent must decide whether to continue base-level computation, stop immediately, or to monitor progress at a cost C_L, and then decide whether to continue or stop deliberation and initiate joint execution. The main result in this section will prove that a DMP with local monitoring decisions can be solved by first converting the problem to a transition independent Dec-MDP. Although the termination decision may seem to imply transition dependence (a “stop” decision by any single agent stops all the agents), the dependence is eliminated in the construction of Theorem 1.

3.2.1 Complexity of local monitoring

When $C_L = 0$, each agent should choose to monitor locally on every step, since doing so is free. When $C_G = \infty$, agents should never choose to monitor globally. The following lemma and theorem shows that even for the simpler case where $C_L = 0$, $C_G = \infty$, and number of agents is fixed, the problem of finding a joint optimal policy is NP-complete in the number of quality levels. The termination decision alone, made by agents with local views of quality, is NP-hard.

Lemma 3.1. *The problem of finding an optimal solution for a DMP with a fixed number of agents $|Ag|$, $C_L = 0$ and $C_G = \infty$ is NP-hard.*

Proof. A nearly identical problem to this special-case DMP with zero monitoring cost is the Team Decision Problem (TDP) introduced by Tsitsiklis and Athans [141]. Unfortunately, unlike in the Team Decision Problem, three joint decisions of a two-agent DMP (when either agent stops, or they both do) contain the same utility. Therefore we proceed directly to the underlying Decentralized Detection problem upon which the complexity proof of TDP is established.

We show that the NP-complete Decentralized Detection (DD) problem can be solved by a three step DMP. The following definition is adopted from [141].

Decentralized Detection: Given finite sets Y_1, Y_2, a rational probability mass function z mapping $Y_1 \times Y_2$ to the set of rational numbers , a partition $\{A_0, A_1\}$ of $Y_1 \times Y_2$. The goal is to optimize $J(\gamma_1, \gamma_2)$ over the selection of $\gamma_i : Y_i \rightarrow \{0, 1\}$, $i = 1, 2$, where $J(\gamma_1, \gamma_2)$ is given by

$$\sum_{(y_1,y_2)\in A_0} z(y_1,y_2)\gamma_1(y_1)\gamma_2(y_2) + \sum_{(y_1,y_2)\in A_1} z(y_1,y_2)(1-\gamma_1(y_1)\gamma_2(y_2)) \tag{3.2}$$

Decentralized detection can be polynomially reduced to a three step DMP (Definition 1) with $C_L = 0$ and $C_G = \infty$. The first step is a known joint quality $\vec{q}^0$. We define a unique quality level at the second and third step for each $y_i \in Y_i$. We will denote the quality level representing y_i by q_{yi}. Transition probabilities to the second step are defined by the probability mass function, $P_i(q_i^2, q_j^2) = z(y_1, y_2)$. Each agent then monitors (for zero cost) and is aware of its local quality.

We model the decision of selecting $\gamma_i = 1$ as a decision by agent i to continue, and of selecting $\gamma_i = 0$ as a decision by agent i to terminate. To accomplish this, the DMP transition model transitions deterministically to a unique quality at step 3, for each quality of step 2 of each agent.

Utility on step 3 is defined so that

$$U(q_{y1}^2, q_{y2}^2, 2) = 0, U(q_{y1}^3, q_{y2}^3, 3) = 1 \quad \text{iff } (y_i, y_j) \in A_0$$

and

$$U(q_{y1}^2, q_{y2}^2, 2) = 1, U(q_{y1}^3, q_{y2}^3, 3) = 0 \quad \text{iff } (y_1, y_2) \in A_1$$

The mapping to DMP is polynomial, as a quality level was created for each member of the decision sets, for three time steps. It should be clear from this construction that an optimal continuation policy which maps q_{yi} to a decision to continue or terminate, can be used to construct $\gamma(y_i)$ in DD. Since DD is NP-complete in the size of the finite sets, DMP must be at least NP-hard in the number of quality levels. $\square$

To show that DMP is in NP, we will reduce to a transition independent Decentralized MDP (Dec-MDP), a problem which was shown by Goldman and Zilberstein to be NP-complete [51].

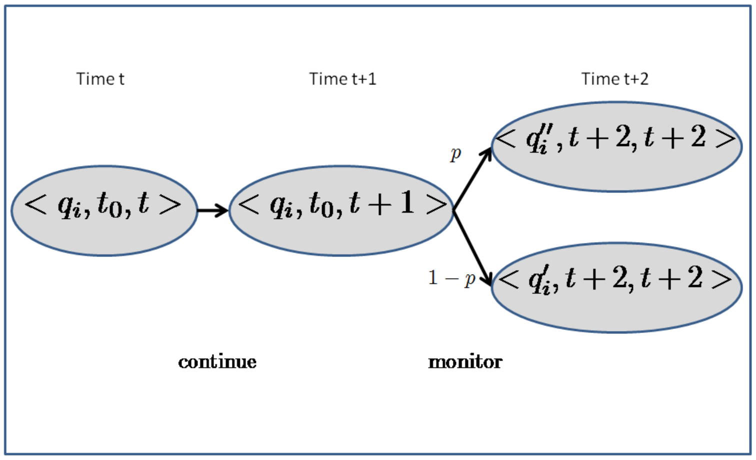


Figure 3.2. An example of the state space for one of the agents, while running the Dec-MDP construction of Theorem 1. When the agent continues, only current time is incremented. When the agent monitors, the agent stochastically transitions to a new quality state based on its performance profile, the current time increments, and monitoring time is set to the current time. Not shown, when either agent terminates, the agents get a reward based on the expectation of utility over their performance profiles.

Theorem 3.1. *DMP Local Monitoring Complexity:*

The problem of finding an optimal solution for a DMP with a fixed number of agents, $C_L = k$ and $C_G = \infty$ is NP-complete.

Proof. NP-hardness follows Lemma 1, with $k = 0$ as a special case. To show NP-completeness, we show that the problem can be reduced to a transition independent Dec-MDP. Policies and policy-values for the DMP will correspond to policies and policy-values for the TI Dec-MDP. The conversion is as follows:

The **state space** S_i for agent i is a tuple $< q_i, t_0, t >$, where q_i is a quality level (drawn from Q_i), t_0 is the time step at which that quality level was monitored, and t is the number of the current time step. We also define a special "terminal" state for each agent.

The **action space** for all agents is $\{$terminate, continue, monitorL$\}$.

The **transitions** consist of the following. For the continue action:

$$\mathcal{P}(< q_i, t_0, t+1 > \mid < q_i, t_0, t >, \text{continue}) = 1$$
$$\mathcal{P}(< q_i, t_0, t' > \mid < q_i, t_0, t >, \text{continue}) = 0, \ \ \forall t' \neq t+1$$
$$\mathcal{P}(< q_i, t_0', t' > \mid < q_i, t_0, t >, \text{continue}) = 0, \ \ \forall t_0' \neq t_0$$
$$\mathcal{P}(< q_i', t_0', t' > \mid < q_i, t_0, t >, \text{continue}) = 0, \ \ \forall q_i' \neq q_i$$

When the action is to terminate, the agent transitions to the terminal state. Let $\mathcal{P}(s'|s,a)$ be the transition function, recalled from the TI Dec-MDP definition. When the action is to monitor, we have $\forall q_i, q_i' \in Q_i :$

$$\mathcal{P}(< q_i', t_0', t' > \mid < q_i, t_0, t >, \text{monitor}) = 0 \ \text{ if } t' \neq t \text{ } or \text{ } t' \neq t_0'.$$
$$\mathcal{P}(< q_i', t_0', t' > \mid < q_i, t_0, t >, \text{monitor}) = P(q_i^{t'} | q_i^{t_0}) \text{ if } t' = t_0' = t$$

The **reward** is defined as zero if all actions choose to continue and as $U(\vec{q}, t)$ (from Definition 1) if one of the agents chooses to terminate and none of the agents are in a terminal state. Reward is adjusted by $-C_L$ for each monitoring action. Superseding these rules, reward is zero if any agent is in a terminal state.

This reduction is polynomial, as the number of states in the Dec-MDP for agent i is $|Q_i|T^2$ and the number of actions is 3. The representation is transition-independent, as the state of each agent does not affect the state of the other agents. Note that when one agent terminates, the other agents do not enter a terminal state, such a specification would violate transition independence. Rather, this notion, that no reward is accumulated once any agent has terminated, is captured by the Reward function. No reward is received if any of the agents are in a terminal state. Since reward is only received when one of the agents enters the terminal state, reward is only received once, and the reward received by the Dec-MDP is the same as the utility received by the DMP.

Figure 3.2 shows a visual representation of the Dec-MDP reduction from a DMP with local monitoring costs. The state is a tuple consisting of a quality level, the time at which the quality was monitored, and the current time. The "continue" action in the first step increments the current time. The "monitor" action increments the monitoring time to the current time, and probabilistically transitions quality according the transition probability of the DMP across multiple steps.

An optimal policy for the Dec-MDP produces an optimal policy for the corresponding multi-agent anytime problem. Note that the uncertainty of quality present when an agent does not monitor is simulated in the MDP. Even though, in an MDP, an agent always knows its state, in this reduction the transition is not executed until the monitoring action is taken. Thus, even though an MDP has no local uncertainty, an agent does not "know" its quality until the monitor action is executed, and thus the local uncertainty of the multi-agent anytime problem is represented.

□

3.2.2 Solution methods with local monitoring

3.2.2.1 Myopic solution

We first build a solution that adapts the single-agent approach of Hansen and Zilberstein to the multi-agent case [55]. The adaption considers the other agents to be part of the environment, and thus we name this myopic solution the Greedy approach. Greedy computation does not take into account the actions of the other agents; we will initiate a greedy computation by assuming that the other agents always continue, and that they will never monitor or terminate. Thus it applies to the case where there is one decision making agent. We will then build upon this solution to develop a *nonmyopic solution* for cases where there is more than one decision maker. For ease of explanation, we will describe the algorithm from a single agent's point of view. If there are multiple decision makers running greedy computation, it should be assumed that each agent is executing this algorithm simultaneously.

Each agent begins by forming a performance profile for the other agents. We will use the term Pr (with a subscript, as in Pr_i, when referring to agent i) as a probability function assuming only "continue" actions are taken, extending the transition model P_i (Definition 1) over multiple steps. Furthermore we can derive performance profiles of multiple agents from the individual agents, using the independence of agent transitions. For example, in the two agent case we use $Pr(\vec{q})$ as shorthand for $Pr(q_i)Pr(q_j)$.

Definition 3.2. *A dynamic local performance profile of an algorithm, $Pr_i(q_i'|q_i, \Delta t)$, denotes the probability of agent i getting a solution of quality q_i' by continuing the algorithm for time interval Δt when the currently available solution has quality q_i.*

Definition 3.3. *A greedy estimate of expected value of computation (MEVC) for agent i at time t is:*

$$\mathit{MEVC}(q_i^t, t, t+\Delta t) = \sum_{\vec{q}^t} \sum_{\vec{q}^{t+\Delta t}} Pr(\vec{q}^t|q_i^t, t) Pr(\vec{q}^{t+\Delta t}|\vec{q}^t, \Delta t)(U(\vec{q}^{t+\Delta t}, t+\Delta t) - U(\vec{q}^t, t))$$

The first probability is the expectation of the current global state, given the local state, and the second probability is the chance of transition. Thus, MEVC is the difference between the expected utility level after continuing for Δt more steps, versus the expected utility level at present. Both of these terms must be computed based on the performance profiles of the other agents, and thus the utilities are summed over all possible qualities achieved by the other agents. Cost of monitoring, C_L, is not included in the above definition. An agent making a decision must subtract this quantity outside the MEVC term.

For time-dependent utility functions, the agent faces a choice as to whether to continue and achieve higher quality in a longer time, or to halt and receive the current quality with no additional time spent. We call a policy that makes such a decision, a monitoring policy.

Definition 3.4. *A monitoring policy $\pi(q_i, t)$ for agent i is a mapping from time step t and local quality level q_i to a decision whether to continue the algorithm and act on the currently available solution.*

It is possible to construct a stopping rule by creating and optimizing a value function for each agent. First, create a new local-agent value function U_i such that

$$U_i(q_i, t) = \sum_{\vec{q}^{\,t}_{-i}} Pr(\vec{q}^{\,t}_{-i}) U(< q_i, \vec{q}_{-i} >, t)$$

Next, create a value function using dynamic programming one step at a time:

$$V_i(q_i, t) = \max_d \begin{cases} \text{if } d = \text{stop:} \\ U_i(q_i, t), \\ \text{if } d = \text{continue:} \\ \sum_{q_i^{t+\Delta t}} Pr(q_i^{t+\Delta t} | q_i) V_i(q_i, t + \Delta t) \end{cases}$$

to determine the following policy:

$$\pi_i(q_i,t) = \text{argmax}_d \begin{cases} \text{if } d = \text{stop:} \\ U_i(q_i,t), \\ \text{if } d = \text{continue:} \\ \sum_{{q_i}^{t+\Delta t}} Pr({q_i}^{t+\Delta t}|q_i^t)V_i(q_i, t+\Delta t) \end{cases}$$

In the above, Δt is assumed to be one and an agent makes the decision to continue or stop at every step. A stop action yields an expected utility over the qualities of the other agents. A continue action yields an expectation over joint qualities at future step $t+\Delta t$. The above definitions exclude the option of monitoring (thus incurring the costs C_L and C_G), the choices are merely whether to continue or act. Thus, we must define a cost-sensitive monitoring policy, which accounts for C_L and C_G.

Definition 3.5. *A cost-sensitive monitoring policy, $\pi_{i,C_L}(q_i,t)$, is a mapping from time step t and quality level q_i (as well as monitoring cost C_L) into a monitoring decision $(\Delta t, d)$ such that Δt represents the additional amount of time to allocate to the anytime algorithm, and d is a binary variable that represents whether to monitor at the end of this time allocation or to stop without monitoring.*

Thus, a cost-sensitive monitoring policy at each step chooses to either blindly continue, monitor, or terminate. It can be constructed using dynamic programming and the value function below. The agent chooses Δt, how many steps to continue blindly, as well as whether to stop or monitor after. If it stops, it receives utility as an expectation over the quality levels of the other agent, if it monitors it achieves the value of its future decisions from that quality level (known from prior computation from dynamic programming), adjusted by a penalty of C_L.

	$q_1^t = 1$	$q_1^t = 2$	$q_1^t = 3$
$q_2^t = 1$	**-2**	0	-1
$q_2^t = 2$	**5**	**-3**	**-1**
$q_2^t = 3$	**-2**	-1	1

Table 3.2. An example of a case where greedy termination policy produces a poor solution. Entries represent the expected utility of continuing for a step.

$$V_{C_L}(q_i,t) = \max_{\Delta t, d} \begin{cases} \text{if } d = \text{stop:} \\ \sum_{q_i{}^{t+\Delta t}} Pr(q_i{}^{t+\Delta t} | q_i^t) U_i(q_i, t+\Delta t) \\ \\ \text{if } d = \text{monitor:} \\ \sum_{q_i{}^{t+\Delta t}} Pr(q_i{}^{t+\Delta t} | q_i^t) V_{C_L}(q_i, t+\Delta t) - C_L \end{cases}$$

A greedy monitoring policy can thus be derived by applying dynamic programming over one agent. Working backwards, such an algorithm assigns each quality level on the final step a value, based on its expected utility over possible qualities of the other agents. Then, continuing to work backwards, it finds the value of the previous step, which is the max over: (1) the current expected utility over the possible qualities of the other agents (if it chooses to stop). (2) The expected utility of continuing (if it chooses to continue). An algorithm to find a cost-sensitive monitoring policy can similarly find the expectation over each time step with and without monitoring, and compare the difference to the cost of monitoring.

3.2.2.2 Solution methods: Nonmyopic policy

The greedy solution can be improved upon to coordinate policies among all the agents. To illustrate, examine Table 3.2. Each entry represents the expected joint utility of continuing (thus increasing utility but also time cost), minus the expected utility of stopping. Assume all entries have equal probability and the local monitoring cost is zero, and that the value of stopping immediately is zero, and thus the values shown represent only the value of continuing. Agent 1 would greedily decide to continue if it is in state $q_1^t = 1$ only, as that is the only column whose summation is positive. Agent 2 would greedily continue if it has achieved quality $q_2^t = 2$, as that is the only row whose summation is positive. However, this would mean that the agents continue from all joint quality levels which are bolded. The sum of these levels is negative, and the agents would do better by selecting to always terminate!

We solve the DMP with $C_G = \infty$ optimally by leveraging the bilinear program approach of Petrik and Zilberstein to solving transition independent Dec-MDPs [94]. The program is centralized at planning time and decentralized at execution time. We first convert the problem to the transition independent Dec-MDP model described above. We prune "impossible" state-actions, for example we prune states where $t_0 > t$, as an agent cannot have previously monitored in the future. Then we convert the resulting problem into a bilinear program. A bilinear program can be described by the following inequalities for the two-agent case (the framework is extensible beyond two agents if more agent-vectors are added).

$$maximize_{x,y}\, r_1^T x + x^T R y + r_2 y$$
$$subject\ to\ B_1 x = \alpha_1$$
$$B_2 y = \alpha_2$$

In our bilinear formulation of a DMP, each component of the vectors x represents a joint state-action pair for the first agent (similarly, each component of y represents a state-action for the second agent). Following the construction of Theorem 3.1, each component of x (and likewise, y) represents a tuple $< q_1^{t_0}, t_0, t, a >$ where q_1 represents the last quality observed, t_0 represents the time at which it was observed, t represents the current time, and a represents a continue, monitor, or terminate. Thus, the length of x is $3|Q_1|T^2$ (assuming no pruning of impossible state-actions). Each entry of x represents the probability of that state and action occurring upon policy execution.

The vectors r_1 and r_2 are non-zero for entries corresponding to state-actions that have non-zero local reward, for agents 1 and 2 respectively. We set these vectors to zero, indicating no local reward.

The matrix R specifies joint rewards for joint actions, each entry corresponds to the joint reward of a single state-action in x and y. Thus, entries in R correspond to the joint utility $U(\vec{q}, t)$ of the row and column state, for state-actions where all agents are not in the final state and any agent terminates. For each entry of R corresponding to a joint (non-final) state-action where one agent monitors and the other agent continues or terminates, that entry is adjusted by $-C_L$. Otherwise, joint reward is 0.

α_1 and α_2 represent the initial state distributions, and B_1 and B_2 correspond to the dual formation of the total expected reward MDP [98]. Intuitively, these constraints are very similar to the classic linear program formulation of maximum flow. Each constraint represents a state triple, and each constraint assures that the probability of transitioning to the state (which is the sum of state-actions that transition to it, weighted by their transition probabilities) matches the probability of taking the outgoing state-actions (which is the three state-actions corresponding to the state triple). A special case is the start quality, from which outgoing flow equals 1.

Bilinear programs, like their linear counterparts, can be solved through methods in the literature [94]. These techniques are beyond the scope of this chapter, one technique is to alternatingly fix x and y policies and solve for the other as a linear program. Although bilinear problems are NP-complete in general, in practice performance depends on the number of non-zero entries in R.

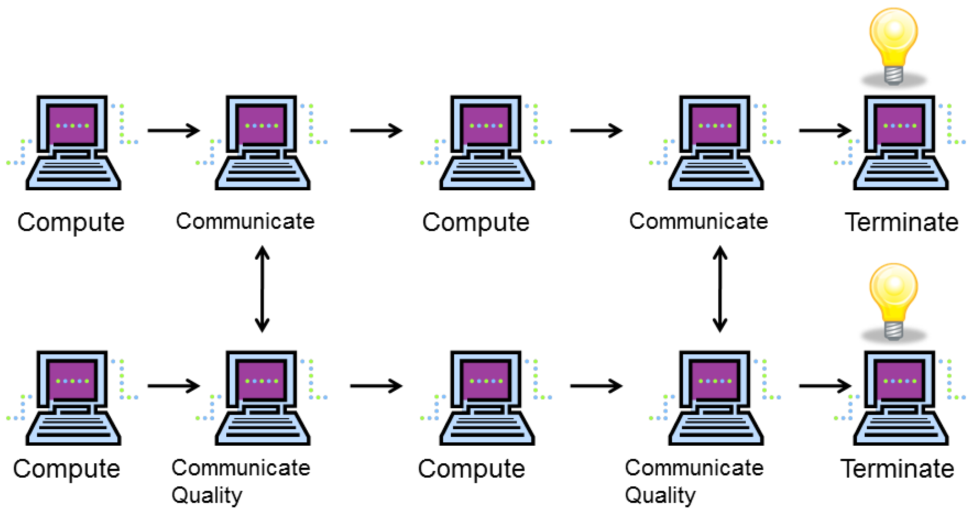


Figure 3.3. Illustration of a joint computation sequence involving global monitoring.

3.3 Global monitoring

Next, we examine the case where meta-level agents can communicate with each other (i.e., monitor globally). We will analyze the case where $C_L = 0$ and $C_G = k$, where k is a constant. For ease of description, we describe an on-line approach to communication. The online approach can be converted to an offline approach by anticipating all possible contingencies (Chapter 5 will provide more detail). We decide whether to communicate based on decision theory, agents compute Value of Information (VoI).

$$VoI = V^*(q_i, t) - V_{sync}(q_i, t) - C_G$$

where V^* represents the expected utility after monitoring, V_{sync} represents expected utility without monitoring (see below), and C_G is cost of monitoring. In order to support the computation of V_{sync} and V^*, joint policies are produced at each communication point (or, for the offline algorithm, at all possible joint qualities). We define a helpful term $V^*(\vec{q}, t)$, (which we will rewrite $V^*(q_i, \vec{q}_{-i}, t)$ to more clearly identify the local agent), which is the value of a joint (optimal and non-communicative) policy after communication and discovery of joint quality $\vec{q}$, as computed through the methodology of the last section with $C_L = 0$. From the point of view of agent i, the value after communicating can then be viewed as an expectation over the quality of the other agents, based on their profiles.

$$V^*(q_i, t) = \sum_{\vec{q}_{-i}} Pr(\vec{q}_{-i}, t) V^*(q_i, \vec{q}_{-i}, t)$$

Similarly, V_{sync} is the value attached to quality q_i and continuing without communicating. The value of this state-action was computed as part of the local monitoring problem at the last point of communication (which can be computed, for example, through the bilinear program of the previous section), we use the subscript "sync" to remind us that $V_{sync}(q_i, t)$ depends on the policies created and qualities observed at the last point of communication.

Non-myopic policies require each agent to make a decision as to whether to communicate or not at each step, resulting in the construction of a table resembling Table 3.2. We examined this table in a previous section when deciding whether to continue or stop. The table is used similarly for global monitoring, except the decision made by each agent is whether to communicate or not to communicate. Communication by either agent forces both agents to communicate and synchronize knowledge. Entries represent the joint state, and are incurred if *either* agent 1 decides to communicate from the row representing its quality, *or* agent 2 decides to communicate from the column representing its local knowledge.

This problem, of deciding whether to communicate after each step, is NP-complete as well. We will show this by reducing to a transition independent (TI) Dec-MDP-Comm-Sync [13]. A Dec-MDP-Comm-Sync is a transition independent Dec-MDP with an additional property: After each step, agents can decide whether to communicate or not to communicate. If they do not communicate, agents continue onto the next step as with a typical transition independent Dec-MDP. If any agent selects to communicate, then all agents learn the global state. However, a joint cost of C_G is assessed for performing the communication. Agents form joint policies at each time of communication. The portion of the joint policy formed by agent i after step t is denoted π_i^t.

Theorem 3.1. *DMP Global Monitoring Complexity: The DMP problem with $C_L = 0$ and C_G is a constant, is NP-complete.*

Proof. The proof of NP-hardness is similar to Lemma 1.

To show that the problem is in NP, we can reduce the problem to that of finding the solution of a Dec-MDP-Comm-Sync [13]. We create the following Dec-MDP-Comm-Sync from a DMP with $C_L = 0$.

- S_i is the set $Q_i \cup \{f_i\}$ for agent i, where f_i is a new "terminal" state for agent i.
- $A_i = \{\text{continue, terminate}\}$; the joint action set is $\prod_i A^i$.
- The transition model:

  $\mathcal{P}(q_i^{t+1}|q_i^t, \text{continue}) = P(q_i^{t+1}|q_i^t)$

  $\mathcal{P}(q_i^{t_2}|q_i^{t_1}, \text{continue}) = 0, \forall(t_2 \neq t_1 + 1)$

  $\mathcal{P}(f_i|q_i^t, \text{terminate}) = 1, \forall q_i^t \in Q_i$
- The reward function $R(\vec{q}^{\,t}, a_i) = U(\vec{q}, t)$ if a_i = terminate for some i; 0 otherwise.
- The reward function is 0 when any agent is in the final state.
- The horizon T is the same as T from the DMP.
- The cost of communication is C_G.

The reduction is polynomial as the number of states added is equal to T, and only one action is added. □

It is straightforward to verify that this reduction is polynomial. Having represented the DMP problem as a Dec-MDP-Comm-Sync, the task is now to solve this model. This will be the subject of Chapter 5. □

3.4 Empirical analysis

We experimented with two decentralized decision problems involving anytime computation. First we experimented on the `Rock Sampling` domain, borrowed from the POMDP planning literature. In this planning problem, two rovers must each form a plan to sample rocks, maximizing the interesting samples according its preferences. However, the locations of the rocks are not known until runtime, and thus the base-level plans cannot be constructed until the rovers are deployed. We selected the HSVI algorithm for POMDPs as the base-level planning tool [133]. HSVI is an anytime algorithm, the performance improves with time, its error bound is constructed and reported at runtime. Prior to runtime, the algorithm was simulated $10,000$ times on randomized Rock Sampling problems, in order to find the performance profile. The resulting profile held 5 quality levels over 6 time steps.

As a second base-level algorithm, we profiled `Max Flow`, the Ford Fulkerson solution method for computing maximum flow [45]. This motivating scenario involved a decentralized maximum flow problem where two entities must each solve a maximum flow problem in order to supply disparate goods to the customer. To estimate the transition model P in the DMP, we profiled performance of Ford Fulkerson through Monte Carlo simulation. The flow network was constructed randomly on each trial, with each edge capacity in the network drawn from a uniform distribution. Quality levels corresponded to regions containing equal-sized ranges of the current flow. From the simulation, a 3-dimensional probability table was created, with each layer of the table corresponding to the time, each row corresponding to a quality at that time, each column

Problem (Local/Global)	Compile Time	Solve Time
Max Flow Local	3.5	11.4
Rock Sample Local	.13	2.8
Max Flow Global	.04	370
Rock Sample Global	.01	129

Table 3.3. Timing results in seconds for non-myopic solvers. Compile Time represents time to compile performance profile into a bilinear problem, Solve Time measures time taken by the bilinear solver.

representing the quality at the next time step, and the entry representing the transition probability. We created software to compile a Decentralized MDP from the probability matrix, as described in the previous sections, and solved the resulting problem using a bilinear program.

Three parameters of utility were varied with respect to each other: the reward for increasing quality, a linearly increasing cost of time, and the cost of monitoring. We varied each in order to characterize the local monitoring algorithms described in previous sections. As a general characteristic, the cost of a time step in the utility function was a fraction of the benefit of a quality level, and the cost of monitoring was a fraction of the cost of a time step. This models algorithms whose profiles contain frequent discrete time steps with small disruptions (such as the cache misses, etc.) caused by monitoring. Algorithms considered were `Continue`, an algorithm which continues until the last time step without performance profiling, `Terminate`, an algorithm which terminates as its first decision, and `Greedy` and `Nonmyopic`, the greedy and nonmyopic local monitoring algorithms described previously.

