2-2009

Agent Interactions In Decentralized Environments

Martin William Allen
University of Massachusetts - Amherst

AGENT INTERACTIONS IN DECENTRALIZED ENVIRONMENTS

A Dissertation Presented

by

MARTIN WILLIAM ALLEN

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 2009

Computer Science

AGENT INTERACTIONS IN DECENTRALIZED ENVIRONMENTS

A Dissertation Presented

by

MARTIN WILLIAM ALLEN

Approved as to style and content by:

Shlomo Zilberstein, Chair

Victor Lesser, Member

David Jensen, Member

Abhijit Deshmukh, Member

Andrew G. Barto, Department Chair
Computer Science

ACKNOWLEDGMENTS

I would like to thank my advisor, Shlomo Zilberstein, for all his hard work and support. From the very beginning of my time at the University of Massachusetts Amherst, Shlomo was incredibly supportive, supplying me with many of the key ideas that form this body of work, and providing me with a first-rate example of how a professional academic operates. I hope that I can provide a similar example to my own students one day. His patience and firm insistence over the last portion of my dissertation writing were key to its successful completion. I look forward to working with him in the future.

The other members of my committee were also key to whatever success I can claim here. Victor Lesser provided me with constructive feedback and fresh ideas throughout the writing process, and I could always count on him to ask the questions that pushed me hardest. David Jensen was a tremendous source of encouragement and practical advice, and in his exemplary work as a teacher and researcher, he continues to inspire me and my work. His course on empirical methods in computer science did more to improve my work than any other in my college career. Finally, Abhijit Deshmukh, my external reader, was very gracious and very helpful. His questions during my defense opened up many new horizons for future work.

My time in the department was immeasurably improved by my fellow students. As the reader will note, the work here draws heavily on developments made by many of them, and I thank Chris Amato, Raphen Becker, Alan Carlin, Zhengzhu Feng, Claudia Goldman, Marek Petrik, and Sven Seuken in particular for sharing their ideas with me. I also wish to give special thanks to Dan Bernstein and Anyuan Guo: not only did they lay the groundwork for much of what I did here, but they

were exceptional friends as well, literally from the first day I ever spent in Amherst. While the list of others may be too great to include in full, I would like to thank the other members of the RBR lab who contributed to my tenure there over the years: Andrew Arnt, Stephen Ambrogio, Mark Gruman, and Akshat Kumar. In addition, my thanks goes out to all the people I lived with over my years here, including (but not limited to) Fernando Diaz, Greg Druck, Jennifer Lyons, Mike O'Neill, Rob Platt, Louis Theran, Beth Quill, and Jerod Weinman. You all made it worthwhile.

I would also like to thank Michele Roberts for all her help to myself and the RBR group over the years, and for doing it all while remaining so friendly, no matter what we demanded of her. Similar thanks go out to Leeanne Leclerc and Sharon Mallory for coordinating the graduate program, and to Michelle Eddy and Barbara Sutherland for all they did as well.

During my long and somewhat peripatetic education, my family supported me through thick and thin. While I am sure they are as thrilled as I am that my school years have finally ended, they were never anything but encouraging, and I could not have done it without them. From childhood, my mother and father allowed me—encouraged me, in fact—to pursue my own intellectual ends and choose my own life path, while always keeping me grounded. Whatever strength of character I can lay claim to, I owe to them. Mom and dad, and all the rest, I love you.

Finally, I would like to acknowledge the single greatest source of my success, my girlfriend, Courtney Maloney. Without her, I honestly do not know where I would be today. While we spent so much time apart during our past few years, it was always made easier by the fact that she was there, and that each step I took brought me closer to the day that we would be together again. She never complained when I spent that extra hour (or two) talking about my work, and had she not talked me through more than a few dark nights of the soul, I have no doubt that I would never have got where I am today. I love you, Courtney.

ABSTRACT

AGENT INTERACTIONS IN DECENTRALIZED ENVIRONMENTS

FEBRUARY 2009

MARTIN WILLIAM ALLEN

B.A. (Hon.), SIMON FRASER UNIVERSITY

M.A., UNIVERSITY OF PITTSBURGH

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

The decentralized Markov decision process (Dec-POMDP) is a powerful formal model for studying multiagent problems where cooperative, coordinated action is optimal, but each agent acts based on local data alone. Unfortunately, it is known that Dec-POMDPs are fundamentally intractable: they are NEXP-complete in the worst case, and have been empirically observed to be beyond feasible optimal solution.

To get around these obstacles, researchers have focused on special classes of the general Dec-POMDP problem, restricting the degree to which agent actions can interact with one another. In some cases, it has been proven that these sorts of structured forms of interaction can in fact reduce worst-case complexity. Where formal proofs have been lacking, empirical observations suggest that this may also be true for other cases, although less is known precisely.

This thesis unifies a range of this existing work, extending analysis to establish novel complexity results for some popular restricted-interaction models. We also establish some new results concerning cases for which reduced complexity has been proven, showing correspondences between basic structural features and the potential for dimensionality reduction when employing mathematical programming techniques.

As our new complexity results establish that worst-case intractability is more widespread than previously known, we look to new ways of analyzing the potential average-case difficulty of Dec-POMDP instances. As this would be extremely difficult using the tools of traditional complexity theory, we take a more empirical approach. In so doing, we identify new analytical measures that apply to all Dec-POMDPs, whatever their structure. These measures allow us to identify problems that are potentially easier to solve on average, and validate this claim empirically. As we show, the performance of well-known optimal dynamic programming methods correlates with our new measure of difficulty. Finally, we explore the approximate case, showing that our measure works well as a predictor of difficulty there, too, and provides a means of setting algorithm parameters to achieve far more efficient performance.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Multiagent environments pose some of the most significant challenges to successful planning and action. In order to make intelligent decisions about what to do, individuals must not only factor in what is known about the world around them, but also take account of the actions of others. While the world may be modeled in terms of its dynamics—the certainties and probabilities governing how it evolves—other agents must be taken into account in different terms. After all, they too will be trying to make their choices intelligently, based in large part upon what they think we will choose to do. Thus, if we do one thing, they may do another in response, and if they do *that*, then perhaps it would have been smarter to have chosen something else altogether, since now the world is going to behave quite differently. These sorts of *agent interactions and influences* are central to solving decentralized multiagent decision problems, and form the basis of our work here.

1.1 Decentralized Problem Solving

Cooperative multiagent planning is the problem of computing a set of behaviors for a group of agents that act in the same environment, and that seek to maximize some joint objective. Such planning problems are *decentralized* when each agent must act towards the *global goal* based upon a *local policy*. That is, the agent has to choose actions based solely upon its own private information, which in general only illuminates some incomplete sub-part of the current state of the environment, and does not usually tell the agent what others are doing, or are planning to do in future.

Decentralized multiagent problems of this sort arise in many real-world domains. Even tightly coordinated groups of problem-solvers, with circumscribed ends in mind, cannot always count on being able to share all relevant information about their world in a global fashion. As an example, teams of athletes—whether they be human beings, or the sorts of artificial agents involved in something like the well-known Robocup competition—play certain specific roles, and share a limited number of simple goals, such as scoring points or preventing them from being scored. During actual game-play, however, each agent will generally only have a limited view of the field of play, focussing on the nearby conditions, or on the opposing player currently being defended. In addition, the immediate decisions of other team members cannot always be known exactly, even in the presence of a general team strategy agreed to ahead of time. This in-the-moment lack of access to global conditions and actions can lead to considerable uncertainty about the wisest immediate course of action, since outcomes can be very different depending upon what others are seeing and doing. Given their ubiquity and their complexity, these sorts of problems are of central concern in modern artificial intelligence research.

Decentralized Markov decision processes (Dec-POMDPs) provide us with a powerful mathematical model of such problems. They are at once generally applicable, while at the same time giving a precise characterization of the intrinsic computational properties of the domains of interest. Unfortunately, it is known that such decentralized multiagent problems have significantly higher worst-case complexity than their single-agent or centralized counterparts. Practical experience with solution algorithms has shown, furthermore, that worst-case complexity is a reliable indicator of general problem difficulty, since most truly decentralized problems are genuinely intractable, and do not succumb to reasonably computable optimal (or bounded-optimal) solution.