Mean running time for the non-myopic (NM) variant of our algorithms is shown in Table 3.3. Compile Time represents the time taken to compile a Dec-MDP problem, and Solve Time represents the time taken to solve it. The Max Flow problem was larger than the Rock Sample problem (containing more quality levels), thus consumed more time.

The global formulations, as opposed to the local formulations, required a subproblem formulation to compute V^* at each communication point, and thus more time elapsed.

Figure 3.4 compares the various strategies, and shows the effect of varying the cost of time on solution quality. Utility functions used to produce this figure were

$$U(\vec{q}, t) = \max(q_i, q_j) - t \cdot TC$$

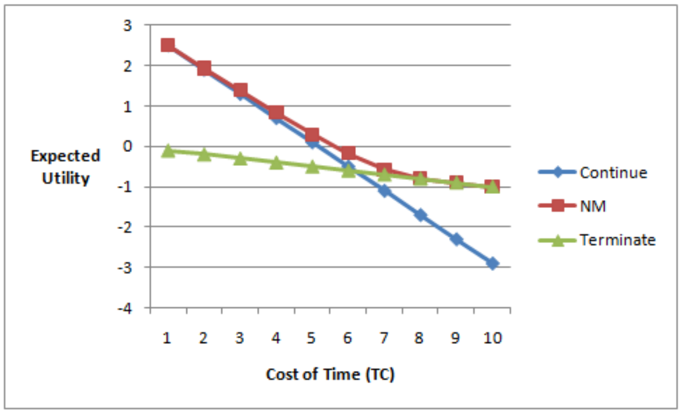
for `Rock Sampling` and

$$q_i + q_j - t \cdot TC$$

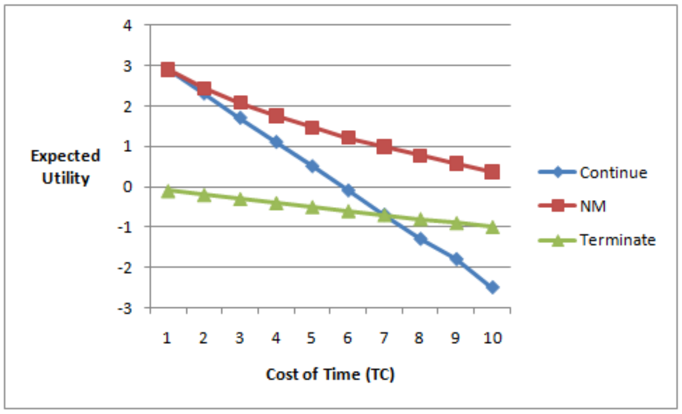
for maximum flow, where TC is a parameter representing cost of time. We assume the problem begins at $t = 1, q_1 = 0, q_2 = 0$, so the value of terminating on the first step is $0 - TC$ for both problems. As the figure shows, as cost of time decreases, the `Continue` strategy becomes closer to optimal, and when cost of time increases, the `Terminate` strategy approaches the optimal. Monitoring does not improve utility as time cost approaches zero or infinity, because the same stopping decisions apply to any current quality, without monitoring. The need for monitoring occurs in the middle of the graph, when the stopping decision is unclear.

Table 3.4 compares the `Greedy` and `Nonmyopic` strategies for `Rock Sampling`. In most cases, a greedy strategy can approximate the best possible local monitoring strategy rather well. However, the value at $TC = .5$ suggests that there may be certain parameters for which decisions of time quality trade-off are more difficult, and a monitoring strategy should be more tightly coordinated. Table 3.5 explores this issue further by varying C_L, when $TC = .5$. The `Greedy` strategy loses the ability to exploit monitoring quicker, and even when monitoring is inexpensive, the `Nonmyopic` strategy is able to make better use of monitoring.

To see why `Nonmyopic` outperforms `Greedy` under certain circumstances, we look more closely at the resulting policies. Table 3.6 shows resulting `Nonmyopic`



(a) Rock Sampling



(b) Max Flow

Figure 3.4. Comparison of nonmyopic monitoring strategy to non-monitoring strategies on two different domains. Cost of time is reported in tenths of a quality level. When the cost of time is low, a strategy to always continue is preferred. When cost of time is high, a strategy of immediately stopping is preferred. The non-myopic strategy produces the optimal solution for all costs of time.

Strategy / TC	.1	.2	.3	.4	.5	.6	.7	.8	.9
Greedy	2.5	1.9	1.4	.83	.22	-.17	-.57	-.80	-.90
Nonmyopic	2.5	1.9	1.4	.83	.30	-.17	-.57	-.80	-.90

Table 3.4. Performance of `Greedy` and `Nonmyopic` strategies for various time costs for the `Rock Sampling` domain. C_L was fixed at .04.

Strategy / C_L	.01	.02	.03	.04	.05	.06	.07	.08
Greedy	.33	.30	.28	.22	.22	.22	.22	.22
Nonmyopic	.37	.33	.31	.30	.28	.26	.24	.23

Table 3.5. Performance of `Greedy` and `Nonmyopic` strategies for various values of C_L on the `Rock Sampling` domain. Cost of time was fixed at $.5$

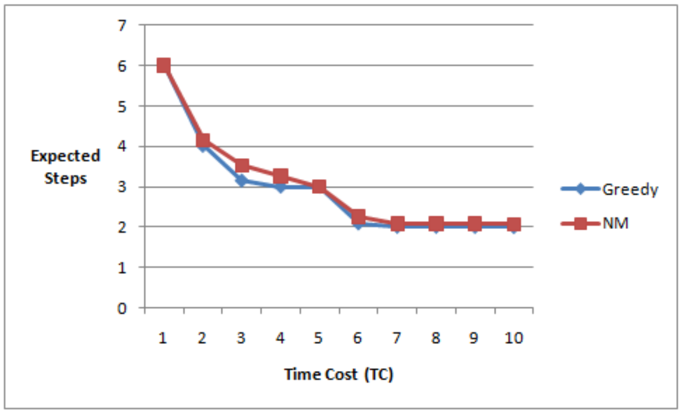


Figure 3.5. Mean number of steps produced until one agent terminates on the `Max Flow` problem. As cost of time increases, agents terminate sooner.

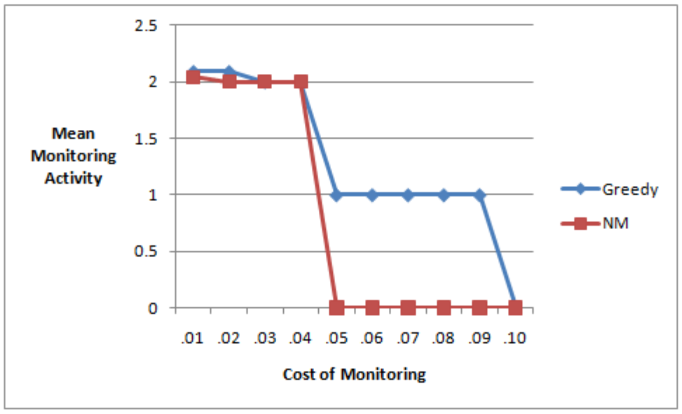


Figure 3.6. Number of monitoring actions for the `Max Flow` problem, versus varying values of C_L.

(a)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	6					
2						
3						
4						
5						
	Agent 2					
1	2M					
2						
3	0	1M	2	2M	1M	0
4	0	2	0	0	0	0
5	0	1	1	1	1	0

(b)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1M					
2	1	1	0	0	0	0
3						
4						
	Agent 2					
1	1M					
2	1M	0	1M	1M	1M	0
3	0	1M	2	2M	1M	0
4						

(c)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1					
2						
3						
4						
	Agent 2					
1	1					
2						
3						
4						

Table 3.6. Nonmyopic local monitoring policy for `Max Flow` problem, varying cost of time. (a) $C_L = .02\,TC = 0.2$. (b) $C_L = .02\,TC = 0.6$. (c) $C_L = .02\,TC = 1.0$. Integer entries represent number of time steps to continue, followed by a terminate action. Integer entries followed by M represent number of time steps to continue, followed by a monitoring action. Empty entries represent quality levels and times that will never occur in the policy.

(a)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	3M					
2						
3						
4	0	2	0	0	0	0
5						
	Agent 2					
1	3M					
2						
3						
4	0	2	0	0	0	0
5						

(b)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1M					
2	1	1	0	0	0	0
3						
4						
	Agent 2					
1	1M					
2	1	1	0	0	0	0
3						
4						

(c)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1					
2						
3						
4						
	Agent 2					
1	1					
2						
3						
4						

Table 3.7. Greedy local monitoring policy for `Max Flow` problem, varying cost of time. (a) $C_L = .02\ TC = 0.2$. (b) $C_L = .02\ TC = 0.6$. (c) $C_L = .02\ TC = 1.0$. Integer entries represent number of time steps to continue, followed by a terminate action. Integer entries followed by M represent number of time steps to continue, followed by a monitoring action. Empty entries represent quality levels and times that will never occur in the policy.

(a)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1M					
2	1	3M	0	0	0	0
3						
4						
5						
	Agent 2					
1	1M					
2	1M	0	1	1	1	1
3	1	0	0	0	0	0
4						
5						

(b)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1M					
2	1	1	0	0	0	0
3						
4						
	Agent 2					
1	2					
2						
3						
4						

(c)

	Agent 1					
	Agent 1					
t \ q_i	1	2	3	4	5	6
1	2					
2						
3						
4						
	Agent 2					
1	2					
2						
3						
4						

Table 3.8. Nonmyopic local monitoring policy for `Max Flow` problem, varying C_L. (a) $C_L = .01$ $TC = 0.6$. (b) $C_L = .03$ $TC = 0.6$. (c) $C_L = .05$ $TC = 0.6$. Integer entries represent number of time steps to continue, followed by a terminate action. Integer entries followed by M represent number of time steps to continue, followed by a monitoring action. Empty entries represent quality levels and times that will never occur in the policy.

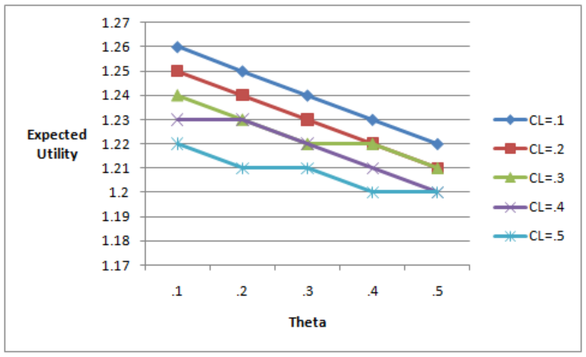
policies on Max Flow at varying costs of time. Table 3.7 shows their Greedy counterparts. In the tables, rows represent time steps and columns represent quality levels. Entries represent the number of steps to continue. If an entry is denoted in the form xM, a local monitoring action is performed after continuing for x steps. Otherwise, the agent terminates after continuing x steps. Blank entries represent quality states that will not be achieved according to the policies. As suggested earlier, when cost of time is low, agents tend to continue, and when it is high, agents tend to stop immediately. To present further detail, Figure 3.5 graphs cost of time versus expected agent stopping time, for execution of the Nonmyopic policy. As cost of time increases, agent policies are expected to use fewer time steps.

Similar to the above paragraph, which shows that a higher cost of time results in agents consuming less time, we also see that a higher cost of monitoring results in agents monitoring less frequently. Table 3.8 explores this relationship by varying cost of monitoring for a constant cost of time. As cost of monitoring increases, agents take fewer monitoring actions.

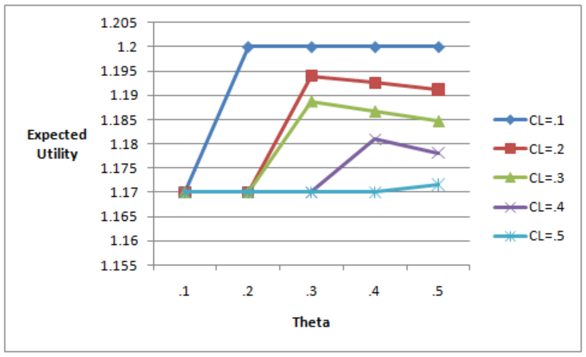
One item of note is that the nonmyopic policies are often asymmetric, one agent often continues while the other monitors and terminates. The greedy policies, by contrast, produce the same policies for both agents when the reward function is symmetric. Because greedy policies do not account for the decision-making abilities of the other agent, neither agent is capable of delegating the stopping decision to the other agent. This lack of coordination suggests that the Greedy strategy may also underperform as delegation becomes a larger issue, for instance with asymmetric utility functions.

To illustrate, we experimented on Max Flow utility functions

$$\theta q_1 + (1-\theta) q_2 - t \cdot TC,$$



(a) Nonmyopic



(b) Greedy

Figure 3.7. Effect of symmetry on performance with $TC = .6$. θ represents a symmetry parameter, when its value is $.5$ the problem is fully symmetrical. For nonmyopic policies, asymmetry increases performance, monitoring decisions can be delegated to the more knowledgeable agent. Greedy policies are not able to exploit this asymmetry as well and report lower values overall.

(a)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	3M					
2						
3						
4	0	0	1	1	1	1
5						
6						
	Agent 2					
1	1M					
2	3	2M	0	0	0	0
3						
4		1	1	1	1	0
5						
6						

(b)

	Agent 1					
t \ q_i	1	2	3	4	5	6
1	1					
2						
3						
4						
5						
6						
	Agent 2					
1	1M					
2	1	1	0	0	0	0
3						
4						
5						
6						

Table 3.9. Resulting policies for (a) `Nonmyopic` and (b) `Greedy` strategies with $C_L = .01$, $TC = .6$, and utility function $.1q_1 + .9q_2 - TC$. Integer entries represent number of time steps to continue, followed by a terminate action. Integer entries followed by M represent number of time steps to continue, followed by a monitoring action. Empty entries represent quality levels and times that will never occur in the policy.

for varying values of θ. Figure 3.7 shows the effect of an asymmetric utility function for $TC = .6$. Nonmyopic policies exploit asymmetry, whereas greedy policies are less able to. As a result, for $C_L = .01$, whereas nonmyopic performance increased, the greedy performance decreased with increasing asymmetry, i.e. $\theta \to 0$. The resulting policies are shown in Table 3.9. The first agent has little knowledge about global utility, and in the nonmyopic case, this agent largely cedes stopping decisions to the second agent. The first agent will stop never stop before step 4, and usually it will wait until step 5. The more knowledgeable agent may stop at various points in time. By contrast, in the greedy case, the first agent completely controls the stopping decision, preventing the second agent from contributing to the decision with its knowledge.

To summarize, monitoring policies generally outperform fixed stopping criteria. Furthermore, they even outperform the upper envelope of fixed stopping criteria, for monitoring costs which are not too high nor too low. Greedy policies often approximate the optimal nonmyopic policy well, with some exceptions. Exceptions are (1) In a narrow band of fixed stopping costs. (2) In situations calling for asymmetry in the policies.

3.5 Extensions

It is also possible to model monitoring with partial observability. Given certain independence assumptions, it should be possible to extend such models to many agents. For example, the ND-POMDP model is defined as follows [71].

- S = $\times_{1 \leq i \leq n} S_i \times S_u$ refers to the set of states. S_i refers to a local state of agent i, and S_u refers to a set of states that are uncontrollable by any agent. Also let $s_i \in S_i$ and $s_u \in S_u$.
- A = $\times_{1 \leq i \leq n} A_i$ refers to a set of actions. A_i refers to the set of actions available to agent i. Also let $a_i \in A_i$.

- $\Omega \times_{1 \le i \le n} \Omega_n$ is the joint observation set.
- $P(\vec{s'}|\vec{s}, \vec{a}) = P_u(s'_u|s_u)\Pi_{1 \le i \le n} P_i(s'_i|s_i, s_u, a_i)$ where $\vec{a}$ is the joint action performed in state $\vec{s}$ resulting in state $\vec{s'}$.
- $O(\vec{\omega}|\vec{s}, \vec{a}) = \Pi_{1 \le i \le n} O_i(\omega_i|s_i, s_u, a_i)$ is the observation function that assumes observation independence between agents.
- $R(\vec{s}, \vec{a}) = \Sigma_l R_l(s_l, s_u, a_l)$ is the reward function which is decomposable among subgroups of agents, each labeled l.
- b_0 represents the initial belief point.

A DMP can be reduced to a ND-POMDP with quality states mapping to ND-POMDP states, monitoring and stopping actions corresponding to the action space, and utility corresponding to the reward space. The reduction is similar to those already shown in this chapter. Using ND-POMDP, the DMP model can also be extended. In the following subsections we summarize some possible extensions.

3.5.1 Partial observability

One aspect modeled by ND-POMDP that is not modeled by the Dec-MDP is partial observability. That is, the ND-POMDP model can represent the fact that each agent may have some uncertainty in its estimate of its current local utility. In a representation of distributed monitoring via ND-POMDP, the observation set Ω contains the set of possible quality levels and the observation function O represents the chance that each quality level is observed given the true quality level. An example of a choice of O is a Gaussian distribution with a mean centered around the true quality level.

3.5.2 Independent stopping criteria

One aspect of the DMP model that was noted throughout the chapter was that in the model, one agent stopping causes all the agents to stop and use the current computation. Using ND-POMDP, it can be shown how this assumption can be relaxed. For each agent, define an additional state in the state space S_i

- s_f, a final state that results when agent i chooses a terminate action.

Then define:

$$R_l(s_l, s_u, a_i) = 0 \quad \text{iff} \quad s_f \in s_l$$

Using the modified reward function, each agent will contribute quality to each joint reward function R_l for which it is a contributing member. However, once the agent terminates, all the rewards to which the agent contributes are zero. The other agents may continue, and still add to utility as long as such utility does not depend on the terminating agent.

3.5.3 Varying cost of time

It is also possible to vary the cost of time using an ND-POMDP model. Let c_t be a variable representing cost of time. This cost of time can be expressed in the members of set S_u. Transitions in S_u are determined according to the dynamics of the distribution of the cost of time. Reward functions depend on the cost of time as shown in the expression below.

- $R_l(s_l, (s_u = c_t), a_i)$

A transition function for the cost of time would be defined as well, i.e. $P(c_t'|c_t, t)$. This transition function is independent of the transition function of the quality levels of the individual agents, q_i.

3.6 Summary

Anytime algorithms effectively gauge the trade-off between time and quality, and we saw in Chapter 2 that conditional performance profiles formalize this notion. Monitoring these algorithms can be performed in order to aid in the control of base-level computation. Existing techniques from the literature weigh the trade-off between time, quality, and monitoring for the single-agent case. Dynamic programming methods provide a solution method for the single-agent case.

This chapter addressed the multi-agent case. It analyzed several cases involving the execution of multiple base-level anytime algorithms. When there is only one decision-maker of stopping time, the single-agent approaches can be extended by defining expectations over the computational state of the other agents. Computational experiments demonstrated that in some cases this greedy method is close to optimal for multiple decision makers as well, but under certain circumstances where the trade-offs are in balance, the greedy method breaks down and other optimal solution methods should be used.

There were several contributions of this chapter. First, the problem of monitoring and anytime control was formalized. Second, its complexity was analyzed. This chapter showed how the multi-agent monitoring problems can be compiled as special cases of Decentralized Markov Decision Processes. Third, solutions were developed. This chapter showed that Dec-MDP solvers from the literature can produce efficient solutions. Fourth, the behavior of the solution was analyzed empirically. Monitoring policies outperformed fixed stopping criteria. Greedy policies were found to approximate optimal policies in some cases, but were shown to underperform within a narrow band of fixed stopping costs and in situations requiring policy asymmetry.

Although we chose to define and analyze the DMP model for simple, clean cases where all agents stop simultaneously, the concepts in this chapter are easily extended to models where agents do not stop simultaneously. The bilinear formulation holds both local and global rewards, and in our implementation we make use of only the global reward. When one agent terminates, it forces global reward to be zero thereafter, thus enforcing a simultaneous stopping rule. However, if local rewards were also assigned, agents would not need to stop simultaneously. Instead, one agent stopping would stop global reward, but other agents could continue to achieve local reward. To conclude the section, we discussed a model that includes this extension as well as several others including many agents and a variable cost of time.

CHAPTER 4

OBSERVATION COMPRESSION IN MULTIAGENT SETTINGS

The previous chapter analyzed the trade-off of time for quality in a distributed environment. This chapter analyzes the trade-off of space for quality. Specifically, this chapter examines planning for the finite-horizon Decentralized Partially Observable Markov Decision Process (Dec-POMDP) model, which was introduced and defined in Chapter 2.2.

The architecture of the model is shown in Figure 4.1. Execution is distributed among multiple agents without communication. At execution time, each agent acts independently, the environment transitions in state based on the joint action, and then each agent receives an observation and chooses its next action. Action selection is based on agent policy, which is developed at planning time. Planning is centralized and occurs prior to execution. This chapter develops a meta-reasoning process for the centralized planning of Dec-POMDP policies.

Specifically, the problem addressed by this chapter is the doubly-exponential growth of Dec-POMDP policy trees with respect to time horizon. Recall Figure 2.5. At each time step, a Dec-POMDP policy must specify the subpolicy to be followed after each observation. If there are $|A|$ possible actions, $|\Pi|$ possible subpolicies, and $|\Omega|$ possible observations on a given step, the number of possible policies at the next step is $|A||\Pi|^{|\Omega|}$. If each of these policies is retained as a possible sub-policy for each step, this makes Dec-POMDP policies doubly-exponential in size, (i.e. after two steps the number of possible policies is $|\Pi|^{|\Omega|^{|\Omega|}}$, and so on). The *MBDP* algorithm prunes the number

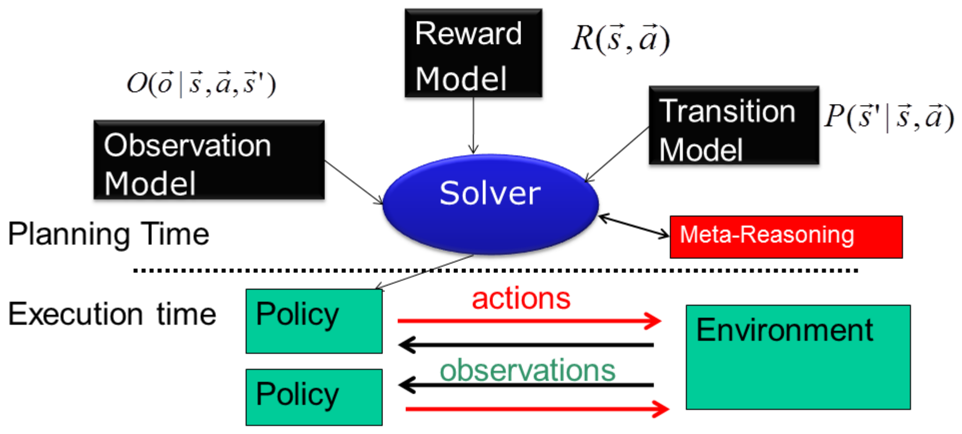


Figure 4.1. Meta-reasoning for a Dec-POMDP.

of policies at each step to a constant, thus reducing complexity to singly-exponential. However *MBDP* still produces an exponential number of policies ($|A||MaxTrees|^{|\Omega|}$), on each step before pruning, this term is large when the number of observations $|\Omega|$ is large. This chapter develops an algorithm that generates a bounded number of policies on each step. It introduces a user-selected parameter $MaxObs$ and reduces the number of policies generated to $|A||MaxTrees|^{|MaxObs|}$.

In order to reduce the space of policies and policy planning, this chapter introduces the *observation compression* (OC) technique for Dec-POMDPs. The observation compression method is loosely inspired by human reasoning processes, which plans for the most important set of contingencies. The most important set of contingencies may not always be the most likely set, as the subplan under a rare observation may have a large impact on utility. Observation compression categorizes observations into groups, and finds the best subpolicy for each category.

The contributions of this chapter are as follows.

- *Compact representation of policy trees.* This chapter introduces a policy tree representation with bounded size. In the first section of this chapter, we will formalize this notion to develop a means of making policies more compact, with limited loss of value from the compression.

- *Augmented* MBDP *planning algorithm.* This chapter introduces an augmented *MBDP* algorithm which uses the compact representation. The observation compression is represented in its own software module, so it can be combined with other planning algorithms as well.

The work presented in this chapter was first published AAMAS 2008 [26]. Since publication, a variety of newer planners have appeared in the literature. The *PBIP* (Point Based Incremental Pruning) planner is also based on *MBDP* , like *MBDP-OC* it does not perform a full backup [42]. As opposed to *MBDP-OC* , *PBIP* performs a heuristic search of joint trees at each step, and thus does not perform a backup at all. It would be possible to combine the heuristic search of *PBIP* over the compressed observation sets developed by *CompressObs* in this chapter. The *PBPG* (Point-Based Policy Generation) algorithm is also based on MBDP [152]. Rather than performing a full backup at each step, *PBPG* solves a linear program for each belief point, for each agent alternately while fixing the policies of the other agents, and stopping when the policy value improvement converges. It is possible to perform the observation compression developed in this chapter before running the linear program, thus running the linear program on a smaller set of observations. This combination would be most useful for problems with a very large number of observations.

Other more recent approaches make use of similar or extended concepts to the observation compression mentioned in this chapter. Incremental policy generation combines the *PBIP* algorithm with a state-pruning measure, which eliminates unreachable states from evaluation as backup is performed [5]. The *GMAA* *-Cluster algorithm performs clustering on action-observation histories rather than just observations as in this chapter. It is used in the context of a top-down heuristic search over joint policies. Nodes representing the continuation of these joint policies are merged together when such a merge can be accomplished losslessly (i.e. when the separate action-observation histories lead to the same belief point) [89]. The most recent successor algorithm to *GMAA* * as of the time of publication of this thesis, called *GMAA* *-ICE, makes use of this underlying lossless clustering technique.

In the next section, we will briefly review a similar method in the literature developed concurrently, the *IMBDP* planner, which plans for the most likely observations rather than the most valuable. Next, we will introduce the observation compression method for both single and multi-agent settings. We will produce an observation compression algorithm, and analyze its theoretical properties. Finally, we will evaluate the observation compression method empirically, as compared to the *IMBDP* method of limiting observations.

4.1 Related work: IMBDP

The *IMBDP* planner limits the number of possible observations to a constant parameter $MaxObs$ [119]. *IMBDP* selects the most likely observations from heuristically determined belief states, and only backs up policy trees with those observations. This process is termed a *partial backup*. Complete policies are then "filled up" with the missing observations, by selecting the best available policies for these observations. Thus, for instance if a partial backup were run for the policies shown in part A of Figure 2.5, and a $MaxObs$ of two were used, only 12 new policies would be generated, 4 for each root action. Although this process is quick compared to a full backup, the error is only bounded for the heuristically selected belief state. The error is given by:

$$T^2(1-\epsilon)(R_{max}-R_{min})$$

where T is the horizon of the problem, ϵ is the probability of receiving one of the remaining observations, and R is the reward function. In *IMBDP* , the probabilities of these observations are taken into account, but not their values.

The defining feature of the *IMBDP* algorithm is that it only considers the most probable observations. However, it may be the case that many subpolicies have the same value under the most probable observation branches, whereas the less probable observations may show great loss in value when the wrong policy is selected. (For example, consider a robot that spends much of its time waiting to respond to an infrequent cue). Therefore bounds on loss due to compression are loose in *IMBDP* .

4.2 Observation compression

This section builds an algorithm to reduce the complexity of the backup process by merging pairs of observations in each iteration. The algorithm will be called *CompressObs* . This observation compression method can be used as a tool that can be used in bottom-up planners to produce compact policies. The *MBDP* planner is an example of a planner that can be augmented by using this algorithm. The *MBDP* algorithm augmented by observation compression will be referred to as *MBDP* -OC.