One of the main sources of this difficulty is the fact that each individual decision-maker lacks information about the states and actions of its fellows when deciding

whether or not to take a particular action at the present time. Even centralized offline planning methods, which can take the entire problem specification as input, must therefore still produce plans that can be *executed* in a decentralized manner. This planning then becomes very complex, since it must take into account all possible beliefs that an agent might end up having about the present state of the environment. These beliefs—if they are to serve as the basis for intelligent plans—must be based upon all the ways in which the various agents and their actions can interact, affecting either the outcomes of others' actions, the rewards that are gained at any instance, or the various observations of the shared environment.

Allowing agents to share information, either explicitly or implicitly, may reduce uncertainty about these global interactions. In extreme cases, when communication is free, instantaneous, and mutually understood, decentralized planning becomes equivalent to single-agent planning, since all agents share a unified perspective. In practice, however, communication has some cost, whether it be the actual bandwidth used by a transmission, or some other function of resources required, and it has been shown that computing policies involving costly communication can be just as hard as computing optimal solutions without communication. Consequently, there is no simple solution to such a general decentralized control problem, and researchers have concentrated on restricting the model in various ways. Since much complexity seems to arise from interacting effects of various agent actions, various special models have been proposed that limit those effects.

Our work develops on this theme, and extends it in yet more general directions. On the one hand, we consider a variety of these simplified and restricted models, examining their properties and interrelations between them. At the same time, we also broaden the focus, moving beyond the specification of distinct special cases in order to consider *general models* of interactions and influence. We examine different ways of doing so, providing and analyzing various means of quantifying otherwise potentially

vague notions, such as the interaction between agents and the relative centralization of control. We demonstrate that the general complexity of decentralized multiagent problem-solving can be understood broadly through a focus on the structure of agent interactions. The project extends the existing literature and provides new directions for research in the field.

1.2 Summary of Contributions

This dissertation examines the relationship between agent interactions and problem complexity in decentralized problems. We establish the computational complexity of some interesting existing models and formalize new measures that allow us to specify and quantify notions like agent influence. On the empirical side, we investigate the performance profiles of some important solution algorithms, providing evidence for the significance of the measures we define. The contained results unify and extend recent research in cooperative decentralized planning and control. We analyze the effects of restricted interactions on problem complexity, putting precise bounds on those effects. Further, we demonstrate how algorithms can exploit measurable degrees of centralization, allowing us to generate high-quality solutions to decision problems without using undue computational resources.

Overall, the thesis strengthens and generalizes existing knowledge about how restricting the ways in which agents interact can or cannot simplify multiagent planning. By using the precise formal framework of decentralized partially observable Markov decision processes (Dec-POMDPs), we are able to gain real traction on the question of how exactly interactions affect problem complexity. On their own, general Dec-POMDPs are far too hard to solve, and known optimal algorithms are primarily of theoretical interest, since they are generally infeasible for practical purposes. While reduced complexity results are known for some special classes of Dec-POMDPs—characterized by various probabilistic independence properties—there are a number

of other special models for which less is known. Our work on these special models produces some significant new formal complexity results, which provide much better bounds than the weak lower or upper range results already known. As we will show, reductions in fundamental worst-case complexity are relatively rare, as even some quite restricted models of interaction have the same high level of difficulty as does the general Dec-POMDP problem.

A unifying notion in many of these models is that of an *event*, which may be thought of for the moment as the taking of some action, resulting in some change of overall system state. Events have been used to model interactions in various ways, including problems where agents interact only via a shared reward function, specified in event-related terms. Such problems have proven amenable to interesting multilinear programming methods. Our work shows how intrinsic features of the mathematical programs relate to the event-based structure, which is used to specify the problems in an intuitive and straightforward manner. These results demonstrate tight connections between the two frameworks, showing how essential problem dimensionality is directly related to the number of events necessary and sufficient to specify it.