During the backup process, *MBDP* -OC operates in a similar manner to *IMBDP* in seeking to limit the number of observations and policies considered for backup [120]. Recall that each branch of a new policy tree corresponds to an observation. In contrast to *IMBDP* , the partial backup in *MBDP-OC* seeks to reduce the exponential generation of new policies by forcing different branches of the new root policies to contain the same subtrees. The distinction can be seen in Figure 2.5(c). *IMBDP* (not shown in the figure) selects the $MaxObs$ most likely observations, finds all combinations of subpolicies for those, and fills in all remaining observations with a single policy. *MBDP-OC* , by contrast, finds $MaxObs$ groups of policies, and in the backup process forces all observations in the same group to have the same subpolicies. Thus, the key difference with *IMBDP* is just before the partial backup. In *MBDP-OC* , the procedure *CompressObs* is called to determine how to group the observations. The groupings are stored in a set Z, where Z is a set of tuples (Z_i, ϵ_i). Each Z_i is a set of observations, and each ϵ_i is an error term associated with observation group Z_i. We call the backup procedure that performs these calculations $PartBackup$ to distinguish it from the partial backup in *IMBDP* .

Let $MaxTrees$ be the number of distinct subtrees that the user desires and $MaxObs$ be the limit on the number of observations. We then have $MaxTrees$ policies before backup and $|A|$ actions, so we generate $|A|MaxTrees^{MaxObs}$ new policy trees for each agent at backup. The implementation then follows [120] by selecting the $MaxTrees$ best policies to retain for the next backup.

4.2.1 Single-agent definitions

We formalize by first presenting definitions for the single agent case, and then expanding these definitions to the multiple agent case. For the single agent case, define:

- π_i^1, π_i^2..... The policy trees for agent i. During the backup process, these policy trees will become subpolicy subtrees. Throughout this chapter we will often refer to a generic single subpolicy for agent i as π_i.
- $\{\pi_i^1, \pi_i^2...\} = \Pi_i$. In the context of *MBDP* , the set Π_i has $MaxTrees$ members.

Algorithm 3: CompressObs($i, b, a_i, A_{-i}, \Omega_i, \Omega_{-i}, \Pi_i, \Pi_j, MaxObs$)

```
input  : Agent number $i$, belief state $b$, root action $a_i$, joint action set $A_{-i}$, sets of
         subpolicies $\Pi_i$ and $\Pi_{-i}$, sets of observations $\Omega_i$ and $\Omega_{-i}$, and observation
         limit $MaxObs$
output: $Z$, a set of tuples $(Z_k, \epsilon_k)$, where each $Z_k$ is a subset of observations
         available to agent $i$, and $\epsilon_k$ is the error introduced by constructing
         observation set $Z_k$
begin
    $Z \leftarrow \{(\{o_1\}, 0), (\{o_2\}, 0), ...\}$
    for each observation $o_i \in \Omega_i$, each policy $\pi_i \in \Pi_i$, and each $\pi_{-i} \in \Pi_{-i}$ and
    $a_{-i} \in A_{-i}$ do
        Precompute $V(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i})$
        Keep track of best policy $\pi_i^*$
    while $|Z| > MaxObs$ do
        $(Z_x, Z_y.\pi_Z) \leftarrow argmin_{(Z_x,\epsilon_x)(Z_y,\epsilon_y)\in Z,\pi_i} WL^*(\pi_i|b, a_i, Z_x \cup Z_y)$
        $\epsilon \leftarrow WL^*(\pi_Z|b, a, Z_x \cup Z_y)$
        $Z \leftarrow Z\backslash\{Z_x, Z_y\} \cup (Z_x \cup Z_y, \epsilon)$
    return $Z$
end
```

- $V(\pi_i|b, a_i, o_i)$. The value of following subpolicy π_i after taking action a_i and receiving observation o_i in belief state b.
- $\pi_i^*(b, a_i, o_i) = argmax_{\pi_i \in \Pi_i} V(\pi_i|b, a_i, o_i)$. The best subpolicy available, after taking action a_i and receiving observation o_i.
- $L(\pi_i|b, a_i, o_i) = V(\pi_i^*|b, a_i, o_i) - V(\pi_i|b, a_i, o_i)$. The value lost if subpolicy π_i is taken instead of the best available subpolicy.
- $WL(\pi_i|b, a_i, Z_x) = \sum_{o_i \in Z_x} O(o_i|b, a_i) L(\pi_i|b, a_i, o_i)$. Extends the loss term above to (weighted) sets of observations Z_x. The loss of following a single policy π_i for the set of (merged) policies in $Z_x \subset \Omega$. Each loss is weighted by the probability of the observation.
- $WL^*(\Pi_i|b, a, Z_x) = \min_{\pi_i \in \Pi_i} WL(\pi_i|b, a_i, Z_x)$. The Weighted Loss of following the best available single policy for the group of policies in Z_x.

The notation above always explicitly specifies (b, a_i, o_i). For a standard POMDP, one could have defined a new belief state b' given that action a_i was taken and o_i was observed from belief state b, but this notation would not scale to the multiagent case. None of these terms consider the immediate reward achieved through action a_i in belief state b. This immediate reward is irrelevant to the compression process, which only affects the value of subpolicies taken after the observation is received. Thus, the terms above compute the probabilities of new belief states once action a_i is taken and o_i is observed, and they find subpolicy values in the new belief state.

We use standard POMDP methods to evaluate $V(\pi_i|b, a_i, o_i)$. Since the agent started at belief state b, took an action a_i and received an observation o_i, this defines a new belief state, which we will call $b(s')$. One can then evaluate π_i :

$$\sum_{s'} b(s')V(\pi_i, s')$$

the probability of being in each state times the value of π_i for that state. The value of π_i for state s' is recursively computed.

$$R(s', a'_i) + \sum_{s'' \in S} P(s''|s', a'_i) \sum_{o'_i \in \Omega} O(o'_i|s'', a_i)V(\pi'_i, s'')$$

where a'_i is the root action in the policy, and $V(\pi'_i, s'')$ is the value of continuing the policy in state s'.

We use $\pi^*_i(b, a_i, o_i)$ to denote the subpolicy for agent i with the highest value, given that action a_i was taken from belief state b and then o_i was observed.

In the definition of L , the difference

$$L(\pi_i|b, a_i, o) = V(\pi^*_i|b, a_i, o_i) - V(\pi_i|b, a_i, o_i)$$

is the value lost by choosing a single policy π_i (instead of the ideal π_i^*) after receiving the observation o_i. One can then weight the probability of observing o_i. The result is a weighted loss, which is abbreviated WL. This weighted loss can be used to denote the sum of the values lost by choosing one single policy for a whole group of observations, rather than the best policy for each observation.

The value of the best single policy for the observations in a set $Z_x \subset \Omega$ is WL^*. This is the cumulative value lost if we are to group all the observations in Z_x together.

4.2.2 Multi-agent definitions

For the multiagent case, the actions become joint actions, and policies become joint policies. The following makes use of the convention $-i$ to represent the other agents. When there are more than two agents, the number of joint actions, joint observations, and joint subpolicies may be exponential in the number of agents.

- $V(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i})$ is the value of following joint policy (π_i, π_{-i}) after taking joint action (a_i, a_{-i}) and after the agents receive observations o_i and o_{-i} in belief state b.
- $\pi_i^*(\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i}) = argmax_{\pi_i \in \Pi_i} V(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i})$. When the arguments to π_i^* are clear from context, as below, we may just refer to it as π_i^*.
- $L_i(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i}) = V(\pi_i^*|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i}) - V(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i})$
- $WL_i(\pi_i|b, a_i, Z_x) = \sum_{o_i \in Z_x} \max_{a_{-i} \in A_{-i}} \sum_{o_{-i} \in \Omega_{-i}} \max_{\pi_{-i} \in \Pi_{-i}}$ $O(o_i, o_{-i}|b, a_i, a_{-i}) L_i(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i})$
- $WL_i^*(\Pi_i|b, a_i, Z_x) = \min_{\pi_i \in \Pi_i} WL_i(\pi_i|b, a_i, Z_x)$

As seen above, we must consider all possible observations, actions, and policies of the other agents. The loss term is modified to be a loss for a fixed action, observation, and subpolicy of the other agent. Similarly, weighted loss terms are modified to sum over the observation probabilities of the other agent, and to find the worst-case actions and subpolicies that the other agent may take based on these observations.

The weighted loss definition means that our algorithm will consider all other policies of the other agent when deciding which observations to merge. This issue is discussed further in the conclusion to this chapter.

The observation compression process itself is displayed in Algorithm 3. As the figure shows, the process can be packaged into a function call, and thus can be seamlessly integrated with *IMBDP* , or indeed any policy tree based algorithm. The function seeks to consider which observations we can merge with a minimum loss of value. Its parameters include the root action for some new set of policies on agent i. The algorithm precomputes the value of each of the existing $MaxTrees$ subpolicies, for each observation and possible policy of the other agents. Possible policies of the other agents include the set of $|A_{-i}|$ root actions as well as the set of existing subpolicies from the previous step. After the value for each subpolicy has been determined, the algorithm identifies the best subpolicy according to its value. This can be used to construct a table of losses (not explicitly shown in the algorithm, refer to the discussion above), which identifies the value lost by each subpolicy if the agent should follow that one instead of the best, given the observation and the policy of the other.

The algorithm weighs this loss of value by the probability of receiving the observation. For each of the $MaxTrees$ subpolicies of the current agent, it finds the weighted loss (WL) introduced by using that one subpolicy for the merging of two groups of observations, which is the sum of the loss for each observation in the group, for all possible policies of the other agent. Thus the $WL_i^*(\pi_i|b, a_{-i}, Z_x)$ of the set of observations Z_x is the summation of the weighted losses for all members of the combined set while using the best possible single policy. The algorithm finds WL^* for each pair of groups in Z, and merges the two groups that introduce the smallest amount of loss.

4.2.3 Observation-compression for meta-reasoning

The compression algorithm iteratively compresses groups of observations. It starts with $|\Omega_i|$ observations, and selects two groups of observations to compress together while minimizing the weighted loss. In the first iteration, each observation in Ω_i is its own group. In successive iterations, it continues to merge groups of observations, until there are only $MaxObs$ observations left. Note that since the algorithm computes a loss bound ϵ for each iteration, it can be easily modified to iterate until a certain loss threshold is achieved.

4.3 Meta-reasoning in point-based decentralized planning

Algorithm 4 shows how *CompressObs* may be called within *MBDP* for the two agent case (the multi-agent case is similar). Each horizon step begins as in *MBDP* , sets of policies stored in Π_i are reduced to size $MaxTrees$ by saving only the best joint policies in heuristically selected belief points. After this pruning, the *CompressObs* algorithm is run. The new sets Π_i are constructed using the compressed observation set Z, rather than by running a full backup as in *MBDP* .

One further point is that it is possible to compress observations until a loss bound is exceeded, rather than until there are $MaxObs$ sets left. In *CompressObs* , this line

$$\text{while}|Z| > MaxObs$$

would be replaced by this line:

$$\text{while (loss} < \text{threshold)}$$

with “loss” being accumulated from the ϵ values found just below. *CompressObs* keeps an online error bound, as will be shown in the next section.

Algorithm 4: The *MBDP-OC* algorithm

```
begin
    $MaxTrees \leftarrow$ max number of trees before backup;
    $H \leftarrow$ pre-compute heuristic policies for each $h \in H$;
    $T$ horizon of the Dec-POMDP;
    $\Pi_i^1, \Pi_j^1 \leftarrow$ initialize all 1-step policy trees;
    for $t = 1$ to $T$ do
        $Sel_i^{t+1}, Sel_j^{t+1} \leftarrow empty$;
        for $k = 1$ to $maxTrees$ do
            choose $h \in H$ and generate belief state $b$;
            for each $\pi_i \in \Pi_i^{t+1}$, $\pi_j \in \Pi_j^{t+1}$ do
                evaluate each pair $(\pi_i, \pi_j)$ with respect to $b$;
            add best policy trees to $Sel_i^{t+1}$ and $Sel_j^{t+1}$;
            delete those policy trees from $\Pi_i^{t+1}$ and $\Pi_j^{t+1}$;
        for each agent $i$ do
            $Z \leftarrow CompressObs(i, A_i, A_{-i}, \Omega_i, \Omega_{-i}, Sel_i^{t+1}, Sel_{-i}^{t+1}, MaxObs)$;
            $\Pi_i^{t+1} = backup(Sel_i^{t+1}, Z)$;
    select best joint policy tree $\delta^T$ from $\{\Pi_i^T, \Pi_j^T\}$;
    return $\delta^T$;
end
```

4.4 Complexity analysis

This section discusses the running time and space used by *CompressObs* , as well as its error bounds. The analysis in this section is limited to the *CompressObs* algorithm itself. Although in this chapter we focus on the example of running *CompressObs* with the *MBDP* planner, it is possible to run *CompressObs* with an optimal planner or an epsilon-optimal planner [4, 54]. In the context of an optimal planner, the global error is bounded by the error of *CompressObs* itself. When embedded in an *MBDP* planner, however, the overall error will also include error produced by the *MBDP* planner itself in its selection of belief points.

This section will consider both special cases and the general case when *CompressObs* is called. In certain special cases, the error introduced by *CompressObs* is low. In the general case, a loose online error bound is constructed that depends on the worst possible choice of the other agent. The online error bound may be optionally used to construct algorithms that terminate after its error reaches a predetermined threshold. The first theorem in this section discusses running time and space for *CompressObs* .

Theorem 4.1. *For the two agent case, the running time of* CompressObs *is*

$$O(\mathit{MaxTrees}^2|A||\Omega|^4)$$

, and the space used is $O(\mathit{MaxTrees}^2|A||\Omega^3|)$.

Proof. We examine each component of the algorithm. First, all subpolicy values are computed and stored. This occurs for each subpolicy of the current agent, for each subpolicy of the other, for each initial action of the other, and for each observation vector. This takes $O(\mathit{MaxTrees}^2|A||\Omega^3|)$ time and space. Identifying the best policies is just a matter of scanning this list. Loss terms can be precomputed in a similar operation, and again requires $O(\mathit{MaxTrees}^2|A||\Omega^3|)$ time and space. The algorithm then enters a while loop where it iteratively shrinks $|Z|$ from $|\Omega|$ down to MaxObs. There are $(|\Omega| - \mathit{MaxObs})$ iterations of this. Within the while loop, weighted losses are found and stored for each pair of groups of observations in Z (that is, for each possible (Z_x, Z_y), requiring $O(|\Omega|^2)$ storage). Each merged group has at most $|\Omega|$ observations, and there are $|Z|^2$ possible pairs, and we know $|Z| < |\Omega|$. Thus there is an order of $O(|\Omega|^4)$ computations of WL. We only keep track of the identities of the best groups to merge, so no additional storage is required. The computation of WL itself looks up joint policy values previously stored in *MBDP* . Since WL is referenced for each possible action and subpolicy of the other agent, and since this occurs in a while loop, the total time spent finding WL function values, including the while loop, is $O(\mathit{MaxTrees}^2|A||\Omega|^4)$. □

This analysis is limited to *CompressObs* as written in Algorithm 3, which minimizes loss for the worst case policy of the other agent. *CompressObs* computes online error bounds by considering the other agent as a limited adversary, so it computes a strategy for such an adversary for each observation. In practice, it may not be necessary to compute around the worst case other agent, and indeed an expectation over the other agent's policy may be more accurate than bounding its worst-case.

The following Lemma states that if *CompressObs* iteratively merges an observation into larger and larger sets, one can find the total error introduced for this observation by examining the weighted loss term of last merge. That is, the error does not accumulate.

Lemma 4.1. *Suppose that the sets of observations Z_x and Z_y are merged by the above algorithm. The total value lost due to the merging of Z_x and Z_y only depends on the components of Z_x and Z_y, not on the value lost during the formation of Z_x and Z_y.*

Proof. The base case is trivial as the observations have not been previously merged. Note that the value lost due to the observation compression process is the difference between value of the best available policy and the value of the best possible policy once the observations are merged. This notion is captured in the definition of the loss term:

$$L = V(\pi_i^*|\pi_j, b, a_i, a_j, o_i, o_{-i}) - V(\pi_i|\pi_j, b, a_i, a_j, o_i, o_{-i})$$

For the inductive case, we find a best policy π_i that minimizes the weighted sum of the loss of all observations o_i where $o_i \in Z_x \cup Z_y$. As the above value difference equation indicates, the value for each observation only depends on the value of the best possible policy and the value of the selected policy, and not on previous merges. Therefore the lemma holds. □

Next, a special case is considered where $MaxObs$ is greater than or equal to the number of $MaxTrees$. Since ***MBDP*** is capable of running with $MaxTrees = 3$, this setting is certainly possible and sometimes desirable.

Proposition 4.1. *If $WL_i(\pi_i|b, a_i, Z) = 0$, then π_i is a best available subpolicy for all of the observations in $\bar{o}$.*

Proof. The definition of $WL_i(\pi_i|b, a_i, Z) = 0$ consists of summations and maximums of terms of form

$$O(o_i|b, a_i, a_{-i})L_i(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i}) = 0.$$

The loss L_i is always non-negative, since by definition of π_i^* the value of any policy given an observation cannot exceed π_i^*. Since the observation probability is always non-negative as well, and the sum of all the terms must be zero, then each individual term must be zero for the theorem pre-conditions to hold. Thus we do not need to consider the nonzero observation probabilities. Decomposing L_i, we are left with the sum of several terms of the form:

$$V(\pi_i^*|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i}) - V(\pi_i|\pi_{-i}, b, a_i, a_{-i}, o_i, o_{-i})$$

where π_i is the selected policy that makes the theorem pre-conditions hold, and o_i is the observation in that term. Thus for the pre-conditions to hold, π_i must equal π_i^* in value. Since π_i has the same value as the best available subpolicy for all the observations in belief state b, for all actions of the other, then π_i is a best available subpolicy for policies rooted in a_i, for all of the observations in Z while in belief state b. □

The next proposition also considers a special case. For the single-agent case, when $MaxObs$ is greater than $MaxTrees$, running observation compression does not lose any value in an *MBDP* environment.

Proposition 4.2. *When the* MBDP-OC *algorithm is applied to a POMDP problem, if $MaxObs \geq MaxTrees$, then* MBDP-OC *constructs the same best available policy tree as* MBDP .

Proof. The algorithm can be run on a POMDP by considering the problem to be a Dec-POMDP where other agents are restricted to one policy, action, and observation. ***MBDP*** chooses a belief state b and generates the best policy tree for that belief state. Proving this proposition equates to proving that with ***MBDP-OC*** , the resulting value lost in the observation compression process is zero for belief state b.

Note that a policy consists of a root action, and each subtree corresponds to an observation. There are $|\Omega|$ subtrees. However, since only $MaxTrees$ policies have been saved from the previous step, only $MaxTrees$ of these subtrees are unique in the policy selected by ***MBDP*** .This means $(|\Omega| - MaxTrees)$ of the subtrees are a duplicate of some other subtree. Take a subtree under policy branch o, and suppose it is a duplicate of the subtree under policy branch o'. Clearly $\pi_i^*(b, a, \{o\})$ and $\pi_i^*(b, a, \{o'\})$ are the same policy.

Thus we choose this policy for π_i in computing $WL^*(\Pi_i|b, a_i, (i\{o\} \cup \{o'\}))$, for any choice of a_i. Since $L(\pi_i^*|b, a, o) = L(\pi_i^*|b, a, o') = 0$ and weighted loss cannot be less than zero, the ***MBDP-OC*** algorithm selects o and o' (or some other pair of observations whose weighted loss is also zero) for compression and selects π_i^* as the subpolicy of the compressed branch, thereby generating the same policy as ***MBDP*** . □

Let $V_{MBDP}(b, a_i)$ represent the expected reward of the best joint policy for belief state b after ***MBDP*** performs a full backup and produces policy trees rooted in a_i for agent i. Likewise, let $V_{MBDP-OC}(b, a_i)$ be the expected value of the best joint policy for belief state b and trees rooted in a_i after ***MBDP-OC*** performs a partial backup.

Next, the general class of Dec-POMDP problems is considered. The following theorem and corollary shows bounds on the loss of value due to observation compression when observation compression is used as a sub-module in ***MBDP*** .

Theorem 4.2. *Let*

$$error(b, a_i) = V_{\text{MBDP}}(b, a_i) - V_{\text{MBDP-OC}}(b, a_i).$$

Then there exists a corresponding policy tree produced by MBDP-OC*, rooted in a_i, with sets of observations $Z = Z_1, Z_2...$ produced by the* CompressObs *algorithm, containing sub-policies $\pi_i^1..\pi_i^{MaxObs}$ respectively, such that*

$$error(b, a_i) \leq \sum_{k=1}^{MaxObs} WL(\pi_i^k | b, a_i, Z_k)$$

Proof. Each π_i^k is the subpolicy that *MBDP-OC* assigns to the observation branch Z_k in order to minimize the error. The partial backup in *MBDP-OC* produces all combinations of assignments of subpolicies for $Z_1, ...Z_{MaxObs}$, thus assuring that $\pi_i^1..\pi_i^{MaxObs}$ must exist.

For the error term, perform induction on the number of policies and actions available to the other agents.

- **Base Case:** when the other agents have just one policy and action available.

  Lemma 4.1 shows that the error introduced when *CompressObs* creates each Z_k is the error introduced when it performed the last merge that created Z_k. For each $o_i \in Z_k$, fixing the policy of the other agent makes the error introduced by choosing π_i^k the difference between the best policy it could choose versus the policy it does choose for each z_k,

$$V(\pi_i^* | (\pi_j, b, a_i, a_{-i}, o_i, o_{-i})) - V(\pi_i^k | (\pi_j, b, a_i, a_{-i}, o_i, o_{-i})).$$

  The contribution of each o_i to the total error is weighted by its probability, and thus the contribution of π_i^k to total error is

$$WL(\pi_i^k | b, a_i, Z_k),$$

  and since no observation is in two different sets, the total error, $error(b, a_i)$, is the sum of the contributions of the components of Z.

- **Inductive Case:** Assume that the theorem holds when the other agents' policies are limited to $|A| - 1$ root actions. Adding another root action choice for another agent means that are $|A|$ more possible joint actions at the root (since a_i is fixed), and thus $|A|$ more belief states must be considered when analyzing the value of joint subpolicies. The possible loss of value from *MBDP* must be analyzed separately for these belief states. If $WL(\pi_i^k|b, a_i, Z)$ is larger for any of these belief states than for belief states specified in the inductive hypothesis, then it is the new error bound and the theorem holds. If not, then by the inductive hypothesis the theorem holds.

  To summarize, adding a subpolicy π_{-i}^k to the other agent means $WL(\pi_i^k|b, a_i, Z)$ may or may not be larger than the weighted losses of existing subpolicies. If it is larger, it contributes to the maximum weighted loss and the theorem holds. If not, the inductive hypothesis says that the theorem holds.

□

Corollary 4.1. *Given a belief state b, in the worst case,* MBDP-OC *loses* $\max_{a_i \in A}\{error(b, a_i)\}$ *per iteration in comparison to* MBDP.

Proof. The previous theorem bounds the loss by *MBDP-OC* in comparison to MBDP for policy trees rooted in a single action. There is a best joint policy produced by *MBDP-OC* , and that joint policy is rooted in an action a_i, and the joint expected reward lost by *MBDP-OC* versus MBDP for that action is $error(b, Z_{a_i})$. □

4.5 Experiments

This section compares the *CompressObs* algorithm used in *MBDP-OC* against the partial backup of *IMBDP* , the previous state-of-the-art algorithm for compressing policy trees, on benchmark problems.

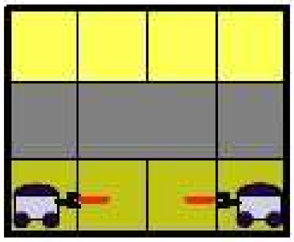


Figure 4.2. Box Pushing Domain.

The problem selected was the Cooperative Box Pushing domain [70], presented in Figure 4.2. In this domain, agents are required to push boxes into a goal area. The variant we used involves 2 agents, each of which can be located on 4 squares of the bottom row of a 3x4 matrix, and each agent can have 4 possible orientations (facing up, down, left or right). There are walls below and to the left of the matrix. In front of the leftmost and rightmost location are 2 small boxes. A large, 2-square box is in front of the middle two locations. The agents have 4 available actions, turn left, turn right, move forward, or stay. Each action has a .9 probability of success. They can receive 5 possible observations of what is in front of them: empty, wall, other agent, small box, or large box. If an agent moves forward, and a small box is in front of it, the box will be pushed into the top row. If both agents push the large box at the same time, it is pushed into the top row. A reward of 10 is received for pushing a small box into the top row, and a cooperative reward of 100 (50 per agent) is received when both agents push the large box into the top row at the same time. A penalty of 5 is received for bumping into a wall or trying to push the large box alone, and a penalty of 6 is received each time it bumps into the other agent.

Each time a box is pushed forward, the goal state is entered, and the problem resets. The initial state of the problem is with the agents at coordinates (3,1) and (3,4), facing upwards. In order to make the problem more challenging, and to enhance the role of

	$MaxObs = 2$		$MaxObs = 3$	
horizon	IMBDP	MBDP-OC	IMBDP	MBDP-OC
1	-.2	-.2	-.2	-.2
5	51.6	69.1	79.1	72.3
10	71.3	88.6	90.9	103.9
20	55.2	127.2	96.0	149.8
50	47.8	221.7	80.8	278.7
100	38.2	350.4	72.8	503.8

Table 4.1. Comparison of IMBDP and MBDP-OC on the Repeated Box Pushing Domain with various horizons. The algorithms were run with $MaxTrees = 3$. Results are shown for $MaxObs = 2$ and $MaxObs = 3$.

observations, we would have the agents transition to a random state when the problem resets itself. Thus, for instance, agent 1 could find itself facing the back wall and need to turn around. There are 96 reachable non-goal states since the domain forces agent 1 to be left of agent 2. There are 4 possible goal states which reset the problem.

Experiments were run with parameters $MaxTrees = 3$ and $MaxObs = 2$ as well as $MaxObs = 3$. Results are displayed in 4.1. *IMBDP* was run with the same parameters for comparison. Results show that the value function computed by MBDP-OC produces improved policies, for both $MaxObs = 2$ as well as $MaxObs = 3$. Runtimes are shown in Table 2. There is a small time penalty for running *MBDP-OC* . However, the program still spends the majority of its time in the classic MBDP portion of the algorithm, where it must evaluate all possible combinations of generated joint policies in all possible states, in order to pick out the $MaxTrees$ policies to retain for the next step.

The second domain chosen was Meeting in a Grid. We ran on a 3×3 instance of the Meeting in a Grid problem introduced by Bernstein et al. [15]. In the selected implementation, agents start in opposite corners of a 3×3 grid. Each agent can move either up, down, left, or right. The chance of a successful action was 60%, with a 15% chance of moving in each perpendicular direction, a 5% chance of not moving

Algorithm	MaxObs	Time (s)
IMBDP	2	29.9
MBDP-OC	2	31.7
IMBDP	3	459.0
MBDP-OC	3	469.9

Table 4.2. Running time per horizon step for the IMBDP and MBDP-OC algorithms on the Box Pushing problem. Results are in seconds.

at all, and a 5% chance of moving in the opposite direction. There is an obstacle in the center square, and the agents cannot move there. Each agent receives observations as to whether the squares to the left, right, above, or below each agent are blocked. With the obstacle in the middle, and the grid being blocked at the borders, there are 6 legal combinations of observations. Thus the domain has 6 observations. Observations are 100 percent reliable. When the agents reach the same square, a reward of 1 is received, and the problem repeats itself. This domain should be more favorable to *IMBDP*, since its weakness, the fact that it chooses a single action and does not explore the policy space for improbable observations, is not as relevant. In this domain, the *MBDP* planners typically pick a square to meet at (they pick this implicitly, through the policies they choose to retain), and once this is done, there is a clear single choice of action for each observation in the domain. One hypothesis would have *MBDP-OC* not do as well, since it may not be correct to consider the same policies for groups of observations. *MBDP-OC* can only be saved by using its value function, to assure that its merge operations will be as harmless as possible given the likely state. If it can do this successfully, it can more fully explore the policy space.

Experiments were run with $MaxTrees = 3$ and $MaxObs = 2$. Results are shown in Table 4.3. Indeed the table shows that under this domain, *IMBDP* was able to attain more comparable results. Still *MBDP-OC* held a small value advantage across experi-

horizon	IMBDP	MBDP-OC
1	0.0	0.0
5	.41	.47
10	1.01	1.04
20	2.01	2.38
50	5.30	6.38
100	11.6	13.0

Table 4.3. Comparison of *IMBDP* and *MBDP-OC* on the 3×3 Meeting in a Grid problem.

ments. Runtime for IMBDP was 76.4 seconds per horizon step, and for MBDP-OC it was 83.4.

4.6 Summary

This chapter examined meta-reasoning in a decentralized and partially observable (Dec-POMDP) environment. Dec-POMDP solvers are centralized at planning time and distributed at execution time. In this chapter, we propose a meta-reasoning mechanism in order to bound resource consumption in time and space at planning time.

There were two central contributions of this chapter. The first was the contribution of observation compression as the meta-reasoning mechanism used. Observation compression allows the planner to group observations together for the purposes of planning, and therefore to limit planning to the different groups of observations rather than each observation individually. The *CompressObs* algorithm performs this merge.

The second contribution was an augmented *MBDP* algorithm which uses the *CompressObs* algorithm. The resulting algorithm was analyzed both theoretically and empirically. Theoretically, it was shown that *CompressObs* is guaranteed to run in polynomial time, and that an online bound for loss can be computed when it is called. Empirically, it was shown that using *CompressObs* outperforms using the most likely observations only.

The work presented in this chapter was first published AAMAS 2008 [26] as well as the MSDM workshop in 2008 where it won best paper for that workshop. The format has been modified for this thesis, and a meta-reasoning element has been added. Since publication, newer planners have eclipsed the results in a point-based context. The newer works either (1) achieve their speedup by a means other than observation compression, and would achieve further speedup if combined with observation compression, or (2) include some variant of observation compression. In the former category are *PBIP* [42], and *PBPG* [152]. *PBIP* includes a heuristic search that may be combined with observation compression, and *PBPG* constructs a linear program that could be compressed. In the latter category is *GMAA* *-ICE, which clusters action-observation histories when it can be done losslessly [89], in contrast to the algorithms in this chapter which may be run in a lossless manner but are primarily used to bound loss. Also in the latter category is incremental policy generation [5], which eliminates unreachable states from evaluation.

It is possible to use *CompressObs* with other planners besides *MBDP* , to reduce the observations considered in planning. Since *CompressObs* tracks performance bounds as it operates, it is also possible to stop compression after a certain error threshold has been reached rather than to terminate when there are $MaxObs$ groups of observations left. Future directions of this work may combine the *CompressObs* algorithm with other planners in the literature besides *MBDP* .

CHAPTER 5

NON-MYOPIC CONTROL OF COMMUNICATION IN PARTIALLY OBSERVABLE MULTIAGENT SETTINGS

Chapter 3 introduced the problem of global monitoring for multiple agents. It outlined how the problem of global monitoring could be reduced to a transition independent Dec-MDP with communication. This chapter will begin by providing more details on this problem model and how to solve it. As in Chapter 3, this chapter builds on a value of information approach to communication. Typically the value of information approach makes two assumptions which we term the *myopic assumptions*. First, it approximates the value of information by assuming that each agent evaluates its information in isolation, without considering the reasoning process of the other agents. Second, a single step horizon is used in sequential decision making, each agent considers its next immediate action and not its ability to take action after that [111]. This myopic approach has been used in multi-agent systems with communication [50]. To begin this chapter, it is shown that the myopic approach biases the system towards overcommunication in Dec-MDP models, and a method is produced to obviate the bias through planning algorithms which look ahead and which consider the perspective of all agents in a Dec-MDP. This provides the details to support global communication as in Chapter 3.

Next, this chapter extends the approach to the partially observable multi-agent setting, specifically the Dec-POMDP model in Chapter 4. Chapter 4 analyzed a partially observable multi-agent environment where agents could not explicitly communicate. In this chapter we add communication to the model. We examine an extension to the Dec-POMDP model called Dec-POMDP-Comm-Sync. The complexity of producing optimal plans in this multi-agent, partially observable communication environment has been shown to be NEXP-hard [17, 118]. Whereas the Dec-MDP model which will be presented in Section 5.1 assumes transition independence, full local observability on the part of each agent, and joint full observability on the part of the group of agents, the Dec-POMDP extension presented in the second half of this chapter does not. As a consequence, when agents communicate to combine information in a Dec-MDP model, state becomes known to all agents, whereas in a Dec-POMDP model the combined information produces a state distribution.

The contributions of this chapter are as follows:

- *Evaluation of a 2-agent Dec-MDP model with communication using a bilinear representation and nonmyopic approach.* Section 5.1 will describe the Dec-MDP model with communication and develop the myopic approach for solving it. Then it will develop the non-myopic approach which jointly reasons about communication both in the present and the future. An evaluation is conducted using the bilinear programming approach as the basis for the developed algorithms [94].

- *Examination of the problem of when to communicate in partially observable multi-agent environments.* Section 5.2 produces communication algorithms for the Dec-POMDP-Comm model. The work for this partially observable model is similar to the work for the Dec-MDP in that a decision-theoretic approach is taken. However, this model relaxes the assumptions of joint full observability, full local observability, and transition and observation independence among agents.

- *Establishment of a data structure to efficiently track beliefs over status of other agents* in a communicative Dec-POMDP environment. The State Subtree (SST) data structure utilized in this chapter can be used by each agent to maintain its view of the global state and the state of the other agents. The advantages of the

SST representation, as presented in this chapter, are (1) Maintenance of SSTs is either lossless or introduces bounded loss, if so desired. (2) SSTs account for agent communication. (3) SSTs support a *merge* operation, which allows for a set of SSTs to be maintained efficiently in space.

It should be noted that the approach in the first part of this chapter, developing the myopic and non-myopic approaches to Dec-MDPs with communication, was first developed by Becker, Lesser, and Zilberstein at IAT [12]. The Section 5.1 approach is largely based on this work; it is based on the later finished work as presented in Becker, Carlin, Lesser, and Zilberstein [13], which (1) provided the software implementation for the theory, using a bilinear program as an underlying submodule unlike the original work, and (2) implemented myopic and non-myopic value of communication computations to obtain the empirical results depicted in this chapter. The work presented in Section 5.2 on Dec-POMDP-Comm was first presented by Carlin and Zilberstein at the MSDM workshop and IAT in 2009 [27].

5.1 Decentralized communication with full local observability

We begin with the model first mentioned in Chapter 3, the transition-independent decentralized MDP (TI-Dec-MDP) with global sync communication. TI-Dec-MDP is composed of n cooperative agents, each agent i working on its own local subproblem that is described by an MDP, $\langle S_i, A_i, \mathcal{P}_i, R_i, T \rangle$. The local subproblem for agent i is independent of local subproblems for the other agents. It is also completely observable, but only by agent i. At each step agent i takes action $a_i \in A_i$, transitions from state $s_i \in S_i$ to $s'_i \in S_i$ with probability $\mathcal{P}_i(s'_i|s_i, a_i)$, and receives reward $R_i(s'_i)$. All agents are aware of the transition models of all the other agents (but not necessarily their states or choice of actions at runtime). The *global* state of the domain is composed of the local states of all the agents. T is a finite number of horizon steps of the problem.

A communicative variant of this problem is referred to as Dec-MDP-Comm-Sync. At each time step, each agent first performs a domain-level action (one that affects its local MDP) and then a communication choice. The communication choices are *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 C_G 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 domain-level and communication 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 $t < T$ since the last synchronization to a domain-level action and a communication action, $\pi_i : S_i \times \langle S_1 ... S_n \rangle \times \{0..T\} \rightarrow A_i \times \{yes, no\}$. We will occasionally refer to domain-level policies and communication policies as separate entities, which are mappings from local state and time 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 domain-level reward function $R : S_1 \times ... \times S_n \rightarrow \Re$. To the extent that the global reward function depends on past history, the relevant history must be captured within the current state of the agents. 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.

Let a_i denote the domain-level action selected by $\pi(s_i)$. The global value function is:

$$V(s_1 ... s_n) = \sum_{i=1}^{n} R_i(s_i) + R(s_1 ... s_n) + \sum_{s'_1 ... s'_n} \mathcal{P}(s'_1..s'_n|s_1...s_n, a_1..a_n) V(s'_1 ... s'_n) \quad (5.1)$$

Transitions on the MDPs are independent of each other; we will therefore assume that without communication $\mathcal{P}(s'_1, ...s'_n|s_1..s_n, a_1..a_n) = \prod_{i=1}^{n} \mathcal{P}_i(s'_i|s_i, a_i)$.

The complexity of finding optimal policies for both the Dec-MDP with transition independence and the Dec-MDP-Comm-Sync classes of problems has been shown to be NP-complete [51], which is lower than the doubly exponential complexity (NEXP-hard) of general decentralized decision making.

5.1.1 Basic myopic approach

Using a myopic algorithm is a common way of dealing with the complexity inherent in finding an optimal solution. We start with a simple algorithm for determining when the agents should communicate. This algorithm is optimal assuming that communication must be initiated by the current agent (agent i in the following description) and that the current step is the only time step in which communication is possible. We denote the agents other than the current local agent with $-i$. Thus, reordering the agent numbers so that $i = 1$, without loss of generality, we refer to the states of the other agents as $s_{-i} = \langle s_2 ... s_n \rangle \in S_2 \times ... \times S_n$. While the problems solved in this chapter are distributed in nature (each agent chooses an action based on its own local view) the planning algorithm itself computes offline the policies for each agent in a centralized manner using a fully specified model of the problem. Then, the individual policies are given to the agents during execution time.

The algorithm works as follows. As long as no communication is initiated, each agent follows the optimal policy assuming no future communication, which was obtained at planning time using a subroutine such as the Coverage Set Algorithm (CSA) [11] or a bilinear program [94]. The subroutine takes a Dec-MDP with no communication as input, and provides joint-policies as well as their values as its output. At each state during execution time, agents choose whether to communicate or not by computing the net *value of communication* (VoC). If the $\mathrm{VoC} > 0$, then the agent initiates communication causing all of the agents to broadcast their local states. This synchronizes the local knowledge of all of the agents to the correct world state. The agents then compute a new optimal policy assuming no future communication, using the 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) $\langle s_i^0, s_{-i}^0 \rangle$, and the time t since the last synchronization. It also depends on the optimal joint policy assuming zero communication that the agents have been following since the previous synchronization, $\langle \pi_i^0, \pi_{-i}^0 \rangle$.

Definition 5.1. *The Value of Communication (VoC) is the difference between the expected value when communicating and the expected value for remaining silent.*

$$\mathrm{VoC}\left(s_i, \langle s_i^0, s_{-i}^0 \rangle, t\right) = \sum_{s_{-i}} P(s_{-i} | s_{-i}^0, t, \pi_{-i}^0) \left[V^*(s_i, s_{-i}) - C_G - V(s_i, s_{-i}) \right], \quad (5.2)$$

where $P(s_{-i} | s_{-i}^0, t, \pi_{-i}^0)$ is agent i's belief about the current state of all other agents, $V(s_i, s_{-i})$ is the expected value for following the current local policy from the joint state, and $V^*(s_i, s_{-i}) - C_G$ 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 5.1. *Assume that the optimal joint policy for each joint state can be found in polynomial time. 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:

- $P(s_{-i}|s^0_{-i}, t, \pi^0_{-i})$ is the t-step transition function for the nonlocal agents. Given the assumption that these agents will never initiate communication,

  $$P(s_{-i}|s^0_{-i}, t, \pi^0_{-i}) = \sum_{s'_{-i}} P(s'_{-i}|s^0_{-i}, t-1, \pi^0_{-i})\mathcal{P}(s_{-i}|s'_{-i}, \pi^0_{-i}). \tag{5.3}$$

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

- $V(s_i, s_{-i})$ and $V^*(s_i, s_{-i})$ are both expected values (see Definition 5.1). 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, $O(|S_i|^n)$.

- Finding the optimal joint policy without communication is assumed to take polynomial time, by hypothesis. For implementation details, see descriptions of the Coverage Set Algorithm [11] and the bilinear programming algorithm [94].

- 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 $O(|S_{-i}|) = |S_2| \times ... \times |S_n|$ times, exponential in the number of agents but run only once for the local agent.

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

□

While in the worst case the VoC computation must run a large number of times, $O(n|S|^{n+2})$, in practice the complexity of computation can be reduced for several reasons. First, many of the joint states and times 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 $\langle s_i^0, s_{-i}^0 \rangle$ remains low because the world state is only synchronized through communicating. Second, the value $V^*(s_i, s_{-i}) - V(s_i, s_{-i})$ is always greater than zero, and therefore the VoC computation can be terminated early (with a decision to communicate) if the value of communication is found to be greater than C_G for a subset of joint states. Third, when there is overlap in computation between calls to VoC, caching can reduce the running time.

5.1.2 Implications of the myopic assumption

The myopic assumption allows a simple, straightforward computation of the value of communication. While this may be a reasonable assumption for the single agent case, there are additional implications that may not be readily apparent in a multi-agent setting. We examine these implications by identifying and analyzing two sources of error in the basic myopic approach, illustrating each with a simple example.

5.1.3 Modeling the other agents

Define the *Basic* myopic approach as one which in which each agent runs its domain level policy, uses Definition 5.1 to compute VoC at each step, and communicates if VoC is greater than zero. Since every agent is following a communication policy based on computing the value of communication, this approach can lead to error. One implication that such an approach does not leverage is that when other agents do not communicate, this is in itself a form of communication. The distribution of states for the other agents

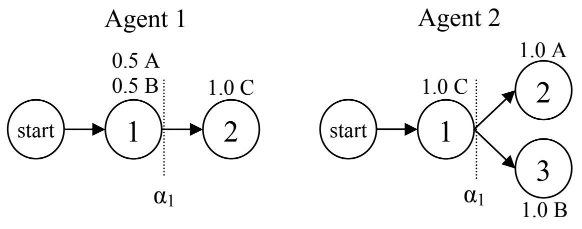


Figure 5.1. A simple example that illustrates how a myopic model for the other agent introduces error. Agents represent rovers, circles represent sites, letters represent the possible samples collected at that site, alongside a probability. Both agents move to site 1 at the first time step, but at the second time step agent 2 has a choice where to go. Joint reward is 10 if both agents collect an "A" or a "B". Reward is 1 for every "C" collected by either agent.

after t steps, $P(s_{-i}|s^0_{-i}, t, \pi^0_{-i})$, changes because the other agents are known to not have passed through states in which any would have communicated.

Another implication is that at the current step, agent i may not need to initiate communication to acquire valuable information from agent j if j can be relied on to initiate communication when it has the information. Figure 5.1 illustrates this with a simple two agent data collection example where agent 1 collects information valuable to agent 2. Suppose both agents represent rovers collecting samples of type A, B, or C. 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 a reward of 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 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% of the time the minimum reward of 2. When agent 1 collects a B, its VoC $= -C_G + 1.0[12 - 2] = -C_G + 10$. As long as the cost $C_G < 10$, agent 1 will initiate communication in this case. Agent 2 does not know what agent 1 has collected, so its VoC $= -C_G + 0.5[12 - 12] + 0.5[12 - 2] = -C_G + 5$. When the

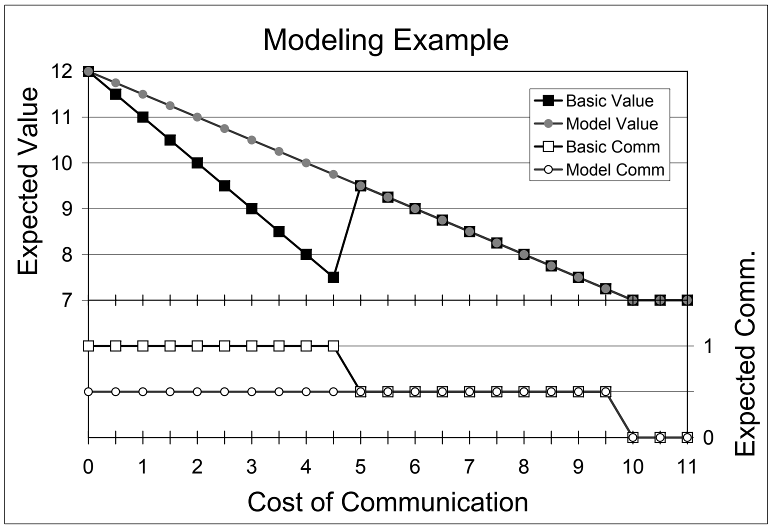


Figure 5.2. Performance comparison of the *Basic* and *Model* approaches.

cost of communication $C_G < 5$ agent 2 will communicate because its $\mathrm{VoC} > 0$. Half of the time this communication is unnecessary because agent 1 had collected an A. When $C_G \geq 5$ it is no longer valuable for agent 2 to initiate the communication and the *Basic* communication policies are optimal.

The *Basic* line in Figure 5.2 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 (still 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 agent 1

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

and columns representing states of agent 2 for the current step (see Figure 5.3).[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,

$$M_{xy} = P\left(s_1 = x | s_1^0, t, \pi_1^0\right) P\left(s_2 = y | s_2^0, t, \pi_2^0\right) \left[V^*\left(x, y\right) - C_G - V\left(x, y\right)\right] \quad (5.4)$$

In Figure 5.3, each row is summed up to construct a communication policy π_{1c} for the row agent, which indicates which row states to communicate from. Agent 1 only knows its local state or row. Similarly, each column is summed up to construct a communication policy π_{2c} for the column agent , as agent 2 only knows its local state or column. The *Basic* approach 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, the *Basic* policy given has a value of -1 (sum of the bold entries) because the valuable

[1]For the multi-agent case, this matrix is a tensor.

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.

The best joint communication policy is the joint policy that maximizes the bolded value of this table. However, there are $2^{|S_1|}$ possible policies for the row agent and $2^{|S_2|}$ possible policies for the column agent. Thus a brute force strategy which looks for the optimal joint policy would run in exponential time. However, a hill-climbing algorithm can find a Nash equilibrium in polynomial time. The *Model* strategy finds such an equilibrium by fixing all agent communication strategies in place while optimizing the remaining agent, then fixing that agent's strategy and performing the step again until equilibrium is reached. The line labeled *Model* in Figure 5.2 shows the performance of such a strategy, resulting in the best policy for the example in Figure 5.1. Creating the table costs no more than the original approach since each entry represents a reachable world state.

The *Model* approach described in this section is not to be confused with Q-POMDP [107], which is a technique designed to account for uncertainty of belief state in a multiagent POMDP. In Q-POMDP, each agent's environment is partially observable, and an agent will communicate when it deduces that communicating its state will change the action of the other agent, much like the *Basic* approach. It is enhanced to consider the true joint belief state in partially observable problems, but not the communication policy of the other agent. In the *Model* approach described above, each agent accounts for the state of the other agent as well as its communication policy.

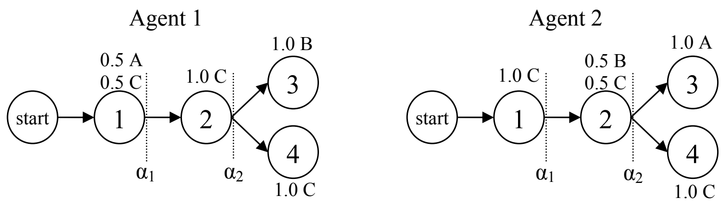


Figure 5.4. A simple example that illustrates how delaying communication can improve the expected value.

5.1.4 Myopic view of the future

The second facet of the myopic assumption is that no agent will communicate in the future. This introduces error in two ways. The first is due to the greedy nature of the algorithm. When communicating immediately has a positive value, $\text{VoC} > 0$, 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:

$$\text{VoC}_{delay}\left(s_i, \langle s_i^0, s_{-i}^0\rangle, t\right) = sum_{s_i'} P(s_i'|s_i, \pi_i^0) \times \max\left(0, \text{VoC}\left(s_i', \langle s_i^0, s_{-i}^0\rangle, t+1\right)\right). \tag{5.5}$$

The agent will initiate communication when its $\text{VoC} > \text{VoC}_{delay}$. 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 5.4 illustrates this with a simple example. Suppose again that agents receive higher reward for collecting the same sample type. If agent 1 collects A at site 1 then agent 2 should go to site 3, otherwise agent 2 should go to site 4. A similar situation occurs with agent 2 collecting B at site 2. As with 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

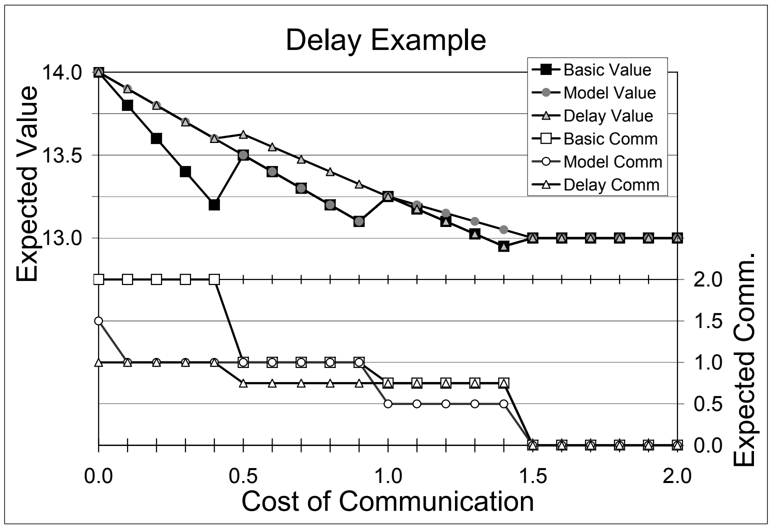


Figure 5.5. The expected value and expected amount of communication as a function of cost.

communication points. The *Basic* approach will always communicate at both α_1 and α_2 when the communication cost is low (See Figure 5.5). 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 initiate 50% of the time at α_2, for a total expected communication of $0.5 + 0.5 = 1.0$. The *Delay* policy, however, 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.

5.1.5 Model-lookahead approach

This section demonstrates how the *Model* approach of 5.1.3 and the *Delay* approach of 5.1.4 can be merged together and extended to consider further steps into the future. The basic idea is an algorithm that makes optimal communication decisions within a lookahead horizon h given fixed domain-level policies based on zero communication.

To start, we introduce two new value functions. $V^h(s_i, s_{-i})$ 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. $V^{*h}(s_i, s_{-i}) - C_G$ is similar but starts with an immediate communication. When the lookahead horizon is 0 these value functions are equivalent to the single-step value functions from Definition 5.1, $V^0(\cdot) = V(\cdot)$, $V^{*0}(\cdot) = V^*(\cdot)$.

$$V^h(s_i, s_{-i}) = \tag{5.6}$$

$$\sum_{s'_i, s'_{-i} \in \mathrm{Comm}} P(s'_i|s_i, \pi_i^0) P(s'_{-i}|s_{-i}, \pi_{-i}^0) \left[\mathcal{R}(s'_i, s'_{-i}) + V^{*h-1}(s'_i, s'_{-i}) - C_G\right]$$
$$+ \sum_{s'_i, s'_{-i} \in \neg\mathrm{Comm}} P(s'_i|s_i, \pi_i^0) P(s'_{-i}|s_{-i}, \pi_{-i}^0) \left[\mathcal{R}(s'_i, s'_{-i}) + V^{h-1}(s'_i, s'_{-i})\right]$$

where $\mathcal{R}$ is the sum of the reward functions, $\mathcal{R}(s'_i, s'_{-i}) = R_i(s'_i) + R_{-i}(s'_{-i}) + R(s'_i, s'_{-i})$. Comm is the set of states in which communication will take place. How it is computed becomes clear when we transform the equation as follows.

$$V^h(s_i, s_{-i}) = V(s_i, s_{-i}) \tag{5.7}$$

$$+ \sum_{s'_i, s'_{-i} \in \mathrm{Comm}} P(s'_i|s_i, \pi_i^0) P(s'_{-i}|s_{-i}, \pi_{-i}^0) \left[V^{*h-1}(s'_i, s'_{-i}) - C_G - V^{h-1}(s'_i, s'_{-i})\right]$$

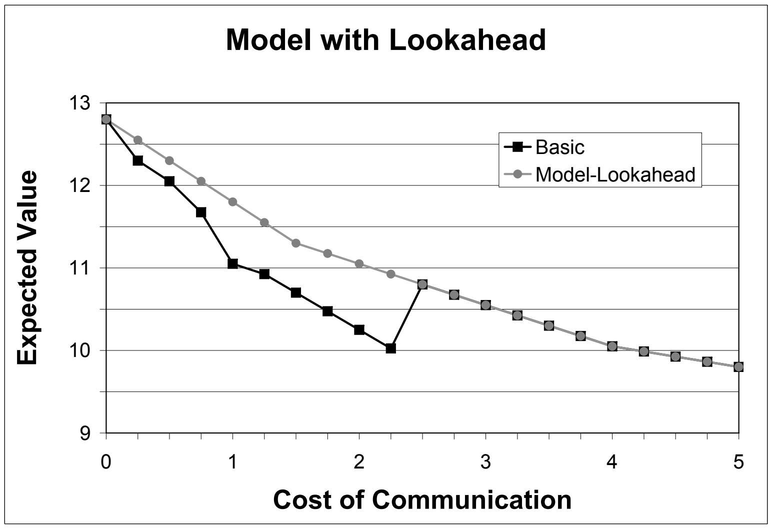


Figure 5.6. Performance of the *Model-Lookahead* Approach with lookahead horizon 2.

$$+\sum_{s'_i, s'_{-i}} P(s'_i|s_i, \pi^0_i)P(s'_{-i}|s_{-i}, \pi^0_{-i})\left[V^{h-1}(s'_i, s'_{-i}) - V(s'_i, s'_{-i})\right]$$

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 (5.7), which bears a remarkable similarity to Equation (5.4), except that this is a recursive function. Thus the same table algorithm can be applied to generate optimal communication policies over the lookahead horizon.