Given that the worst-case difficulty of the general model applies also in many more restricted cases, new techniques are needed to identify the average-case and empirical difficulty of decentralized problems in general. We tackle this problem in our last two chapters, providing overall measures of the degree to which agents interact, and showing that they correlate with the practical difficulty of the associated decision problem. These measures are defined using information-theoretic relationships between an agent's actions and the dynamics of the underlying Dec-POMDP; the most important of these, *influence gap*, measures the difference between the degrees to which different agents can control the system. We define these relationships and associated measures, and prove some interesting properties about them. Among other

things, we show how they can be used to identify agents that are unimportant to the system dynamics, and that can be ignored, making planning simpler.

Finally, we demonstrate some significant connections between the influence gap measure and the performance of optimal and approximate dynamic programming algorithms for Dec-POMDPs. In the exact case, an increasing gap—which we associate with increased centralization—correlates with an increased ability to solve problems optimally. In the approximate case, similar results apply: as influence gap increases, the absolute value of possible solutions goes up dramatically, and the approximation algorithms converge more quickly to near-maximal solutions. This latter result is perhaps the most interesting, since it allows us to categorize problems in advance, setting algorithm parameters to return high-value solutions much more efficiently.

1.3 Previous Work

Our work on agent interaction draws on a well-established and deep body of artificial intelligence research over the last decades. Here, we outline some of this history, and discuss relevant publications and results.

1.3.1 Centralized Markov Decision Models

The *Markov decision process* (MDP) framework has long proven useful in artificial intelligence research. The MDP model allows an exact analysis of agent actions in fully observable, stochastic problem environments. Furthermore, there are a number of well-understood and well-developed solution algorithms, providing tractable methods for generating good solutions to such problems, in the form of *policies*, which are mappings from domain states to agent actions; authoritative treatments can be found in Puterman [103], or Russell and Norvig [111]. MDPs are also well-suited for the application of a variety of learning algorithms. While extremely large problems are still challenging, learning methods converge to near-optimal policies in an efficient

manner for those with more reasonable state-spaces; see Sutton and Barto [123] for a full account of such techniques.

MDPs have become central to research on planning for uncertain domains; for an overview, see the survey of Boutilier, Dean, and Hanks [20]. Of course, there are bounds on what traditional algorithms can achieve for very large MDPs, where the state-space is so big that establishing and evaluating complete policy-mappings is combinatorially daunting. However, there has been a great amount of progress in exploiting domain structure to make solving larger problems more feasible, at least in some cases. Examples include the use of Bayesian network representations (Boutilier, Dearden, and Goldszmidt [21]), or algebraic decision diagrams (Feng and Hansen [38]), and research on so-called *weakly coupled problems* (Meuleau et al. [83]). Approximate solution methods for MDPs have also been studied, for example by Guestrin et al. [50, 51, 52], employing the assumption that the reward function can be decomposed into local reward functions, each of which depends upon only a small subset of the overall state variables.

Another strain of research has involved problems with somewhat more complex structure. In particular, interest in cases where an agent lacks complete information about the current state of its environment led to the introduction of the *partially observable* (POMDP) problem variant. In such problems, agents do not have access to the actual state of the environment, but rather only to observations of that environment, which can only induce probabilistic belief-states about the current state. For practical purposes, research on POMDPs dates to the 1960's Operations Research work of Aström [6], who showed that such problems could reduce to a special form of belief-state MDP, and the work a decade later of Smallwood and Sondik [120], who were able to establish optimal dynamic programming algorithms for such problems, based on value iteration. Later work by Platzman [101] and Sondik [121] extended the

range of dynamic programming methods, providing solution algorithms for problems with infinite or indefinite time-horizons.