5.1.6 Experiments

Figure 5.6 illustrates the performance of this approach on a larger problem with 6 time steps. The Coverage Set Algorithm was used to find the underlying communication-free joint subpolicies [11]. The agents represent Mars rovers traversing sites and collecting data. State reflects the current site of the agent and data at that site, and battery life (from 0 to 8) remaining to the agent. The first agent's state and transition matrices correspond to Figure 5.7. Actions available are Move Left, Move Right, Wait, Quick Analysis, Detailed Analysis. The effects of Move Left and Move Right for the first

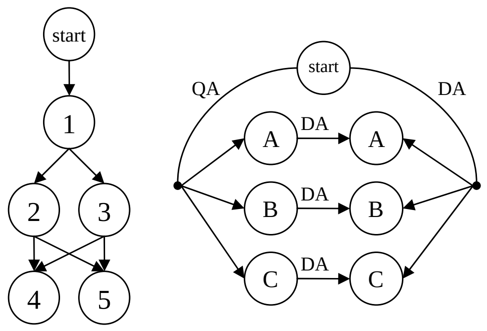


Figure 5.7. Graphical depiction of a sample decision problem. (left) A partially ordered list of 5 sites. (right) A decision problem for one site with three potential classes.

agent can be seen on the lefthand side of the figure. The second agent simply moves from site 0 to sites 1,2, and 3 in a straight line. The righthand side of the figure corresponds to the classes of data available at a specific site. There are 5 classes of possible data in all, A-E. Each site has a probability distribution over the classes available. A Quick Analysis determines the class at a site, and a Detailed Analysis actually obtains the reward. Each agent starts with 8 energy units on its battery. Movement costs one energy unit, as does a Quick Analysis. A Detailed Analysis costs two energy units. A reward of 10 is obtained for jointly obtaining classes A-D, while a reward of 1 is received for obtaining class E.

The *Model-Lookahead* approach performs significantly better than the original *Basic* approach and demonstrates a smooth and monotonic reduction of the expected value as the cost for communication increases.

Figure 5.8 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 lookahead horizon of 0 due to the added cost of finding the optimal communication policies of the tables. The worst case complexity of *Model-Lookahead* is exponential in the size of the lookahead horizon, but due to caching and the structure of the problem, in practice this is not always the case. In this example, the

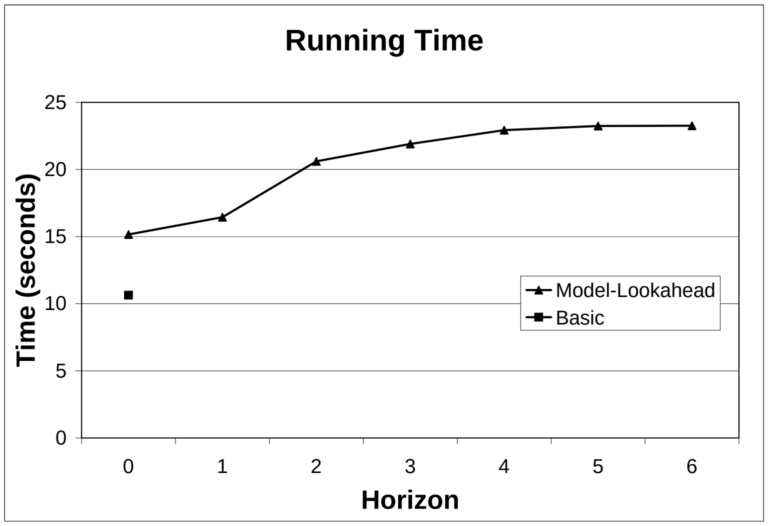


Figure 5.8. Comparison of the time to compute the policy for the *Basic* approach versus the *Model-Lookahead* approach of various depths.

running time started out with an exponential curve but that changed as the lookahead horizon approached the number of steps in the problem.

To further test the generality of the approach, we ran experiments on a second domain, using the bilinear programming technique to construct the underlying policies [94]. We also changed some of the characteristics of the domain, allowing actions to vary in duration as well as in their effects.

The selection of the domain was motivated by mapping scenarios from NASA and the US Geological Survey [82], whereby data from different imagers can be assimilated. Suppose our agents are sensors on separate satellites, which scan geographical locations on different bands. Data is most worthwhile if it gets scanned by both satellites at the same time. Actions available to the satellites are to Scan the current location or to Wait. Rewards can be both local and joint, for performing a scan. A joint reward is only received if the scan is initiated at the same time by both satellites. After a satellite is done scanning one location, it moves on to the next location. The time taken to perform a scan is a distribution. In contrast to the previous examples, the uncertainty in this domain is with respect to time, rather than with respect to the type of data collected. The bilinear programming technique used to solve the communication-free subproblems is newer and faster than the Coverage Set Algorithm. This allowed us to solve larger problems.

We first converted the state space in this example into a state space appropriate to the Value of Communication methodology. There are two types of states, the first type is when the satellite is at a location and can choose to scan or not to scan. This defines lh states where l is the number of locations, and h is the total horizon of the problem. To make the domain appropriate for Value of Communication analysis, it is necessary that each agent have a defined state for each time step. In order to assure this, one can simply include additional states for the case when a satellite has initiated a scan and is waiting for it to finish. This is a tuple (s, f, l), where s is the current time, f is the time at which the action will be finished, and l is the location of the agent when the scan is finished. Combinations of these tuples introduce lh^2 new states. A terminal state is also added. Thus, the total number of states is $lh + lh^2 + 1$.

In particular, we chose an example with $h = 8$ and $l = 4$. This defined 289 states for each agent, and 578 state/action pairs. We chose local rewards for the four sites to be $.5$, 5, 5, and 10 respectively. There was a shared joint reward of 20 if and only if the second site was explored by both rovers at the fifth time step, representing an interesting but time-critical event. The duration of the scan of the first site would always be one step for the first agent, and a uniform distribution centered at four steps for the second agent. Successive scans by both agents would take a mean duration of 3 with a standard deviation of 1.6.

Results are shown in Figure 5.9. The figure shows that–as we observed in the Rovers domain–following the *Basic* communication strategy results in overcommuni-

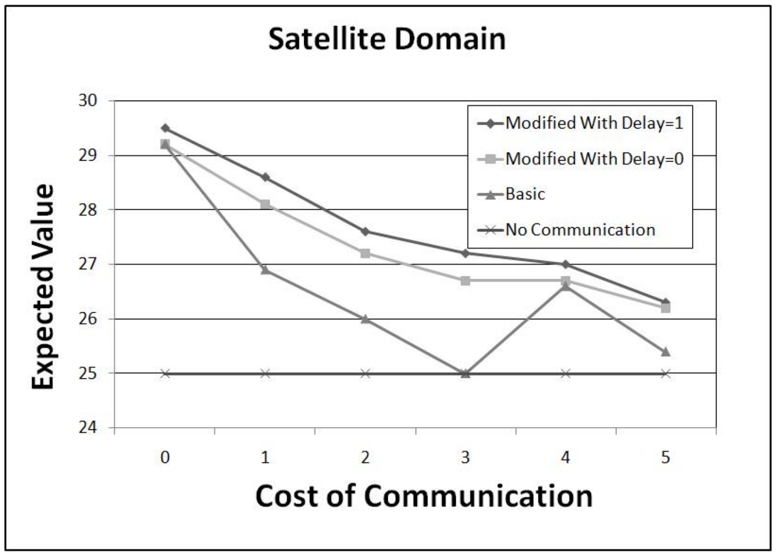


Figure 5.9. Results on satellite domain, showing the fixed value of no communication, compared with the values of the Basic strategy, Model with horizon 0, and Model with horizon 1.

cation. The key to this problem is that there is a large reward for completing all the scans, and an even larger reward for performing the valuable joint scan of the second site at step 5. The first satellite needs to choose between completing all the scans it can, versus waiting and attempting the joint scan.

Under the *Basic* strategy, the first satellite will overcommunicate after completing its first scan. The decision on whether communication is beneficial is mostly dependent on the second agent. If the second agent's scan terminates quickly, there will be time to synchronize for a second joint scan, and the agents should communicate in order to perform it. If it does not, then there is no need to communicate and synchronize. When the *Basic* policy is followed, the first satellite merely computes expected value of communication from its own perspective, without considering the policy of the other agent, and as a result overcommunication occurs, as described in the previous sections. The problem is corrected when the *Model Lookahead* policy with a delay is followed, which accounts for both the communication policy of the other agent as well as the ability to defer communication to future time steps. *Model Lookahead* consistently outperforms the *Basic* communication strategy, except when communication is either ubiquitous (at $Cost = 0$), or never useful.

To summarize, the *Model Lookahead* approach offers a simple but effective way to overcome the limitations of a naive myopic approach to communication. In two different domains it produced smooth and monotonous degradation of value as communication cost increases. This approach, however, does have its limitations. Even when the lookahead horizon is equal to the number of steps in the decision problem, the policy generated is not guaranteed to be an optimal joint policy. This is because the domain-level actions taken by the agents are generated assuming no future communication. Future work will focus on extending this algorithm to allow a larger domain-level action lookahead horizon.

5.2 Communicating with partial observability

This section adds partial observability to the model of the last section. Each agent takes an action, receives an observation, and then has the option to initiate communication before taking the next action. Like the last section, we restrict discussion to the *sync* communication model, so the communication language allows transmission of the agents' complete action/observation histories before each action. Communication is instantaneous; a message is received without delay as soon as it is sent. A constant C_G, representing a fixed cost on each step of communicating these synchronization messages, is specified. The fixed cost of C_G is incurred if *any* number of agents chooses to communicate. Otherwise, if no agent communicates, they incur no penalty. The problem is NEXP-hard. This is shown by considering the case when communication is prohibitively expensive, in which case the model becomes a Dec-POMDP with no communication.

Since the problem has a finite horizon T, one can use a policy tree to represent a non-communicative policy of an agent. As seen in previous chapters, in the policy tree representation nodes represent actions and branches represent observations. Each agent i follows its own policy tree generated at the last synchronization step, referred to as π_i^0 with its first action corresponding to the root at time $t = 0$, and its last action corresponding to the leaves. π_i^0 contains a number of subpolicies, each corresponding to an observation sequence as the tree is traversed. An observation sequence of a single agent is referred to as $\bar{o}_i$ and the resulting subpolicy as $\pi_i(\bar{o}_i)$. Note that if an agent's initial policy and its sequence history of observations are known, one can derive its sequence of actions. Furthermore, the next sections will show that the local history of an agent can be combined with Bayesian reasoning on the Dec-POMDP model and the initial policies of the other agents to form a belief about the histories of other agents. To summarize, each node of an agent's policy tree maps to:

- A unique action/observation sequence $\bar{o}_i$
- A future local subpolicy rooted at the node $\pi_i(\bar{o}_i)$
- A belief about the global state as well as the action/observation histories of the other agents.

Let $b(s)$ be a belief state, and let s' be a variable representing a successor state. Let a_i and a_{-i} be the root actions of policies $\langle \pi_i, \pi_{-i} \rangle$. The value of a joint policy tree, $\langle \pi_i, \pi_{-i} \rangle$ at a given belief state is recursively defined as the expected sum of the rewards of the subpolicy trees. That is, at the base level, the value of one-step policies is the associated reward:

$$V(a_i, a_{-i}, b(s)) = \sum_{s \in S} b(s) R(s, a_i, a_{-i}) \tag{5.8}$$

And the value of multi-step policies is defined:

$$V(\langle \pi_i, \pi_{-i} \rangle, b(s)) =$$

$$\sum_s b(s) R(s, \pi_i(\epsilon), \pi_{-i}(\epsilon)) +$$

$$\sum_{s,s',o_i,o_{-i}} \left[b(s) P(s'|s, a_i, a_{-i}) O(o_i, o_{-i}|s', a_i, a_{-i}) V(\langle \pi_i(o_i), \pi_{-i}(o_{-i}) \rangle, s') \right]$$

The above says that the value of the joint policy at b can be decomposed into cases where the root actions result in a transition to state s', resulting in observations o_i and o_{-i}.

5.2.1 Solution method

This section develops a method whereby plans and communication strategies are determined offline and stored for use at runtime. The method begins by pre-computing optimal joint policies without communication. Any non-communicative Dec-POMDP planner which generates policy trees (e.g. *MBDP*) can be used for this step. It also pre-computes non-communicative joint policies for (some or all) reachable belief states of horizons $1...T$ (more details on this are in the next section), and stores these policies and their value in a cache. It uses these to construct a cache function for reachable belief distributions on the global state, and at runtime the cache is accessed by each agent through a function call:

$$\text{CACHE}_i(b(s), t) \rightarrow \langle \pi_i^*, \pi_{-i}^* \rangle$$

where i is the identity of the local agent accessing the cache, $b(s)$ is the belief state it wants to query, h is the depth of the policy and π^* represents that the policy is specific to that belief state. Optionally, one can store a mapping of some or all observation sequences to communications decisions (if these are not stored, they must then be computed by the agent at execution time).

$$\langle b(s), \bar{o}_i \rangle \rightarrow \{\textit{true},\textit{false}\}$$

where $\bar{o}_i$ is a vector composed of the observations agent i has made on prior steps. At execution time, each agent follows its policy and its communications policy. Upon communication, it retrieves the appropriate policy from the cache (in the case of a cache miss, it computes the appropriate policy) for the discovered belief state. We note trivially that if agents' policies are known to each other, then a joint observation sequence $\langle \bar{o}_i, \bar{o}_{-i} \rangle$ also determines a unique action history, and a unique $b(s)$ can be constructed by starting at the initial belief state and performing a belief update as in a POMDP.

Before each action, each agent decides whether to communicate. To do this, it uses the *Value of Communication*. Let $Pr(s', \bar{o}_{-i} | \bar{o}_i, \langle \pi_i, \pi_{-i} \rangle, b_0)$ represent the probability of reaching state s' while the other agents receive observations $\bar{o}_{-i}$ after $|\bar{o}_i|$ steps, given a starting belief state b_0 with policies $\langle \pi_i, \pi_{-i} \rangle$, and local observation sequence $\bar{o}_i$. (The computation of this probability will be deferred to the next section). Let $\langle \pi_i, \pi_{-i} \rangle$ be the joint policy before communication and $\langle \pi_i^*, \pi_{-i}^* \rangle$ be the joint policy that results from communication and discovery of a joint belief state.

Definition 5.2. *The Value of Communication (VoC) is the difference between the expected value when communicating and the expected value for remaining silent.*

$$VoC(\bar{o}_i, \langle \pi_i, \pi_{-i} \rangle, b_0) = \sum_{s'} \sum_{\bar{o}_{-i}} P_{s', \bar{o}_{-i}} (V^* - C - V)$$

where

$$\begin{aligned} P_{s',\bar{o}_{-i}} &= Pr(s', \bar{o}_{-i} | \bar{o}_i, \langle \pi_i, \pi_{-i} \rangle, b_0) \\ V^* &= V^*(\langle \pi_i^*, \pi_{-i}^* \rangle, s', t) \\ V &= V(\langle \pi_i(\bar{o}_i), \pi_{-i}(\bar{o}_{-i}) \rangle, s', t) \end{aligned}$$

b_h is the belief distribution at time h given $\langle o_i, o_{-i} \rangle$ and b_0.

To understand the above definition, consider the perspective of agent i. It had synchronized with the other agents at time $t = 0$ and had jointly entered belief state b_0, it knows that the other agents have been following policies π_{-i} since then, and that it has observed $\bar{o}_i$ since synchronization. In order to contemplate the value of remaining silent, it must consider the joint probability that the other agents' have observed $\bar{o}_{-i}$, and that the real current state is s'. If this is the case, it knows that the agents will continue along subpolicies $\langle \pi_i(\bar{o}_i), \pi_{-i}(\bar{o}_{-i}) \rangle$, and the value of staying silent is the value of the joint subpolicy from state s. If the agents do communicate, they will combine observations to form a new joint belief state b_t, and they will follow a new joint policy for the belief state, $\langle \pi_i^*, \pi_j^* \rangle$. The new joint belief state does not affect the fact that the true state is s', and so it computes the value of the new joint policy for s'.

For example, consider the multiagent Tiger problem [85]. In this problem, agents take a joint action of opening either the right or the left door, or they listen for the tiger. If both agents choose to listen, each agent will then receive its own observation with some defined probability of error, and because of this probability of error it is possible that both agents will not receive the same observation. Consider the perspective of an agent after it has listened and observed Tiger-Left. In order to evaluate the value of communicating, the agent must consider each scenario that occurs after communication, one of which is the chance that the other agent has also observed Tiger-Left, that they use the combined observations to open the door on the right, but that the true state was

Algorithm 5: Find SSTs at current step

```
input  : Synchronized Belief State $b_0$, Nonlocal joint policies $\pi_{-i}$, Local observation
         history $\bar{o}_i$, Local action history $\bar{a}_i$, time step $t$, threshold. A Dec-POMDP tuple
         $< S, A, P, \Omega, O, R >$
output : An array of SSTs, each containing the true state, the remaining policies of the
         other agents, and a probability
begin
    $D, D' \leftarrow$ arrays of StateSubTrees, initialized to empty
    for $k = 1$ to $|S|$ do
        $D[k] \leftarrow \langle s_k, \pi_{-i}, b_0(s_k), false\rangle$
    for $step = 1$ to $t$ do
        $D' \leftarrow$ empty
        for $k = 1$ to $|D|$ do
            $\bar{a}_{-i} \leftarrow$ joint action at root of $D[k].tree$
            $a_i \leftarrow \bar{a}_i[t]$
            $o_i \leftarrow \bar{o}_i[t]$
            for each state $s'$ in $S$ do
                for $o_{-i} = 1$ to $|\Omega_{-i}|$ do
                    $SST \leftarrow$ new SST
                    $p_{SST} \leftarrow$
                    $(D[k].p)P(s'|D[k].s, a_i, a_{-i})O(D[k].s, a_i, a_{-i}, o_i, o_{-i}, s')$
                    if nonmyopic then
                        for each nonlocal agent $j$ do
                            Lookup its communication policy
                            if $CACHE\langle b_0, \bar{o}_j\rangle \rightarrow$ true then
                                Prune this SST
                        if $SST.comm ==$ true then
                            prune SST
                    $SST.s = s'$
                    $SST.p = p_{SST}$
                    $SST.\Pi$ = $D[k].\Pi.subTrees[o_{-i}]$
                    Add $SST$ to $D'$
        Merge SSTs with equivalent subpolicies
        Prune SSTs with $p <$ threshold from $D'$
        Normalize each $SST.p$ in $D'$
        $D \leftarrow D'$
    return $D$
end
```

Tiger-Right, resulting in a large penalty. Although the possibility of this happening is small, it will motivate communication if the penalty is large enough.

5.2.2 Estimating the joint history

This section explains how $Pr(s', \bar{o}_{-i}|\bar{o}_i, \langle \pi_i, \pi_{-i} \rangle, b_0)$ is computed. Any such computation must address these features of the Dec-POMDP model which differ from the Dec-MDP model: (1) the local agent's history of actions has affected non-local transition probabilities (2) the other agents have adjusted their actions based on their observation history, not the true state (3) each local agent only holds its own observations, not necessarily the observations of the other agents.

Definition 5.3. *Let a State SubTree (SST), be a tuple $\langle s, \pi_{-i}, p, comm \rangle$, where s is a state, π_{-i} is a finite-horizon (joint) policy, p is a probability, and comm is a boolean.*

Algorithm 5 shows how $Pr(\bar{o}_{-i}, s'|\bar{o}_i, \langle \pi_i, \pi_{-i} \rangle, b_0)$ is estimated. The algorithm takes as input initial belief state b_0, the action and observation histories of the current agent i, and the known policies of the other agents at b_0. It outputs a set of SSTs at the current time step, each SST assigns a probability to one world state, composed of the actual state and the current policy of the other agents. SSTs are computed in a forward fashion. The set of SSTs is initialized to contain one element for each nonzero state in b_0, with its p being $b_0(s)$, its probability of being in that state, and its Π being the initial joint policy of the other agents. At each time step, the current set of SSTs is used to generate a new set. Each SST in the new set represents a joint action taken by the other agents, a joint observation received, and a global state transition from an old SST, resulting in the new SST's state and subpolicy. The forward probability p_{SST} is the probability of the old SST times the probability that the other agents made this transition, given the local agent's knowledge of its own action and observation on that step.

SSTs with the same subpolicy are merged. That is, if two observation histories of the other agents lead to the same subpolicy, there is no need to distinguish the two cases.

Formally, if there are two SSTs:

$$\langle s, \Pi, p1, comm \rangle \quad \text{and} \quad \langle s, \Pi, p2, comm \rangle,$$

they can be merged into a single SST:

$$\langle s, \Pi, p1 + p2, comm \rangle.$$

This can be particularly useful in practice, if the non-communicative plans were built by an algorithm based on *MBDP* , which builds plans where only a limited set of subpolicies are generated, and different observations lead to the same subpolicy.

In practice, other augmentations may be beneficial to Algorithm 5. (1) The cache can be smaller and only contain likely decision points. At run-time, when a non-cached state is encountered, the agent can either initiate an online computation, or it can use the joint-policy from the least (Manhattan) distant cached belief-state. (2) SSTs can be generated by sampling from agent histories, rather than direct computation.

The following theorem proves correctness of Algorithm 5.

Theorem 5.2. *Suppose agent i calls Algorithm 5 with threshold 0 at time t after observing $\bar{o}_i$, and the algorithm returns a set D of SSTs. Then $\forall D[k] \in D$, if $D[k] = \langle s, \Pi, p, comm \rangle$, then p is the probability that the global state is s and the other agents' policies on this step are Π at time t.*

The theorem is a consequence of the following Lemma.

Lemma 5.1. *The problem of constructing set D at time t has an equivalent Hidden Markov Model (HMM) representation.*

Proof. We can convert the problem of estimating $M.p$ into an HMM, and then solve using the forward-backward algorithm [108]. Each state of the HMM corresponds to a global state and an observation history of the other agents (we use the fact that each joint observation history maps to a specific joint subpolicy such as Π_{-i}). State transition probabilities of the HMM correspond to state transition probabilities of the Dec-POMDP, given the local agent's action histories, times the probability of making the last observation. The transition probability is zero if the new observation history cannot follow from the old. That is, a state with an observation history $w_1 w_2$ cannot transition to a state with an observation history $w_2 w_2 w_3$, but it can transition to a state with observation history $w_1 w_2 w_3$.

Given this transition model, it is can be shown through induction (with the base case consisting of S when Algorithm 5 is initialized) that the forward computation used to generate the leaves in the last step of Algorithm 5 are the same as the steps used to generate the corresponding states in the Hidden Markov Model. $\square$

The number of SSTs can grow exponentially in each step, in the worst case. In practice, however, the number only grows with reachable belief states, and often on real-world problems only a small number of observations will be possible on each step. In order to keep the algorithm tractable, the algorithm can optionally prune SSTs with low probabilities at each step.

5.2.3 Myopic algorithm

Algorithm 6 shows the myopic procedure executed by each agent. Belief state b_0 is initialized at the beginning of execution and known to all the agents. Each agent executes its policy π_i one step at a time, adding the received observation o_i to its history. It maintains a set of SSTs via the $FindSSTs$ procedure. For each SST, it finds the value of not communicating (thus continuing the current policy), versus the value of

Algorithm 6: Basic Myopic Execution

```
input  : initial belief state $b_0$, joint policy $\langle \pi_i, \pi_{-i} \rangle$
output :
for $t = 1$ to $T$ do
    $a_i \leftarrow \pi_i(\bar{o}_i)$;
    execute $a_i$;
    receive observation $o_i$;
    $\bar{o}_i \leftarrow \bar{o}_i + o_i$;
    $D \leftarrow FindSSTs(b_o, \pi_{-i}, \bar{o}_i, a_i, t)$;
    $V = 0$;
    $V^* = 0$;
    $\forall s, b_t(s) \leftarrow 0$;
    for $k = 1$ to$|D|$ do
        $b_t(D[k].s) = b_t(D[k].s) + D[k].p$;
    Sum SSTs to determine belief state $b_t$;
    $\pi_i^*, \pi_{-i}^* \leftarrow MBDP(b_t, T - t)$;
    for $k = 1$ to $|D|$ do
        $V = V + (D[k].p)V(s, \pi_i(\bar{o}_i), D[k].\Pi_{-i})$;
        $V^* = V^* + (D[k].p)V(s, \pi_i^*(\bar{o}_i), D[k].\Pi_{-i}^*)$;
    if $V^* - V > C$ then
        communicate;
        agree on new $b_0$;
        $\pi_i, \pi_{-i} \leftarrow MBDP(s, T - t)$;
        $\bar{o}_i \leftarrow \epsilon$;
```

communicating and finding a new policy. The value of communicating is determined by calling a planner on the current belief state, the pseudo-code in Algorithm 6 uses the *MBDP* planner, but any planner can be used that begins with an initial belief point and derives a joint policy. The belief point b_t is derived from the SSTs, each SST contains a state and a probability, so belief point b_t is obtained by summing the probabilities for each state. All SSTs are considered, thus an expected value of communicating and not communicating is found. If the difference in expected value exceeds the communication cost, the agent communicates. After communication, agents will construct a new belief state b_0 out of the joint history and find a new optimal joint policy for this belief state.

5.2.4 Non-myopic reasoning

Algorithm 5 does not take into account the communication policy of the other agents, nor does it take into account the fact that communication need not be immediate, it may be deferred to future steps. In this section, we discuss how we improve the algorithm past this *myopic assumption*. The algorithm can be improved in three ways, first by using the fact that other agents did not communicate since the last *sync*, second by using the fact that other agents can communicate in the present, and finally by using the fact that communication can be deferred to the future.

5.2.4.1 Other agents in the past

Each agent knows the time of the last communication, and agents share a communication policy each time they synchronize. Therefore, information can be inferred when other agents have not communicated since the last synchronization. To do this, we use the *comm* field in the SST structure. At planning time, each agent computes VoC given its possible observation sequences and synchronized belief states. If VoC is positive, it sets *comm* to true. The *comm* value is stored for this history.

As Algorithm 5 is executed, each SST represents one possible observation history of this agent, and its children represent a continuation of that history. If the *comm* field is set to true for a corresponding observation history, this means that the agents would have communicated at this point. But any agent executing Algorithm 5 knows that didn't happen, since no agents have communicated since synchronization. Therefore it is known that the observation histories represented by such an SST never occurred, and the SST can be pruned.

5.2.4.2 Non-myopia with respect to other agents

Having modeled the communication strategy of the other agent on past steps, we turn to modeling the present step. To do this, we construct a tensor. For the two agent case, the tensor is a matrix and each row of the matrix corresponds to the SSTs for one agent, and each column corresponds to the SSTs for the other agent (for the multiagent case, each dimension represents another agent). Entries in the matrix correspond to the VoC given the history represented by the corresponding joint history, multiplied by the probability of that joint history. Each agent has the ability to communicate or not to communicate given a history. Communicating after a history corresponds to turning a row (or column, for the other agent) "on" or "off". The value of a joint communication strategy is the sum of the "on" values in the matrix. The myopic strategy discussed in above sections corresponds to turning each row or column on if its entries sum to a positive number. However, this illustrates the flaw of myopia, it does not maximize the value of the whole matrix, only its individual rows and columns. Since the row agent and column agent are not coordinating, they may double count entries. We improve on this by finding a better joint strategy. The approach is similar to the one described in [13], except (1) The rows and columns and probabilities correspond to observation histories, not states. (2) To reduce time of computation, agents can only alter K rows, where K is a parameter specified by the users. The remaining rows are toggled through myopic computation. As noted in [13], it takes an exponential amount of time with respect to the matrix size to find an optimal row/column strategy, but finding a Nash equilibrium is a reasonable alternative which can be done in polynomial time. Thus, in our implementation we find a Nash equilibrium.

An example can be found in Figure 5.10. Each entry in the table corresponds to VoC for a single joint history. Using an agent-myopic strategy, Agent 1 has decided that it should communicate given the history represented by s_1^2, because its VoC of 2 (the sum of its row) is positive, and Agent 2 has decided that it should communicate from state s_2^1, because its VoC of 1 is positive. VoC decisions are shown in the figure as π_{1c} and π_{2c}. In the figure, all joint-histories that result in communication under a myopic strategy are bolded. 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. Because we are using the sync model of communication, an entry is only counted once, even if both agents initiate communication. This corresponds to all joint-states where communication happens, weighted by their probability. In the example, the myopic policy has a value of -1, computed by summing the bold entries. The reason for this negative value is because the joint history in the first column of the second row 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.

Creating the table costs no more than the original approach since each entry represents a reachable joint history. Note that in problems with structure, or where communication has occurred on a recent step, the number of reachable joint histories is limited. For larger problems, though, there will be a large number of reachable joint histories, and in future work we plan on reducing the dimensionality of the matrix while minimizing loss of information. Equilibrium solutions to the reduced matrix problem will correspond to reasonable joint communication policies.

5.2.4.3 Value of deferring communication

The value of deferring communication to the future can be computed. For a given SST, the value of delay is the reward achieved by not communicating on the current step, added to the expected reward after communicating on the next step. The immediate reward is $(SST.p)R(s, \mathbf{a})$ and it is added to:

	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 5.10. A simple table M showing the expected gain in value for communicating for the two agent case. Each row represents an agent history for agent 1. The table represents 3 possible histories for each agent, or 9 belief states overall.

$$SST.p \sum_{s',\mathbf{o}'} P(s'|\mathbf{a}, s) O(\mathbf{o}'|\mathbf{a}, s') \mathbf{V}(\langle \pi_i^*(b_{t+1}), \pi_{-i}^*(b_{t+1}) \rangle, s')$$

where p is the probability associated with the SST, $\mathbf{a}$ is the joint action specified by continuing the current policy of the local agent and the SST, s is the state in the SST, $\mathbf{o}'$ the next joint observation, and b_{h+1} is the belief state that would result at the next step. $\mathbf{V}$ is used to represent the fact that VoC must be retrieved for the local agent's observation in $\mathbf{o}'$, and if it is positive then $\mathbf{V} = V^*$ and b_{h+1} is the belief state that results from communication while if $\mathbf{V}$ is negative, $\mathbf{V} = V$ and the joint policy continues. To compute the value of delaying communication, the computation above is summed for all SSTs returned by algorithm 5. If the sum is greater than or equal to the value of communicating on the current step, the agent does not communicate. A new value of delay will be computed after the next action is executed. Because of this, it is possible that the decision to postpone communication will cascade across several steps.

5.2.5 Experiments

We considered our algorithm, labeled *VoC-NM* (Value of Communication - Non-Myopic), as compared to the algorithms of *No Communication* (labeled No-Comm),

horizon	Cost	No-Comm	Periodic	VoC-NM
3	0	5.19	12.5	12.5
3	5	5.19	5.46	7.99
3	10	5.19	5.19	6.03
5	0	4.92	26.2	26.2
5	5	4.92	6.3	9.14
5	10	4.92	4.92	5.62
8	0	9.00	41.8	41.8
8	5	9.00	12.3	24.3
8	10	9.00	9.00	10.6
10	0	9.4	53.2	53.2
10	5	9.4	12.87	22.7
10	10	9.4	9.4	11.9

Table 5.1. Comparison of various communication strategies for the Tiger problem.

Full Communication (communicating on every step), *Periodic Communication*, as well as the algorithm of Roth et al. For the *Periodic* strategy, we ran an algorithm which communicated every K steps, and we used results from the best value of K from 1 to the horizon of the problem. Thus, *Periodic* will provably outperform *No Communication* and *Full Communication*, so we do not separately list results for full communication. Our algorithm was implemented as follows: we precomputed values of communication for each agent for reachable histories at planning time by running a large number of simulations, and then stored this in a cache. We used a pruning threshold of 0, thus we did not prune SSTs. We used the IMBDP planner [119] as the non-communicative submodule for this step. Then we ran a new 100,000 simulations of the non-myopic algorithm, referencing this cache on each simulation. Since MBDP-based planners only store a handful of subpolicies for each horizon step (using the same subpolicies for various branches of the larger policy tree), this choice of planners kept the size of the cache smaller.

horizon	C=0	C=5	C=10	C=15
3	3.0	.44	0	0
5	5.0	1.4	.92	0
8	8.0	1.9	.79	.02
10	10	2.5	.72	0

Table 5.2. Average number of communications for each run of the *VoC-NM* strategy on the Tiger problem. Each row represents results for a different horizon.

horizon	Cost	No-Comm	Periodic	VoC-NM
5	0	59.6	78.7 (4.0)	78.7 (4.0)
5	15	59.6	64.3 (1.0)	64.9 (.89)
5	30	59.6	60.3 (1.0)	64.1 (.80)

Table 5.3. Comparison of various communication strategies for the BoxPushing-5 problem. Parentheses show the mean number of communications for each simulation.

Results for the Multiagent Tiger problem [85] on various horizons are shown in Table 5.1. Results show that the *VoC-NM* planner was able to successfully communicate for both lower and higher costs of communication. We also performed experiments using the planner from Roth et al. [106], which was available for use with the Tiger domain. Note that this planner was not constructed with Cost of Communication in mind; it develops the policies first and then each agent communicates when it considers the state of the other agent ambiguous. It also uses the *tell* model of communication. Thus we do not present the results side-by-side in the table. Still, it is interesting to compare [106] as an alternative to *VoC-NM*. *VoC-NM* outperformed the Roth et al. planner on all experiments. On the horizon 10 problem, *VoC-NM* outperformed the Roth et al. planner by 11% and 36% respectively on communications costs 5 and 10. The difference in performance continues to increase as the cost of communication increases. We also sought to compare to other works in the literature by requesting Comm-MTDP [99] and Communicative DP-JESP [86]. The former is implemented within a framework which evaluates based on input non-communicative policies but does not generate the policies themselves, and the latter was unavailable for experimentation.

Results for the *VoC-NM* strategy are more closely examined in Table 5.2. Execution of the *VoC-NM* strategy was simulated 100,000 times for each cost of communication (C). Each entry is the mean number of communications for per simulation, given that cost of communication. For example, the column C=0 represents a configuration where there was no cost of communication, and thus the agents communicated at every step. As expected, the table shows that the expected number of communications decreases as the cost of communication increases.

Running time for *VoC-NM* was 9 seconds for the precomputation, and 2 seconds for the $100,000$ simulated runs after that. We also ran a myopic variant of the *VoC* planner, it did not include the algorithm enhancements of Section 4.2. The result across all tests was an approximately 10% decrease in score at C equal to 5 or 10.

We also ran the larger BoxPushing problem [119] for horizon 5, a problem in which the value of the generated centralized and decentralized plans only differ by 20. Still, results similarly show that a *VoC-NM* methodology outperformed the other strategies because it communicates less, resulting in a gradual decrease in value as communication cost gets higher. The time taken for BoxPushing-5 was 4300 seconds at the planning stage, and then $.38$ seconds to run each simulation at execution time.

Across all experiments, a simple communication policy such as *Periodic* can be adequate when communication cost is low, or when communication points can easily be picked from the domain. As the cost of communication gets higher, and agents are motivated to avoid communication if possible, the *VoC-NM* approach is required. Even assuming, as we did, that the best period can be determined, a periodic communicator is forced to either choose to not communicate at all, or else to overcommunicate. This was shown as the *VoC-NM* approach reduced the amount of communication by 10% on BoxPushing when communication cost was 15, and by 20% when cost was 30.

5.3 Summary

This chapter has presented a general approach for reasoning about costly communication for both the Dec-MDP and the Dec-POMDP models. For both models, a value of information approach was used. However, two sources of error were identified within this approach, which were referred to as myopic assumptions and defined the *Basic* approach in this chapter. These sources were, respectively, the inability of a value of information approach to consider the ability of other agents to communicate, and the inability of such an approach to consider deferring communication to future steps. For Dec-MDP, both of these sources of error may lead to overcommunication. The *Model* and *Delay* approaches were developed in order to modify the approach to address these issues, and the *Model-Lookahead* approach combined the features of the improved approaches. Experiments highlighted situations where the *Model-Lookahead* approach outperformed the *Basic* approach, specifically on problems where the decision to communicate was borderline. In the experiments shown, expected value in the *Model-Lookahead* approach degraded smoothly with value of communication, whereas in the *Basic* approach it did not.

The Dec-POMDP framework added partial observability to the problem. This variant was more challenging, as computations of value after communication became computations of expected value after communication, because even after communication the environment was partially observable. Furthermore, in Dec-POMDP each agent receives different partial observations and must reason about the observations of the other agent, as well as the possible synchronized state of the system after communication. Still, this chapter showed that computing the value of communication can be used effectively to determine the utility of communicating versus the utility of staying silent. A data structure, the State Subtree (SST), was introduced which allows for efficient computation of this value. The SST allows for a merging operation which can reduce computational overhead, and furthermore it was shown that in *MBDP* -based planners this merging operation can be used quite frequently. As opposed to previous approaches, the merging operation can be used to make problems tractable while bounding loss.

CHAPTER 6

RELATED APPLICATION: META-REASONING WITH A HUMAN IN THE LOOP

This chapter presents an application related to the concepts developed in this thesis. It illustrates how meta-reasoning can be applicable to the domain of air safety, specifically in-flight alerting systems for pilot decision support. In Chapter 2 and Chapter 3, we saw how performance profiles can be used to inform meta-reasoning processes in order to optimize the performance of the base-level decision makers. This chapter explores the question – what if one of the decision makers is a human? Due to the nature of the application, the framework in this chapter differs from the previous chapters in several ways. First, as mentioned, one of the base-level decision-makers is a human. Second, the meta-reasoning process is co-located with the base-level reasoning process in one decision maker (the automated aircraft system) but not the other (the human). Third, time is treated as continuous.

In the work related to Chapters 2 and 3, performance profiles could be obtained by either using known performance characteristics of the base-level procedure or by running the procedure a large number of times and accumulating performance statistics. The inclusion of a human decision maker for this task presents a set of research challenges. The challenges include the estimation of the current state of the human decision maker as well as the decision of how to best communicate with the human decision-maker. Communication will be addressed by issuing pilot notifications at varying *stages of automation* [91], a concept which is elaborated in this chapter. Each stage of automation is associated with a pilot performance profile, and the best stage is selected according to the best profile to use as selected by decision theory, specifically a Time-Dependent Markov Decision Process (TMDP) model [22]. In order to construct the performance profiles at each stage of automation, a pilot state model is constructed.

The contributions of this chapter are as follows:

- A design for a meta-reasoning system which allows an automated aircraft system to cooperate with a pilot system in order to address aircraft hazards.

- A TMDP implementation of a planner which produces plans for the optimal stage of automation at each given point of time, given the level of aircraft alert and pilot state model.

- Demonstration software which includes (1) a Bayes network to estimate the severity of hazards using the evidence of the aircraft alert systems (2) A system developer interface, which allows the pilot model and Bayesian network to be used in order to construct performance profiles, which in turn are used to call the TMDP planner, which selects a stage of automation and thus the interface to the pilot.

- A preliminary pilot model for this system, which allows performance profiles of the pilot to be constructed. The model specifies how an assessment of pilot workload can be used to define performance profiles required in order to plan to issue pilot alerts. It should be noted that this preliminary pilot model contains several parameters which have yet to be assigned value. Constructing a full pilot model with parameters is a large research undertaking and the work provided in this chapter only provides the preliminary foundations. However, it is the goal of this line of research to identify the set of parameters required in order to construct pilot performance profiles, so that they can be assigned values in future research.

This work was produced while at a summer internship at Aptima, Inc., for an NRA (NASA Research Announcement) under the sponsorship of Dr. Kara Latorella at NASA.

It was conducted under the supervision of Dr. Nathan Schurr at Aptima, who first proposed the system architecture shown in Figure 6.1. The planning portion of this system was presented at UAI 2010 by Carlin, Schurr, and Marecki [29]. The pilot state model was first constructed under the supervision of Dr. Amy Alexander as well as Dr. Schurr. The pilot state model was presented by Carlin, Alexander, and Schurr at the Modeling and Simulation World Conference and Expo (ModSim 2010) [28]. Details of the visualization were published by Saffell, Alexander, Carlin, Chang, and Schurr in the International Symposium of Aviation Psychology [112]. The research reported in this chapter is a portion of a larger multi-disciplinary effort. A report of the complete research has been compiled for NASA by Alexander, Saffell, Alaverdi, Carlin, Chang, Durkee, Galster, Geiselman, Latorella, Wickens, and Schurr [2], this chapter summarizes the work presented at UAI and ModSim.

6.1 Background

Next Generation Air Transportation System technologies will introduce new, advanced sensor technologies into the cockpit. With the introduction of such systems, the responsibilities of the pilot and the density of air traffic are both expected to dramatically increase (Joint Planning and Development Office, 2007). As a result, the number of potential hazards and relevant information that must be perceived and processed by the pilot will grow. This information is likely to come from a variety of sources, requiring the pilot to integrate this information in order to evaluate hazard potential. Evaluating hazard potential will depend on the consideration of, and differentiation between, immediate (current) hazards and situations requiring re-planning or coordination (future). It will also require reasoning under uncertainty, as the actual state of the world needs to be reasoned from the hazards, and also a plan needs to be constructed for the pilot and artificial aircraft intelligence to handle the hazards, despite uncertainty as to the effectiveness of each, and temporal uncertainty about the duration required to handle each hazard.

To support these responsibilities, the pilot has a need for an Integrated Alerting and Notification (IAN) system that will monitor multiple sources of information to evaluate hazard potentials, track multiple potential hazards, provide caution/warning/alerting (CWA) notifications and context-relevant decision support to the pilot, and determine the best method of presenting this information to ensure that the information can be viewed and used efficiently and effectively. There are two broad challenges that need to be addressed before an IAN system can become operational. First, existing methods cannot reason under uncertainty about the proposed scale of information and hazards in such a time-critical environment [97, 129]. Specifically, these methods do not provide a robust approach for integrating, interpreting, and providing recommendations that can be generated by the diverse and large set of data expected within the NextGen concept of operations. Second, the interaction between a human pilot and an automated system is complex [46] and also uncertain, and the design of an advanced alerting technology must leverage methods for ensuring effective collaborative performance of the human-system team.

6.2 ALARMS approach

The ALerting And Reasoning Management System (ALARMS) addresses these challenges. The ALARMS approach is shown in Figure 6.1. Hazards exist in the real world, as depicted on the left of the diagram. The sensors on the Flight Deck (current and NextGen) perceive these hazards. In the first phase of the ALARMS effort, the hazards and sensor systems were identified. The results of this effort will be described in the next section. The results of this analysis were used to construct probability tables for a State Estimation Bayesian Network, labeled ”State Estimation” in the Figure

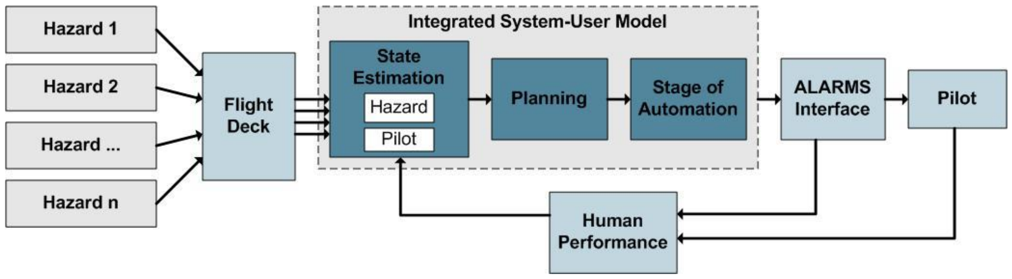


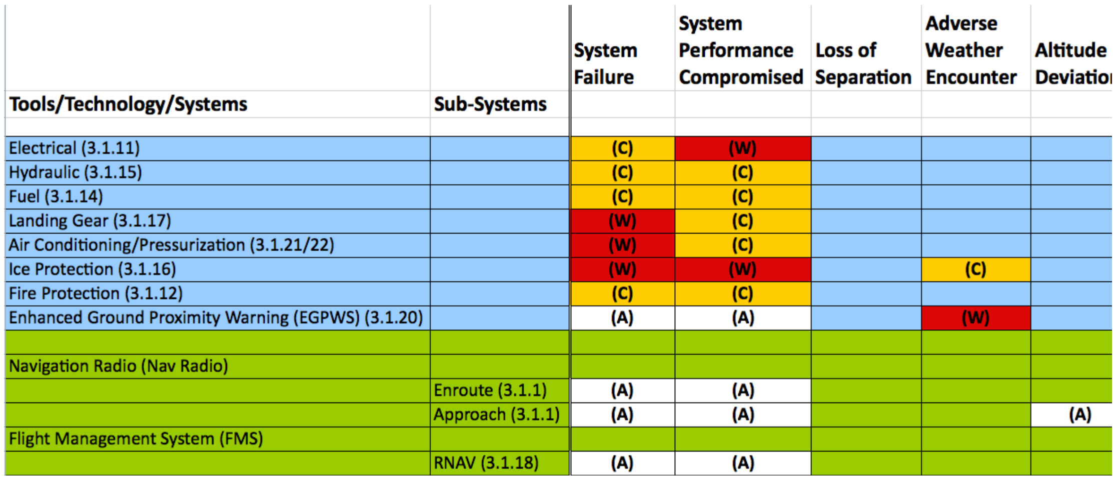
Figure 6.1. The ALARMS approach. Hazards in the environment are detected by aircraft sensors, and pilot state is estimated through Cognitive Work Analysis in the Human Performance module. A Bayesian Network (State Estimation) weights the sensor output to estimate the hazard state. A Planning module forms a time-sensitive plan for the pilot and automated system to address these hazards. The result is a plan with various stages of automation. The ALARMS Interface displays information at the appropriate stage of automation to the pilot.

6.1. The input to the State Estimation Bayesian Network is the alerts issued by the sensor systems on the aircraft. The output is an estimate of the probability and severity estimate of the underlying hazards. Once the hazard and pilot state is estimated, the information is sent to the ALARMS planning module, which will construct a plan to address the hazards. The Planning module is a TMDP (Time-Dependent Markov Decision Processes) [22]. The TMDP model can be used to capture both state uncertainty in the environment as well as duration uncertainty in human (pilot) actions [116]. Its input is the hazard and pilot states, as well as a Markov model of the effectiveness of the pilot and automation in handling the hazards, given various levels of alert. Its output is a time-dependent plan for addressing the alert. The plan is interpreted by the Stages of Automation module, which interprets the level of automation and decides what level of alerts and options to send to the pilot. This decision will then be sent to the ALARMS interface, which displays the information to the pilot.

This chapter summarizes the UAI and ModSim efforts which report on the phase of the ALARMS effort corresponding to the “Integrated System User Model” in Figure 6.1. Other parts of the ALARMS effort are shown in Figure 6.1 as well, and involve modeling and predicting human performance. Through Cognitive Work Analysis (CWA) [24], ALARMS has focused on allowing the system to predict pilot performance in an online fashion. This module is labeled “Pilot State” in the figure. Whereas we will see that the current work involves a primitive model of pilot state which accounts for the phase of flight, in future work the Pilot State Estimate will be constructed using techniques from the cognitive sciences literature, resulting in a richer pilot state space for the Integrated System User Model [73]. Another, orthogonal task, is labeled “ALARMS Interface” in Figure 6.1. In this task, we are leveraging established human factors design principles to develop an interface that (1) maximizes pilot performance in detecting and responding to a diverse set of threats, and (2) is flexible in its integration with existing NextGen features and the concepts of operations. The interface is being designed to support various stages of automation [46]. At low stages, decisions are made almost entirely by the pilot, whereas at high-stages of automation they are made by the system. At medium stages of automation, decisions are cooperative, for example the system may pre-compute several options for complex maneuvers, and let the pilot select from these options.

6.2.1 ALARMS hazard matrix

As the first step in the ALARMS effort, the ALARMS project identified (1) Current and Next Generation aircraft systems. (2) Aircraft Hazards. (3) The interaction between systems and hazards. The ALARMS hazard matrix was constructed as a result of this work, and a small portion of this matrix is shown in Figure 6.2. Each row represents a tool, technology, or system that can issue an alert in the next generation cockpit. Each



Tools/Technology/Systems	Sub-Systems	System Failure	System Performance Compromised	Loss of Separation	Adverse Weather Encounter	Altitude Deviatio
Electrical (3.1.11)		(C)	(W)			
Hydraulic (3.1.15)		(C)	(C)			
Fuel (3.1.14)		(C)	(C)			
Landing Gear (3.1.17)		(W)	(C)			
Air Conditioning/Pressurization (3.1.21/22)		(W)	(C)			
Ice Protection (3.1.16)		(W)	(W)		(C)	
Fire Protection (3.1.12)		(C)	(C)			
Enhanced Ground Proximity Warning (EGPWS) (3.1.20)		(A)	(A)		(W)	
Navigation Radio (Nav Radio)						
	Enroute (3.1.1)	(A)	(A)			
	Approach (3.1.1)	(A)	(A)			(A)
Flight Management System (FMS)						
	RNAV (3.1.18)	(A)	(A)			

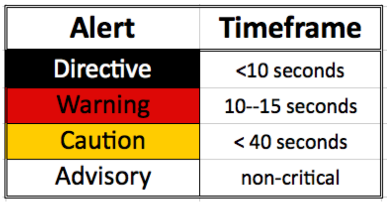
Figure 6.2. A portion of the ALARMS hazard matrix. Each row represents a different sensor subsystem (blue rows are aviation systems, green rows are navigation systems), each column after the Hazards label represents a potential aircraft hazard. Color-coded entries represent the highest urgency alert that the sensor system may issue.

column represents a potential hazard in the environment. Thus, there is an entry for each case where a sensor can issue an alert for a given hazard.

As shown in the figure, entries are also color-coded with the labels D/W/C/A (Directive/Warning/Caution/Advisory). These labels, presented in order of decreasing urgency, represent the highest level of alert possible for the sensor/hazard combination. Thus, entries labeled “A” correspond to systems that will only issue low level advisories for that given hazard, whereas entries labeled “W”, such as an Adverse Weather hazard from the Enhanced Ground Proximity Warning System (EGPWS), correspond to systems that are capable of issuing a more urgent warning. Alert level also corresponds to the timeframe in which the hazard must be addressed, according to Figure 6.3.

6.2.2 ALARMS Bayesian network

The goal of the ALARMS modeling effort relevant to this thesis is to construct a plan for handling sensor alerts; however, it is not truly the sensor alerts that must be handled, it is the underlying hazards which they represent. For example, multiple



Alert	Timeframe
Directive	<10 seconds
Warning	10--15 seconds
Caution	< 40 seconds
Advisory	non-critical

Figure 6.3. Color-coded entries of the hazard matrix represent the level of alert, which in turn corresponds to the timeframe in which that alert must be addressed.

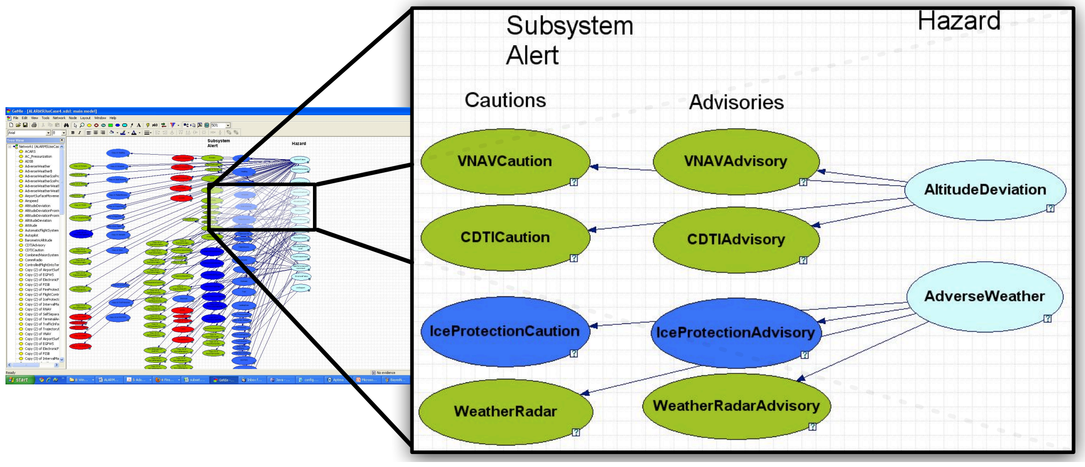


Figure 6.4. The ALARMS bayesian Network.

subsystems (such as Vertical Navigation (VNAV) and Cockpit Display of Traffic Information (CDTI)) can issue alerts which relate to an Altitude Deviation hazard. The ALARMS system should deduce that there may be an Altitude Deviation hazard if either of these sensors issue an alert, and the hazard should be more certain and urgent if both systems issue one.

In order to model the sensor systems, a Bayesian network was built, as shown in Figure 6.4. Ovals on the right represent hazards, corresponding to the columns of the ALARMS hazard matrix. Ovals on the left represent hazard alerts from sensors, corresponding to the entries in the hazard matrix. Directionality proceeds from right to left, indicating that hazards cause sensor alerts. However reasoning proceeds from left to right, the sensor output will act as evidence and the hazard level on the right is deduced. Each entry on both sides can exist at several levels (Advisory, Caution, Warning, Directive), according to the severity of the hazard and the sensor alert.

To construct the network, the GeNIE (Graphical Network Interface) and SMILE (Structural Modeling, Inference, and Learning Engine) software packages were used, from the Decision Systems Laboratory at the University of Pittsburgh [43].

6.2.3 TMDP planner

We now recall the TMDP model and then show how the ALARMS planning module employs it. Time Dependent Markov Decision Processes (TMDPs) [22] assume a finite set S of discrete states and a finite set A of actions. When action $a \in A$ is executed in state $s \in S$, the process transitions with probability $P^{s,a}(s')$ to some state $s' \in S$. The transition itself is not instantaneous; it consumes t units of time with probability $d^{s,a}_{s'}(t)$ where $d^{s,a}_{s'} \in D$ is a probability density function (a.k.a. *action duration distribution*) for a given s, a, s'. Similarly, the reward $R^{s,a}_{s'}(t)$ that the transition provides depends on s, a, s' as well as on the time t at which the process enters state s'. (Note, that t in

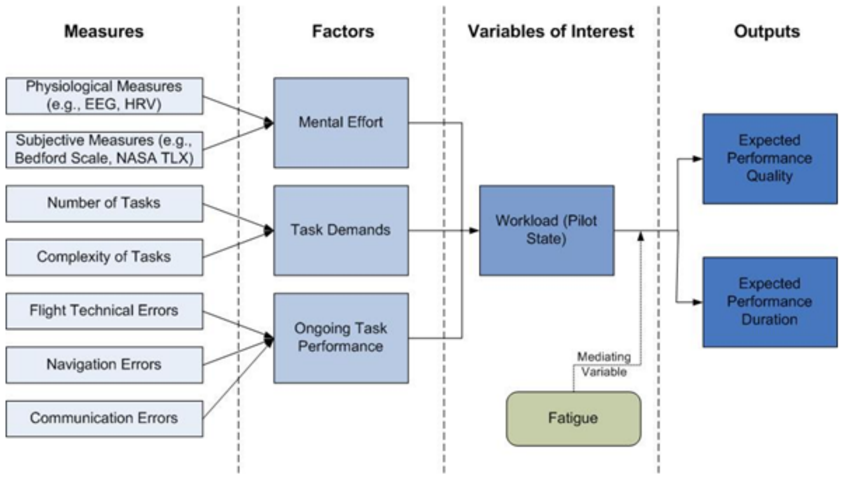


Figure 6.5. The pilot state model

$d_{s'}^{s;a}(t)$ is the transition duration whereas t in $R_{s'}^{s;a}(t)$ is the time at which the transition terminates.) A deterministic TMDP policy π is therefore a mapping $S \times [0, \Delta] \to A$ where Δ (a.k.a. *the deadline*) is the earliest point in time after which all the rewards $R_{s'}^{s;a}(t)$ are zero. Denote by $V^{\pi}(s,t)$ the total expected reward for following a policy π from state s at time t. (For a given s, $V^{\pi}(s,t)$ is often viewed as a continuous *value function* over $t \in [0, \Delta]$.) The optimal TMDP policy π^* thus satisfies $V^{\pi^*}(s,t) \geq V^{\pi}(s,t)$ for all $s \in S$, $t \in [0, \Delta]$ and $\pi \neq \pi^*$.

Let Ψ be the set of alert levels (e.g. “Nominal” (N), “Advisory” (A), “Caution” (C), “Warning” (W), or “Directive” (D)), Φ be an ordered set of hazards (e.g. “Weather” (hazard 1), “Altitude Deviation” (hazard 2)) and Ω be a set of autonomy levels (e.g. “No Autonomy” (0), “Some Autonomy” (1), or “Full Autonomy” (2)). A TMDP in ALARMS planning module is instantiated as follows:

- **States**: A state $s \in S$ is a mapping from the hazards to their alert levels. That is, $s = (\psi_{\phi})_{\phi \in \Phi}$ is a vector where $\psi_{\phi} \in \Psi$ is the alert level of hazard $\phi \in \Phi$. For example, given three hazards, state $s = (N, A, W)$ defines that the first hazard is at Nominal level, the second hazard is at Advisory level, and the third hazard is at Warning Level.

- **Actions**: The actions of the ALARMS system represent the different ways in which the system displays the information about the hazards on the pilot's GUI. In general, the higher the degree of autonomy $\omega_\phi \in \Omega$ for a hazard $\phi \in \Phi$, the less intrusive the way in which the information about hazard ϕ is presented on the pilot's GUI. An action $a \in A$ is therefore represented by a vector $(\omega_\phi)_{\phi \in \Phi}$. For example, given three hazards, action $a = (1, 3, 2)$ will mark on the pilot's GUI the information about hazard 1 to have high importance (autonomy level 1), the information about hazard 2 to have no importance (autonomy level 3) and the information about hazard 3 to have some importance (autonomy level 2). There is also a special autonomy level 0 reserved for actions that do not address the hazard at all (the pilot is not informed about a hazard and the automation does not address it).

- **Transitions**: ALARMS assumes that all the hazards will eventually be addressed (their alert levels will return to "Nominal" (N) values) as a result of human or autonomy actions. That is, for all states $s \in S$ and actions $a \in A$, $P^{s,a}(s') = 1$ only for $s' = (\psi_\phi)_{\phi \in \Phi}$ such that $\psi_\phi = N$ for all $\phi \in \Phi$. An exception to the above is when the action is not to address the hazard, in which case the state remains the same.

- **Durations**: ALARMS models action duration distributions by assuming that actions at a high level of automation take place quickly whereas actions at a low level of automation (i.e. that involve the pilot) have a longer duration. In essence, an attentive pilot will be more efficient at addressing hazards whereas an inattentive or overburdened pilot will perform poorly. As part of the ALARMS effort, profiles of pilot performance have been constructed at various phases of flight.

The phase of flight affects pilot attentiveness, which in turn affects the action duration distributions. In the next phase of the ALARMS project, we expect to construct richer models of pilot state, based on Cognitive Work Analysis (CWA).

- **Rewards**: Reward is achieved for addressing the hazard and transitioning back to a nominal state. (Each hazard can have a different reward associated with it.) Actions with a low level of automation (hazards handled by the pilot) accumulate greater reward, whereas actions taken with a higher level of automation (handled by the system, without pilot feedback) achieve a lower level of reward. As actions that provide higher rewards usually take longer to execute, given a time deadline Δ after which no rewards can be earned, an optimal TMDP policy must often trade-off high reward actions for their faster, but lower reward counterparts. We illustrate these trade-offs in the next Section, where optimal TMDP policies are found using the CPH algorithm [78].

6.3 Application

We constructed an analysis tool to allow the flight deck designer to understand the behavior of the flight deck for different hazard and pilot states. A picture of the tool can be seen in Figure 6.6. In the example shown, four sensor systems are considered (Weather Radar, Ice Protection, VNAV, and CDTI) which provide alerts. Options for the level of alert (Directive/Warning/Caution/Advisory/None) are configurable through an .xml file. The systems are run through the Bayesian Network to determine the underlying hazard state. Adjustable sliders allow the user to configure the rewards for addressing each hazard. (Since the TMDP reward function is defined using rewards for addressing the actual underlying hazards, instead of the sensor system rewards, the reward for the hazard is merely the maximum of the connected sensor system rewards). Phase of Flight can also be selected to determine the pilot state. Clicking the "Run" button induces the program to perform two actions: (1) It reads in the Bayes Net (in .xml format), and finds the underlying hazards by using the user-selected GUI options as evidence and (2) It builds and solves the underlying TMDP to find an optimal plan that addresses the hazards, and plots the results.

For our example problems containing four sensors and two hazards, the Bayesian computation took less than a second, and the CPH solver found the optimal solution in 40.5 seconds. For aircraft deployment, we anticipate producing the TMDP policies for hazard combinations in advance, and at flight time implementing the resulting policies through a lookup table.

Figure 6.6 displays the policies for various hazard and pilot states. The horizontal axes in all the plots mark the time to a deadline (the elapsed time can be viewed as proceeding from right to left) and the vertical axes represents the expected value (=sum of expected rewards). Differing actions are plotted in different colors (or can be seen as a break in the graph for those who read this in black and white). The action with the largest expected value for a given point in time should be executed if a decision is to be made at that point in time.

Consider the first plot in Figure 6.6, with two hazards to be addressed: hazard 1 = "Weather" and hazard 2 = "Altitude Deviation". The TMDP state considered in this plot is (A, N) which means that the "Weather" hazard is in Advisory mode and the "Altitude Deviation" hazard is in Nominal mode. As can be seen, when the deadline is > 5 seconds away, action "L10" (shorthand for "Handle Hazard 1 at automation level 1, Handle Hazard 2 at automation level 0") is selected, the pilot is expected to handle the "Weather" hazard with some importance and the "Altitude Deviation" hazard with no importance. However, as we approach the deadline, action "L20" is more preferable, as automation level "2" is expected to act more quickly, thus potentially providing reward more quickly. (Note that only the upper envelope, that is the parts of the value functions that are not dominated by other value functions, is shown in the Figure.)

Subsequent plots in Figure 6.7 show how the optimal TMDP policy changes when we consider different states. Figure 6.7a shows the optimal policy for a state (C, A). Notice, that when the deadline is far away, both hazards are assigned to the pilot (automation level 1, the portion of the curve shown in green). However, as the deadline nears, the system deduces that there is not sufficient time for the pilot to handle both hazards. Consequently, the system assigns the less severe hazard (“Altitude Deviation”) to the automation, shown as the purple portion of the curve on the left of the graph. Conversely in (Figure 6.7b), when we set a higher relative reward for addressing the “Altitude Deviation” hazard (seen on the number next to the slider bars on the right), as the deadline approaches, the system makes the opposite decision. It assigns the less prioritized (rewarded) hazard (“Weather”) to the automation. On the right of the application, under “Actions”, we can see that the purple left-portion of the plot now represents “L21” instead of “L12”. Finally, the last plot in Figure 6.7 shows the effects of changing the phase of flight on the TMDP policy. By changing the phase of flight from “Enroute” to “Land”, the assumptions about the pilot state have changed as well. During the Landing phase, the pilot is assumed to be less efficient than usual at performing tasks, and thus the value functions are shifted to the right, reflecting the fact that the action with the higher level of automation is favored farther away from the deadline than in the graph above it.

6.4 Pilot state model

We model human performance as shown in Figure 6.5. The Output from the module is an estimate of expected pilot performance, in terms of duration and quality of pilot handling of hazards. The Variable of Interest, Workload, is the key parameter representative of pilot state that changes over time and directly impacts performance. The Mediating Variable, Fatigue, influences the relationship. Other variables of interest (e.g., situation awareness) or mediating variables may be considered in future work. In environments with task demands, workload affects the mental resources that a pilot can access to address the demands[148]. Specifically, the effect can be modeled through a performance resource function, or PRF [90]. When cognitive resources are unavailable or unused for a task, performance will be diminished. As more resources are dedicated, performance will improve, until the task becomes limited by data and not resources. When multiple tasks must be accomplished, such as is the case when a pilot must supervise multiple systems in the cockpit, resource limitation becomes an issue [67]. The workload of the pilot will define the availability of a pilot's resources to handle alerts. It is possible to assess workload as an index, and several criteria have been specified to compute the index [148, 87]. Among these criteria: a satisfactory workload index is sensitive to changes in task demands, diagnoses the cause of workload variation, is selective in that factors that do not affect workload are not included in the index, is unobtrusive in that the computation of the index does not affect workload itself, and is reliable. For ALARMS, we identify three factors that predict workload: mental effort, task demands, and ongoing task performance. We also identify relevant measures of these factors from the literature.

6.4.1 Mental effort

We follow the literature by specifying mental effort as a contributing factor to workload. High levels of performance can be achieved under conditions of normal mental effort while extremely high mental effort situations tend to result in decreased performance. Measures of mental effort include both subjective and physiological measures [144]. Subjective information in our model includes potential measures such as the

NASA TLX scale [56], which allows the operator to specify mental demand, physical demand, temporal demand, performance, effort, and frustration level. The Bedford Workload scale (Roscoe, 1984), on the other hand, is a decision tree, and the leaves of the tree provide a workload score on a single dimension. Physiological information can also be obtained. Examples of potential measures include electroencephalography (EEG) or heart rate variability (HRV). It has been shown that heart rate can differentiate between phases of flight (which require different levels of mental effort) for pilots and co-pilots [21], even when subjective measurements do not.

6.4.2 Task demands

In the prior subsection, mental effort is described as being necessary to accomplish tasks. The level of effort demanded will depend on the task. Simple tasks will require smaller amounts of resources, while complex tasks will require a higher degree of mental effort. Measures of Task Demands include both the complexity of tasks and the number of tasks. Task complexity can affect workload; specifically, complex tasks will result in a higher workload. For example, the landing phase of flight produces higher workload than the Enroute phase [21]. As a second example, more automated tasks consume fewer resources than less automated ones [115]. Number of tasks affects workload as well, in two ways. First, the presence of additional tasks adds to workload. Second, there is a cost to switching among tasks [105]. Thus, the contribution of tasks to workload exceeds the sum of the tasks complexities.

6.4.3 Ongoing task performance

Workload contributes to the model insofar as it is predictive of pilot performance. Thus, a well-accepted manner of estimating workload is to examine performance directly. Potential measurements include Flight Technical Errors, Navigation Errors, and

Communication Errors. These errors can be measured by the ALARMS system at run-time.

6.4.4 Interface to ALARMS planner

As shown in Figure 2, Workload affects the duration and quality of pilot actions in the ALARMS model. This is accomplished by performing a two step process. First, a workload score is computed from measurements of factors. This is accomplished through a linear weighting of the factors: Workload = $\alpha(ME) + \beta(TD) + \gamma(TP)$ where ME represents Mental Effort, TD represents Task Demands, and TP represents Task Performance. α, β, and γ represent linear weights that allow the prioritization of the factors to be varied. In the second step, the workload score is used to modify the Duration and Reward function of the ALARMS TMDP. We use the Workload estimate to feed information into the Integrated User Module about the expected capabilities of the pilot, specifically the expected performance quality and the expected duration of pilot actions. The effect of Workload varies according to the stage of automation. In Stage 1 of automation, increasing workload in our model will greatly decrease quality and increase duration for high workload conditions as compared to low workload conditions. In Stage 2, increasing workload will decrease quality and increase duration. In Stage 3, we make the effects negligible. The specific quantities attached to these effects are parameters in our model. At present, we set quality and duration to halve and double, respectively, in Stage 1, when workload is changed from Low to High. Similarly we set quality and duration to decrease and increase 25 percent in Stage 2, and to decrease and increase 5 percent in Stage 3. Medium workload is currently simulated by interpolating between the high and low workload conditions.

6.5 Related work

There are many works on the use of Bayesian networks for diagnosis [8, 23]. The idea of a time-dependent Markov model was first mentioned in [22]. Progress towards solving these problems in fast time was made in more recent publications [75, 78].

Systems where MDPs or POMDPs were used for adjustable autonomy include [114, 142]. These works did not allow for error checking or further consideration once an assignment was made. Furthermore, the state space was very large, not taking advantage of the TMDP framework, resulting in thousands of states for similarly sized problems.

As the cockpit has grown more complicated, numerous works have been published studying the effects on pilots. Cognitive Work Analysis has been used in order to model the effects of varying system states on pilot workload, and the effects of workload on performance [24]. Much work attempts to study the interaction between the human pilot and the automated system and include the levels of automation concept [46, 91]. However, these works have not produced algorithms, whereby the assignment of tasks to pilot and human were varied automatically in a planned manner.

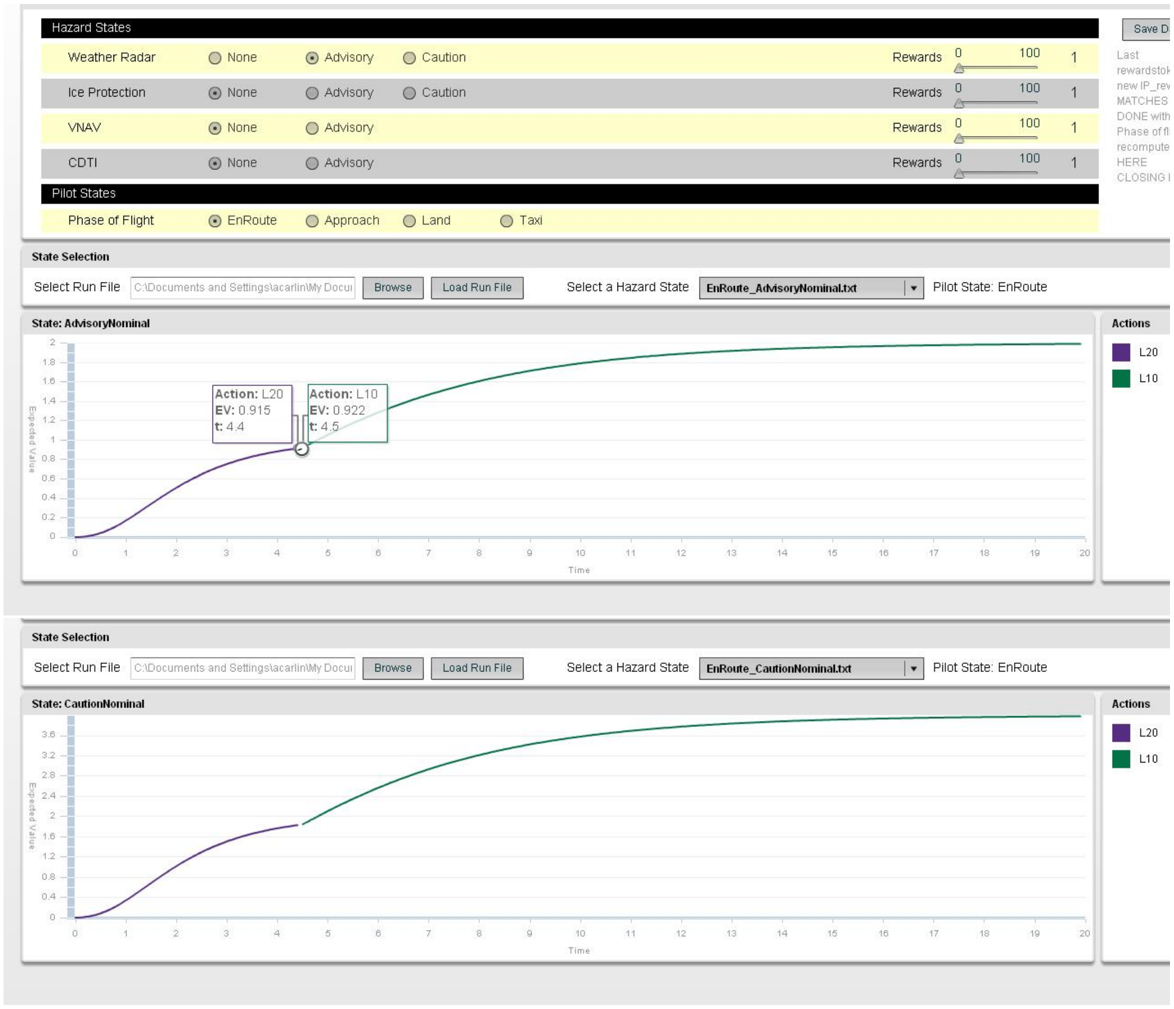
6.6 Acknowledgements

I would like to thank Aptima and NASA for presenting me with this research topic. Also I would like to thank Scott Galster for aiding in the construction of the ALARMS Hazard matrix presented earlier in this chapter, Nathan Schurr for developing the general architecture, and Gilbert Mizrahi for developing the graphical user interface of the ALARMS application.

6.7 Conclusion

In the ALARMS project for NASA, we have developed several components necessary for operation of aircraft in a NextGen environment. First, a study of existing systems was conducted, and a matrix correlating aircraft sensor output (both legacy and NextGen) with real-world hazards was constructed. Second, this matrix was used to create a Bayesian Network whereby sensor output becomes evidence, and the presence and severity of real-world hazards is derived. Third, a TMDP model was created; allowing the aircraft sensor system to select the appropriate level of automation which best addresses hazards in a time-dependent environment. Finally, a demonstration application was created, linking the applications and thereby producing automation plans directly from the simulated sensor output.

Future work will continue in several directions. First, TMDP models will be scaled to handle not just the use cases from the demonstration application, but the whole Bayesian network. Second, work from the cognitive sciences literature will be leveraged to better estimate the pilot state. Finally, we will develop a user interface based on levels of automation. Through the combined efforts of hazard state estimation, pilot state estimation, and human-automation planning, it is our hope to provide a robust, smooth transition to the Next Generation aircraft cockpit.



Hazard States

Weather Radar	None	[✓ Advisory]	Caution	Rewards	0 — 100	1
Ice Protection	[✓ None]	Advisory	Caution	Rewards	0 — 100	1
VNAV	[✓ None]	Advisory		Rewards	0 — 100	1
CDTI	[✓ None]	Advisory		Rewards	0 — 100	1

Pilot States

Phase of Flight	[✓ EnRoute]	Approach	Land	Taxi

```
Last
rewardstok
new IP_rev
MATCHES
DONE with
Phase of fl
recompute
HERE
CLOSING I
```

State Selection

Select Run File: C:\Documents and Settings\acarlin\My Docu | Browse | Load Run File | Select a Hazard State: EnRoute_AdvisoryNominal.txt | Pilot State: EnRoute

State: AdvisoryNominal

Action: L20
EV: 0.915
t: 4.4

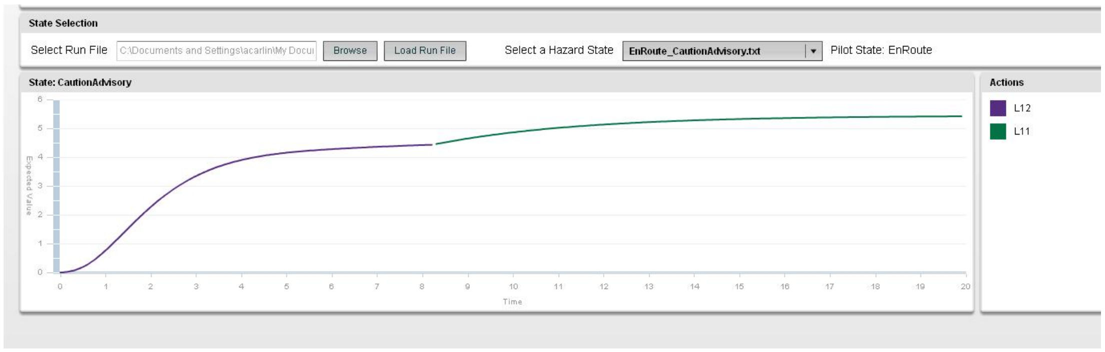
Action: L10
EV: 0.922
t: 4.5

State Selection

Select Run File: C:\Documents and Settings\acarlin\My Docu | Browse | Load Run File | Select a Hazard State: EnRoute_CautionNominal.txt | Pilot State: EnRoute

State: CautionNominal

Figure 6.6. The ALARMS application. The top plot shows an Advisory for one hazard, the bottom plot was generated by changing the Advisory to a Caution (resulting in a doubling/rescaling of the y-axis).


State Selection

Select Run File: C:\Documents and Settings\acarlin\My Docu

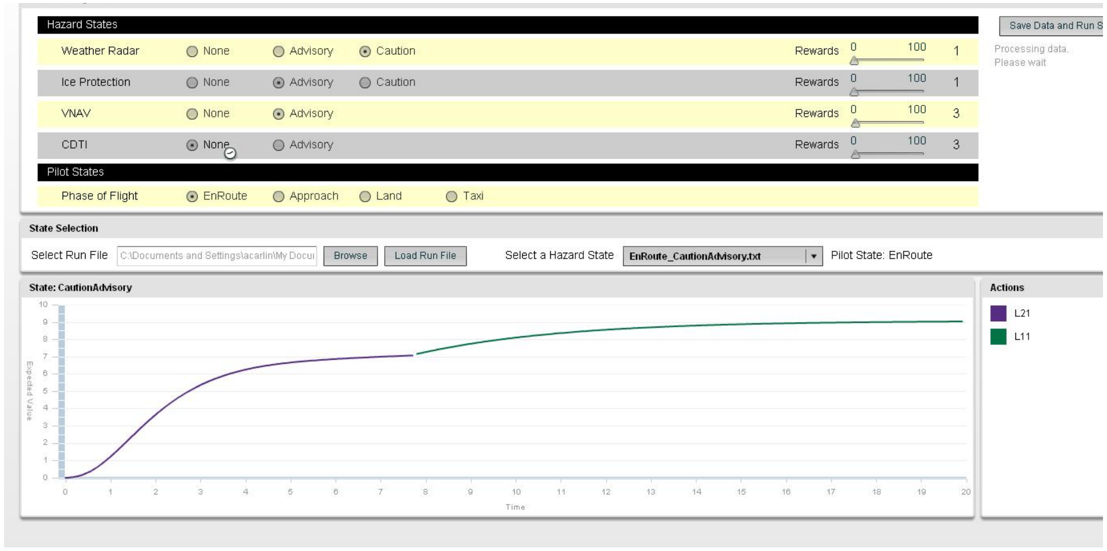
Select a Hazard State: EnRoute_CautionAdvisory.txt

Pilot State: EnRoute

State: CautionAdvisory

Actions: 🟪 L12, 🟩 L11

(a) Two hazards. The second hazard is handled at a higher level of automation, close to the deadline.


Hazard States

Weather Radar	None	Advisory	[✓ Caution]	Rewards 0–100	1
Ice Protection	None	[✓ Advisory]	Caution	Rewards 0–100	1
VNAV	None	[✓ Advisory]		Rewards 0–100	3
CDTI	[✓ None]	Advisory		Rewards 0–100	3

Pilot States

Phase of Flight: [✓ EnRoute] Approach Land Taxi

Processing data. Please wait

State Selection

Select Run File: C:\Documents and Settings\acarlin\My Docu

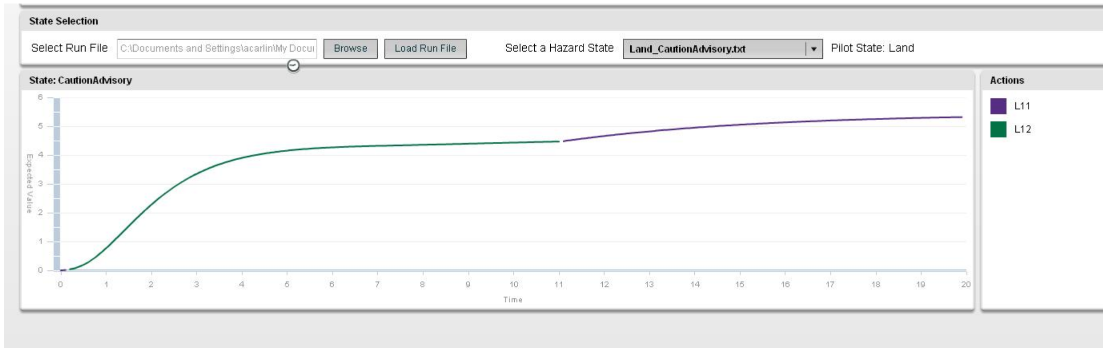
Select a Hazard State: EnRoute_CautionAdvisory.txt

Pilot State: EnRoute

State: CautionAdvisory

Actions: 🟪 L21, 🟩 L11

(b) The second hazard reward is increased via the values next to the slider bars, thus now it is handled by the pilot.


State Selection

Select Run File: C:\Documents and Settings\acarlin\My Docu

Select a Hazard State: Land_CautionAdvisory.txt

Pilot State: Land

State: CautionAdvisory

Actions: 🟪 L11, 🟩 L12

(c) Phase of flight is changed to land. The pilot's attention is diverted and the graph is shifted right.

Figure 6.7. The ALARMS application

CHAPTER 7
CONCLUSION

Meta-reasoning continues to be a necessary area of research which is applicable to any situation in which the cognitive resources available to a decision maker are limited. This thesis focuses on two particular limitations, time and space. Decentralized meta-reasoning is necessary when the meta-reasoning process does not occur in one single location. This thesis has examined the monitoring and control of base-level algorithms, including when to stop one type of computation and begin another. A decision-theoretic approach is taken, throughout this thesis an expected value is derived for each computation, and the computation with the largest expected value is selected. For decentralized meta-reasoning, a joint or non-myopic computation plan is constructed, as opposed to a myopic plan which considers the point of view of only one decision-maker. The meta-reasoning decisions are applied under the Dec-MDP, Dec-POMDP, and Dec-POMDP-Comm models.

7.1 Summary of contributions

Chapter 3 considered the decentralized monitoring and control of separate base-level algorithms, each with its own performance profile. It expanded on previous work in the literature which had considered the monitoring and control of a single algorithm. First, the problem of decentralized monitoring and control was formalized into the Decentralized monitoring problem (DMP) and model. The model specifically represented stopping and monitoring decisions on the part of each meta-reasoning agent. Variants of the model were constructed which allowed for either local monitoring of the current agent or global monitoring of all agents. For local monitoring, a complexity analysis was conducted and the problem of making joint stopping decisions was shown to be NP-complete through reduction to a transition-independent Dec-MDP model. The reduction was used to construct a bilinear programming problem from the DMP, thus allowing solutions to the bilinear program to apply to the DMP. For global monitoring, the problem was reduced to a Dec-MDP with communication. Experiments were conducted for each variant, to demonstrate how the model can be used to make stopping and monitoring decisions for base-level algorithms. An empirical evaluation was conducted in order to evaluate myopic versus coordinated reasoning techniques. Both techniques were shown to be effective, with the coordinated technique being more effective in cases where the decision of whether to stop or not (or monitor or not) was particularly close in expected value, as well as in cases where the contributions of base-level algorithms to overall utility were asymmetric.

Chapter 4 constructed meta-reasoning capabilities for the Dec-POMDP model, in which agents execute in distributed fashion but a centralized algorithm produces execution policies at planning time. A method of observation compression was introduced which allows a planner to cluster observations together, saving time and space. Observation compression was produced in a principled manner; observations were grouped together based on the worst-case expected value loss incurred due to clustering. This allowed algorithms to be developed for both lossless compression as well as lossy compression with bounds. The observation compression method was combined with point-based decentralized planning algorithms, producing a more efficient algorithm. The resulting algorithm was analyzed for complexity in time and space, as well as loss bounds, which can be computed online. Using these bounds, a planner that conducts meta-reasoning over the online bounds was produced. After the original development of observation compression, more recent algorithms have either included their own form of compression or can be augmented by observation compression if desired.

Chapter 5 examined meta-reasoning in both Dec-MDP and Dec-POMDP models with communication. Unlike the model in Chapter 3, this model allowed for full synchronization of state (for Dec-MDP) and belief state (for Dec-POMDP). For both models, a myopic algorithm was first developed which computes a value of communication for each agent. These algorithms were evaluated, and it was shown that they could result in overcommunication. Two improvements were made. First, agents were augmented to construct joint communication policies rather than single-agent communication policies. Second, agents computed the value of delaying communication. It was shown that these two improvements could mitigate the overcommunication problem.

Chapter 6 briefly describes a real-world application of meta-reasoning. In the application, one of the base-level decision-makers was a human being, specifically an airline pilot addressing external hazards. Meta-reasoning took place on the proposed cockpit automation; this automation would evaluate its own performance profile for handling the hazards and compare it to the human pilot's performance profile under varying stages of automation. A TMDP model was constructed to optimize joint pilot-machine performance, the input to the model was the performance profiles and the output was a plan for issuing alerts at the appropriate stage of automation over time. In order to construct the performance profile of the pilot, a pilot state model was proposed.

7.2 Final thoughts

Before discussing future directions, I would like to discuss thoughts of future obstacles. Both have to do with balance.

First: a year before this thesis was written, two definitions of meta-reasoning were considered, which I roughly consider to be the "tight" and "loose" definitions. The tight definition corresponds to Chapter 3. It involves a well-defined procedure of forming performance profiles and using decision theory to reason about them. The POMDP-based models discussed in this thesis (POMDP, Dec-POMDP, etc.) would have all been used to construct meta-reasoning policies and never base-level policies, unlike Chapters 4 and 5. Dec-POMDP especially would have been applied to elaborate, non-independent performance profiles. The loose definition was the one adopted for this thesis, and is shown in Table 1.4. Dec-POMDP planning was not used as a meta-reasoning method improved by observation compression, rather Dec-POMDP planning was the base-level computation and observation compression itself was the meta-reasoning method. Similarly, making non-myopic communication decisions was depicted as a meta-reasoning method.

Second: One item that may not come through in the text is how powerful and effective the myopic solutions were. In the experiments in Chapter 3, the myopic solutions were effective except when the decisions were close. But most decisions are not close, for example it was very clear in most experiments in Chapter 3 when to stop, and when to continue. In the experiments of Chapter 5, the myopic strategies would often find very clever (but myopic) ways to produce near-optimal strategies.

An open empirical question is, in my mind, whether the cleverness of the myopic strategies scales. Or whether it is the case that as problems become more complex, the myopic strategies fail. I did not find this to be the case in the experiments conducted for this thesis. As a matter of fact, as problems became more complex, the complex factors would roughly "offset", and a quick and dirty method such as the myopic strategies seemed to suffice.

An open literature question is, in my mind, whether the tight or loose definition of meta-reasoning scales better. The advantage of the tight definition is that it can define a set procedure, one can imagine a "performance-profile file format" someday to standardize meta-reasoning software. The advantage of the loose definition is that it is more applicable to more domains.

7.3 Future directions

I would like to take this work in two directions in the future. First, I would apply the theory of Chapter 3 to more complex distributed applications. The challenge in doing this lies in identifying useful domains in which such monitoring and control is applicable. Augmenting the theory of Chapter 3 in order to address the compilation problem in Section 2.1.4.2 will augment this work by allowing it to address problems where the output of one module links to the input of a separate module. The techniques shown in Chapter 3 seem like a natural combination for a Hadoop framework and distributed data mining base-level algorithms.

Specifically, the fulfillment of NASA's System-Wide Safety and Assurance Technology (SSAT) project at NASA requires leveraging vast amounts of data into actionable knowledge. Models such as the Accident Causation Model describe active errors, latent errors, windows of opportunity, and a causation chain. Each of these concepts would be better understood by examining the large amounts of "everyday" flight data, and not just the small amount of data proximal to high-profile incidents. Analysis of a larger data set of data would provide more examples and thus better opportunity for machine learning algorithms to reach statistically significant conclusions. The specific data mining algorithm will depend on the data sets that are available. One data set which may be available is in the aviation domain and involves FOQA (Flight Operations Quality Assurance) data, which is maintained in separate databases by separate airlines and comprehensively records flight data from instruments. Using this data, I hope to find or confirm previously unknown correlations between the distributed data (e.g., "aircraft that fly through certain weather experience more part failures within the next year", "flight crews who are overworked tend to miss more landings", etc.). FOQA data sets are (1) large, (2) distributed, and (3) heterogeneous, making analysis difficult, and making them a potential candidate for distributed meta-reasoning.

A second area that I'd like to continue is in the mixed initiative planning in Chapter 6. The model depicted in that chapter is high level, and many of the parameters need definition. There are three areas that need further work. First, the pilot state model needs to be further defined, so that its input is unobtrusive measurements and its output is a performance profile of time and quality distribution. The second is a study of how separate stages of automation can affect the performance profile. I hope to work with cognitive scientists in the near future to achieve these augmentations. Third, the hazards should be addressed one by one, and a model should be defined which inputs sensor information and outputs a level of alert.

BIBLIOGRAPHY

[1] S. Abdallah and V. Lesser. Learning the task allocation game. In *AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[2] A. Alexander, T. Saffell, O. Alaverdi, A. Carlin, A. Chang, K. Durkee, S. Galster, E. Geiselman, K. Latorella, C. Wickens, and N. Schurr. *ALARMS: ALerting and Reasoning Management System*, NASA, under review.

[3] M. Allen and S. Zilberstein. Agent influence and intelligent approximation in multiagent problems. In *IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 1988.

[4] C. Amato, A. Carlin, and S. Zilberstein. Bounded dynamic programming for decentralized POMDPs. In *MSDM '07: AAMAS Workshop on Multi-agent Sequential Decision Making in Uncertain Domains*, 2007.

[5] C. Amato, J. Dibangoye, and S. Zilberstein. Incremental policy generation for finite horizon Dec-POMDPs. In *ICAPS '09: Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

[6] B. An, V. Lesser, D. Westbrook, and M. Zink. Agent-mediated multi-step optimization for resource allocation in distributed sensor networks. In *AAMAS '11: Proceedings of the 10th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2011.

[7] M. Anderson. A review of recent research in metareasoning and metalearning. *AI Magazine*, 28, 7–16, 2007.

[8] S. Andreassen, M. Woldbye, B. Falck, and S. Andersen. MUNIN: A causal probabilistic network for interpretation of electromyographic findings. In *IJCAI '87: Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 1987.

[9] R. Aras, A. Dutech, and F. Charpillet. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *ICAPS '07: Proceedings. of the International Conference on Automated Planning and Scheduling*, 2007.

[10] K. Aström. Optimal control of Markov decision processes with incomplete state estimation. In *Journal of Mathematical Analysis and Applications*, 10, 174–205, 1965.

[11] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving transition-independent decentralized Markov Decision Processes. In *Journal of Artificial Intelligence Research*, 22, 423–455, 2004.

[12] R. Becker, V. Lesser, and S. Zilberstein. Analyzing myopic approaches for multi-agent communication. In *IAT '05: Proceedings of Intelligent Agent Technology*, 2005.

[13] R. Becker, A. Carlin, V. Lesser, and S. Zilberstein. Analyzing myopic approaches for multi-agent communication. In *Computational Intelligence*, 25, 2009.

[14] R. Bellman. A Markovian Decision Process. In *Journal of Mathematics and Mechanics*, 6, 1957.

[15] D. Bernstein, E. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *IJCAI '05: Proceedings of the Nineteenth International Joint Conf. on Artificial Intelligence*, 2005.

[16] D. Bernstein, C. Amato, E. Hansen, and S. Zilberstein. Policy iteration for decentralized control of Markov Decision Processes. In *Journal of Artificial Intelligence Research*, 34, 89–132, 2009.

[17] D. Bernstein, R. Given, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. In *Mathematics of Operations Research*, 819–840, 2002.

[18] D. Bertsimas and S. Patterson. The air traffic flow management problem with enroute capacities. In *Operations Research*, 46(3), 406–422, 1998.

[19] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. In *Artificial Intelligence*, 67, 244–285, 1994.

[20] M. Boddy, B. Horling, J. Phelps, R. Goldman, R. Vincent, A. Long, and B. Kohout. *Ctaems language specification*. v. 1.06, 2005.

[21] M. Bonner and G. Wilson. Heart rate measurements of flight test and evaluation. In *International Journal of Aviation Psychology*, 12, 63-77, 2002.

[22] J. Boyan and M. Littman. Exact solutions to time-dependent MDPs. In *NIPS '00: Proceedings of Neural Information Processing Systems*, 1026–1032, 2000.

[23] J. Breese, E. Horvitz, M. Peot, R. Gay, and G. Quentin. Automated decision-analytic diagnosis of thermal performance in gas turbines. In *Proceedings of the International Gas Turbine and Aeroengine Congress and Exposition*. American Society of Mechanical Engineers, 1992.

[24] T. Callantine, C. Mitchell, and E. Palmer. Tracking operator activities in complex systems: An experimental evaluation using Boeing 757 pilots. In *Proceedings of Ninth International Symposium on Aviation Psychology*, 1997.

[25] A. Carlin, J. Rousseau, J. Ayers, and N. Schurr. Agent-based coordination of human-multirobot teams in complex environments. In *AAMAS '10: Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2010.

[26] A. Carlin and S. Zilberstein. Value-based observation compression for Dec-POMDPs. In *AAMAS '08: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[27] A. Carlin and S. Zilberstein. Myopic and non-myopic communication under partial observability. In *IAT '09: IEEE Conference on Intelligent Agent Technology*, 2009.

[28] A. Carlin, A. Alexander, and N. Schurr. Modeling pilot state in aircraft alert systems. In *MODSIM World 2010 Conference and Expo*, 2010.

[29] A. Carlin, N. Schurr, and J. Marecki. ALARMS: Alerting and Reasoning Management System for Next Generation Aircraft Hazards. In *UAI '10: Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, 2010.

[30] A. Carlin. Solving the parking problem in Vehicular Ad-hoc Networks (VANETS) with Decentralized POMDPs. Synthesis project for University of Massachusetts Computer Science, unpublished, 2008.

[31] A. Carlin and S. Zilberstein. POMDP and Dec-POMDP point-based observation aggregation. In *AAAI Workshop on Advancements in POMDP Solvers*, 2008.

[32] A. Carlin and S. Zilberstein. Decentralized monitoring of distributed anytime algorithms. In *AAMAS '11: Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2011.

[33] A. Carlin and S. Zilberstein. Bounded rationality in multiagent systems using decentralized metareasoning. In *Decision Making with Imperfect Decision Makers*, T. Guy, K. Valentine, and D. Wolpert (eds), Springer, 2011.

[34] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, 1427–1429, 2006.

[35] S. Cheng, A. Raja, and V. Lesser. Multiagent meta-level control for a network of weather radars. In *IAT '10: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, 157–164, 2010.

[36] C. Cherniak. *Minimal Rationality*, MIT Press: Cambridge, 1986.

[37] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, 2001.

[38] G. Cooper Probabilistic inference using belief networks is NP-hard. In *Artificial Intelligence*, 42, 393–405.

[39] M. Cox and A. Raja (eds). *Metareasoning: Thinking about thinking*. MIT Press, Cambridge, MA, 2011.

[40] T. Dean and M. Boddy. An analysis of time-dependent planning. In *AAAI '88: Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[41] K. Decker. TAEMS: a framework for environment centered analysis and design of coordination mechanisms, chapter 16. In *Foundations of Distributed Artificial Intelligence*, G. Hare and N. Jennings (eds.), 429–448. Wiley Inter-Science, 1996.

[42] J. Dibangoye, A. Mouaddib, and B. Chai-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *AAMAS '09: Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2009.

[43] M. Druzdzel. SMILE: Structural Modeling, Inference and Learning Engine and GeNIE: a development environment for graphical decision-theoretic models. In *AAAI '99: Proceedings of the Eleventh National Conference on Artificial Intelligence*, 902–903. AAAI.

[44] Z. Feng, R. Dearden, N. Meuleau, and N. Washington. Dynamic programming for structured continuous Markov decision problems. In *UAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 154–161, 2004.

[45] L. Ford and D. Fulkerson: Maximal flow through a network. In *Canadian Journal of Mathematics*, 8, 399–404, 1956.

[46] S. Galster. *An examination of complex human-machine system performance under multiple levels and stages of automation*. WPAFB, OH: Technical Report No. AFRL-HE-WP-TR-2003-0149, 2003.

[47] A. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Planning, Scheduling and Control*, 23(6), 1491–1502, 1993.

[48] A. Garvey and V. Lesser. Design-to-time scheduling and anytime algorithms. *SIGART Bulletin*, 7(3), 1996.

[49] P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. In *Journal of Artificial Intelligence Research*, 24, 49–79, 2005.

[50] C. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[51] C. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. In *Journal of Artificial Intelligence Research* 22, 143–174, 2004.

[52] I.J. Good. Twenty-seven principles of rationality. In *Foundations of Statistical Inference*, Holt, Rinehart and Winston, 1971.

[53] P. Deegan, R. Grupen, A. Hanson, E. Horrell, S. Ou, E. Riseman, S Sen, B. Thibodeau, A. Williams, and D. Xie. Mobile manipulators for assisted living in residential settings. In *Autonomous Robots, Special Issue*, 2007.

[54] E. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI '04: Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004.

[55] E. Hansen and S. Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. In *Artificial Intelligence*, 126, 139–157, 2001.

[56] S. Hart and L. Staveland. Development of NASA-TLS (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*, Amsterdam: North Holland, 1988.

[57] J. Hoey, R. St-Aubin,A. Hu, and C. Boutilier. Optimal and approximate stochastic planning using decision diagrams. *University of British Columbia Technical Report TR-00-05*, 2000.

[58] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic Planning Using Decision Diagrams. In *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999.

[59] J. Hoey, P. Poupart, C. Boutilier, and A. Mihailidis. POMDP models for assistive technology. In *Proceedings of the AAAI Fall Symposium on Caring Machines: AI in Eldercare*, 2005.

[60] J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *IJCAI '05: Proceedings of International Joint Conference on Artificial Intelligence*, 2005.

[61] B. Horling, V. Lesser, R. Vincent, T. Wagner, A. Raja, S. Zhang, K. Decker, and A. Garvey. The taems white paper. white paper, 1999.

[62] E. Horvitz. Time-Dependent utility and action under uncertainty. In *UAI '91: Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, 1991.

[63] E. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Workshop on Uncertainty in Artificial Intelligence*, 1987.

[64] E. Horvitz., G. Cooper, and D. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *IJCAI '89: Proceedings of International Joint Conference on Artificial Intelligence*, 1989.

[65] E. Horvitz. *Computation and Action Under Bounded Resources*. Ph.D. Thesis, Stanford University, 1990.

[66] Joint Planning and Development Office. *Concept of Operations for the Next Generation Air Transportation System*. JPDO, 2007.

[67] B. Kantowitz and P. Casper. Human workload in aviation. In *Human Factors in Aviation*, 157–187, San Diego, CA: Academic Press, 1988.

[68] H. Kitano and S. Tadokoro. RoboCup rescue. *The Computer Journal*, 2009.

[69] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. In *IEEE Transactions on Information Theory*, 47, 498–519, 2001.

[70] C. R. Kube and H. Zhang. Task modeling in collective robotics. *Autonomous Robots*, 4(1), 53–72, 1997.

[71] A. Kumar and S. Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *AAMAS '09: Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2009.

[72] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach*, Pearson Education, 2009.

[73] Latorella, K. Investigating interruptions: implications for flightdeck performance. *NASA/TM-1999-209707 Technical Report*, NASA, 1999.

[74] V. Lesser, C. Ortiz, and M. Tambe (eds.). *Distributed Sensor Networks: a Multiagent Perspective*. Kluwer Publishing, 2003.

[75] L. Li and M. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI '05: Proceedings of the Seventh National Conference on Artificial Intelligence*, 1175–1180, 2005.

[76] S Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21, 498–516, 1973.

[77] R. T. Maheswaran, P. Szekely, M. Becker, S. Fitzpatrick, G. Gati, J. Jin, R. Neches, N. Noori, C. Rogers, R. Sanchez, K. Smyth, and C. Vanbuskirk. Predictability and criticality metrics for coordination in complex environments. In *AAMAS '08: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[78] J. Marecki, S. Koenig, and M. Tambe. A fast analytical algorithm for solving Markov Decision Processes with real-valued resources. In *IJCAI '07: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[79] D. McLaughlin et al. Short-wavelength technology and the potential for distributed networks of small radar systems. *Bulletin of the American Meteorological Society*, 90, 1797-1817, 2009.

[80] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 438–445, 2004.

[81] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 161–168, 2003.

[82] R. Morris, J. Gasch, L. Khatib, and S. Covington. Local search for optimal global map generation using mid-decadal landsat images. In *AAAI '08: Proceedings of the Twenty-First Conference on Artificial Intelligence*, 1706–1711, Chicago, Illinois, 2008.

[83] H. Mostafa and V. Lesser. Offline Planning for Communication by Exploiting Structured Interactions in Decentralized MDPs. In *IAT '09*, 2009.

[84] D. Musliner and R. Goldman. Coordinated plan management using multiagent MDPs. In *AAAI Spring Symposium*, 73–80, 2006.

[85] R. Nair and M. Tambe. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI '03: Proceedings of the Third International Joint Conference on Artificial Intelligence*, 2003.

[86] R. Nair, M. Tambe, M. Roth, and M. Yokoo. Communication for improving policy computation in distributed POMDPs. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent systems*, 2004.

[87] R. O'Donnell and F. Eggemeier. Workload assessment methodology. In *Handbook of Perception and Performance (vol 2)*, K.R. Boff et al. (eds.), New York: Wiley, 1986.

[88] F. Oliehoek, M. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *AAMAS '08: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems*, 2008.

[89] F. Oliehoek, S. Whiteson, and M. Spaan. Lossless clustering of histories in decentralized POMDPs, In *AAMAS '09: Proc. of the International Conference on Autonomous Agents and Multi-Agent Systems*, 2009.

[90] D. Norman and D. Bobrow. On data-limited and resource limited processing. In *Journal of Cognitive Psychology*, 7, 44–60, 1975.

[91] R. Parasuraman, T. Sheridan, and C. Wickens. A model for types and levels of human interaction with automation. In *IEEE Transactions on Systems, Man, and Cybernetics*, 30, 286–297, 2000.

[92] D. Pellier and H. Fiorino. Coordinated exploration of unknown labyrinthine environments applied to the pursuit evasion problem. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[93] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI '05: Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 19, 266, 2005.

[94] M. Petrik and S. Zilberstein. A bilinear approach for multiagent planning. In *Journal of Artificial Intelligence Research* 35, 235–274, 2009.

[95] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In *IJCAI '03: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[96] P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes.* Ph.D. thesis, University of Toronto, 2005.

[97] Proctor, P. Integrated cockpit safety system certified. *Aviation Week and Space Technology*, 61, 1998.

[98] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. John Wiley And Sons, 2005.

[99] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: analyzing teamwork theories and models. In *Journal of Artificial Intelligence Research*, 16, 389–423, 2002.

[100] E. Rachelson, P. Fabiani, and F. Garcia. Adapting an MDP planner to time-dependency: Case study on a UAV coordination problem. In *4th Workshop on*

Planning and Plan Execution for Real-World Systems: Principles and Practices for Planning in Execution, 2009.

[101] A. Raja and V. Lesser. Coordinating agents' meta-level control. In *Proceedings of the AAAI 2008 Workshop on Metareasoning: Thinking about Thinking*, 2008.

[102] A. Raja and V. Lesser. Meta-level reasoning in deliberative agents. In *IAT '04: Proceedings of the International Conference on Intelligent Agent Technology*, 141–147, 2004.

[103] A. Raja and V. Lesser. A framework for meta-level control in multi-agent systems. In *Autonomous Agents and Multi-agent Systems*, 15, 147–196, 2007.

[104] L. Ren, D. Chang, S. Solak, J.P. Clarke, E. Barnes, and E. Johnson. Simulating air traffic blockage due to convective weather conditions. In *Proceedings of the 2007 Winter Simulation Conference*, 2007.

[105] R. Rogers and S. Monsell. Costs of a predictable switch between simple cognitive tasks. In *Journal of Experimental Psychology: General*, 124, 207–231, 1995.

[106] M. Roth, R. Simmons, and M. Veloso. Decentralized communication strategies for coordinated multi-agent policies. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, 4, 2005.

[107] M. Roth, R. Simmons, and M. Veloso. What to communicate? Execution-time decision in multi-agent POMDPs. In *Eighth International Symposium on Distributed Autonomous Robotic Systems*, 2006.

[108] S. Russell, P. Norvig, J. Canny, J. Malik, and D. Edwards. *Artificial Intelligence: a Modern Approach*. Prentice Hall Englewood Cliffs, NJ, 1995.

[109] S. Russell, D. Subramanian, and R. Parr. Proving bounded optimal agents. In *IJCAI '93: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.

[110] S. Russell and D. Subramanian. Provably bounded optimal agents. In *Journal of Artificial Intelligence Research*, 2, 575–609, 1995.

[111] S. Russell and E. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 1989.

[112] T. Saffell, A. Alexander, A. Carlin, A. Chang, and N. Schurr. An integrated alerting and notification system utilizing stages of automation and uncertainty visualization. In *Proceedings of the International Symposium of Aviation Psychology*, Dayton, OH: Wright State University, in press.

[113] T. Sandholm. Terminating decision algorithms optimally. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 2003.

[114] P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real world. In *Journal of Artificial Intelligence Research*, 17, 171–228, 2002.

[115] W. Schneider and A. Fisk. Concurrent automatic and controlled visual search: Can processing occur without cost? In *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 8, 261–278, 1982.

[116] N. Schurr and J. Marecki. Resolving inconsistencies in adjustable autonomy in continuous time (RIAACT): A robust approach to adjustable autonomy for human-multiagent teams. In *AAMAS '08: Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, 2008.

[117] M. Schut and M. Wooldridge. The control of reasoning in resource-bounded agents. In *Knowledge Engineering Review*, 16(3), 215-240, 2001.

[118] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. In *Autonomous Agents and Multi-Agent Systems*, 17(2), 190–250, 2008.

[119] S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for Dec-POMDPs. In *UAI '07: Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence*, 2007.

[120] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for Dec-POMDPs. In *IJCAI '07, Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[121] D. Slate and L. Atkin. CHESS 4.5 – Northwestern University chess program. In *Chess Skill in Man and Machine*, 82–118, 1977.

[122] J. Shen and V. Lesser. Communication management using abstraction in distributed bayesian networks. In *AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[123] H. Simon. A behavioral model of rational choice. In *Quarterly Journal of Economics*, 69, 99–118, 1955.

[124] H. Simon. From substantive to procedural rationality, In *Method and Appraisal in Economics*, 129–148, 1976.

[125] H. Simon and J. Kadane. Optimal problem solving search: All or nothing solutions. *Computer Science Technical Report CMU-CS-74-41*, Carnegie Mellon University, 1974.

[126] R. Smallwood and E. Sondik. The optimal control of partially observable Markov processes over a finite horizon. In *Operations Research 21*, 1973.

[127] E. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. Ph.D. thesis, Stanford University, 1971.

[128] S. F. Smith, A. Gallagher, T. Zimmerman, L. Barbulescu, and Z. Rubinstein. Distributed management of flexible times schedules. In *AAMAS '07: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007.

[129] L. Song and J. Kuchar. Dissonance between multiple alerting systems. In *IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans*, 33(3), 366–385, 2003.

[130] R. St-Aubin, J. Hoey, and C. Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *NIPS '00: Advances in Neural Information Processing 13*, 2000.

[131] M. Stefik. Planning and meta-planning. In *Artificial Intelligence*, 16(2), 141–170.

[132] http://stoprog.org

[133] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *UAI '04: Proceedings of the International Conference on Uncertainty in Artificial Intelligence*.

[134] S. Solak, J-P. Clarke, Y. Chang, L. Ren, and A. Vela. Air traffic flow management in the presence of uncertainty. In *Proceedings of the 8th USA/Europe Seminar on Air Traffic Management Research & Development*, 2009.

[135] Perseus: Randomized point-based value iteration for POMDPs. In *Journal of Artificial Intelligence Research*, 24, 195–220, 2005.

[136] M. Spaan, G. Gordon, and N. Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006.

[137] M. Spaan, F. Oliehoek, and N. Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *ICAPS '08: Proceedings of the International Conference on Automated Planning and Scheduling*, 2008.

[138] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *UAI '05: Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 2005.

[139] D. Teneketzis and Y. Chi Ho. The Decentralized Wald Problem. *Information and Computation*, 73, 23–44,1987.

[140] B. Thuraisingham. Privacy-preserving data mining:: Development and directions. In *Journal of Database Management*, 16(1), 75–87, 2005.

[141] J. Tsitsiklis and M. Athans. On the complexity of decentralized decision making and detection problems. In *IEEE Transactions on Automatic Control* 30(5), 440–446, 1985.

[142] P. Varakantham, R. Maheswaran, and M. Tambe. Exploiting belief bounds: practical POMDPs for personal assistant agents. In *AAMAS '05: Proceedings of the Fourth International Conference on Autonomous Agents and Multiagent Systems*, 2005.

[143] L. Valiant. A theory of the learnable. In *Philosophical Transactions of the Royal Society of London A*,27(11), 1134–1142,1984.

[144] H. Veltman. *Mental Workload: Lessons learned from subjective and physiological measures*, FAA, 2001.

[145] K. Vicente. Cognitive work analysis toward safe, productive, and healthy computer-based work. In *IEEE Transactions on Professional Communication* 46(1), 2003.

[146] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*, Princeton University Press, 1944.

[147] A. Wald. *Sequential Analysis*, Wiley, New York, 1947.

[148] C. Wickens and J. Hollands. *Engineering Psychology and Human Performance*, Upper Saddle River, NJ: Prentice Hall, Inc., 2000.

[149] J. Williams, P. Poupart, and S. Young. Partially observable Markov Decision Processes with continuous observations for dialog management. In *Recent Trends in Discourse and Dialogue*, 191–217, 2008.

[150] S. Williamson, E. Gerding, and N. Jennings. Reward shaping for valuing communications during multi-agent coordination. In *AAMAS '09: Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, 641–648, 2009.

[151] A. Wolfe. *Observation Abstraction in Partially Observable Environments*. Ph.D. thesis, 2010.

[152] F. Wu, S. Zilberstein, and X. Chen. Point-based policy generation for decentralized POMDPs. In *AAMAS '10: Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2010.

[153] P. Xuan and V. Lesser. Multi-agent policies: From centralized ones to decentralized ones. In *AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002.

[154] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents*, 2001.

[155] P. Xuan and V. Lesser. Incorporating uncertainty in agent commitments. In *Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages*, 1999.

[156] S. Zilberstein and S. Russell. Optimal composition of real-time systems. In *Artificial Intelligence*, 82, 181–213, 1996.

[157] S. Zilberstein and A. Carlin. Bounded rationality in multiagent systems using decentralized metareasoning. Presented at NIPS 2011 workshop *Decision Making with Multiple Imperfect Decision Makers*, 2011.

[158] M. Zink, E. Lyons, D. Westbrook, J. Kurose, and D. Pepyne. Closed-loop architecture for distributed collaborative adaptive sensing of the atmosphere: meteorological command and control. *International Journal for Sensor Networks*, 6(4), 2009.