Since that earlier period, solution algorithms for POMDPs have been developed in many different directions, and the model has since proven very popular in AI research, being both relatively simple to specify and broadly applicable. The range of work here is extremely wide, encompassing any number of distinct algorithmic techniques. Many extended optimal methods have been developed, based on such ideas as: best-first search (Lark [64]); Bayesian networks for compact representation (Boutilier and Poole [22]); incremental pruning techniques (Cassandra, Littman, and Zhang [29]); factored state representations (Hansen and Feng [57]); hierarchical finite-state controllers (Hansen and Zhou [60]); and structured pruning techniques (Feng and Zilberstein [39, 40]). Learning techniques have been explored (Littman, Cassandra, and Kaelbling [76]), as have approximate methods (Parr and Russell [96], Hauskrecht [61], Poupart and Boutilier [102]). Most recently, work on point-based approximations by Pineau, Gordon, and Thrun [100], and Ross et al. [110], has proven very interesting.[1] Given this profusion of advances, much recent AI work on single-agent planning for stochastic domains has thus been based on MDPs and POMDPs, as described in Dean et al. [36] and Kaelbling, Littman, and Cassandra [66].

The single-agent Markov models just outlined can be straightforwardly extended to cases where multiple agents work together, with joint access to either the same global system state or the same partial observation of that state. As Boutilier [19] has pointed out, such *centralized multiagent MDPs* (MMDPs) are effectively equivalent to single-agent problems, since the joint actions of multiple agents can be regarded simply as factored actions of an individual central controller. Clearly, the same point applies to centralized multiagent POMDPs as well, and in each case all the resources

[1] Useful surveys of algorithmic results from various eras can be found in the work of Cheng [32], Lovejoy [78], Cassandra [27, 28], Hansen [58], Zhang [138], and Murphy [84].

of existing work on single-agent problems can be exploited. Such models provide some interesting challenges, and work has been done on both planning (Guestrin et al. [51, 50, 53]) and learning methods (Claus and Boutilier [33], Kapetanakis and Kudenko [68], Wang and Sandholm [133], Chalkiadakis and Boutilier [30, 31]).

1.3.2 Decentralized Problems and Models

In real-world domains, however, it is rare for all agents to possess the same sorts of information, and we would argue that the centralized types of multiagent problems hold somewhat less interest than their genuinely decentralized cousins, where agents must act on the basis of local information, even as they seek to coordinate and cooperate. Decentralized control problems abound in practice, and researchers have considered many different examples over the years. Such problems include:

Autonomous space exploration vehicles. Projects like NASA's Mars Rover program involve multiple vehicles working together, pursuing science goals in highly uncertain planetary environments. Such efforts would be much improved if exploration could proceed autonomously. Decentralized models for rover domains have been proposed by Zilberstein et al. [143] and Becker et al. [12].

Distributed tracking networks. Many modern applications feature multiple sensors or robotic agents that must track or identify such things as mobile targets or weather phenomena. Given the limited range and efficacy of the individual tracking agents or nodes, such domains often require decentralized control over a widely dispersed network. Parker [94] proposes decentralized techniques for mobile team robotic tracking of targets, while Horling et al. [62] and Nair et al. [87] present methods for use in distributed sensor nets.

Disaster rescue and response. Recent advances in robotics have made the use of semi-autonomous mechanized emergency response more feasible. To do so, however, requires more sophisticated coordination techniques, especially given

the chaotic and unpredictable environments in which rescue agents must operate. Casper, Micire, and Murphy [26] address robotic planning techniques for such domains, while Thomas [127] addresses decentralized learning and planning methods for the coordination of firefighting response units.

Decentralized systems design. Many common task in modern computing systems, such as network routing and distributed information processing, now proceed by way of networks of localized decision-makers. Coordination and efficiency in such tasks requires new control schemes, distinct from the centralized operations of more traditional computing. Ooi and Wornell [89] first formulated a decentralized multi-access broadcast channel problem, which has been used since in much research on Dec-POMDPs. Cogill et al. [34] consider the problem of load balancing for systems of decentralized queues.

In each of these cases, centralized decision models are inadequate, since agents cannot share all their information among themselves (or incur direct costs for doing so). Even if planners in such domains has access to a complete model of the problem dynamics, and take a centralized viewpoint, the actual execution of plans is still decentralized, as each agent acts based solely upon what it knows locally.

As we shall see, these kinds of problems are considerably more complex than their single-agent or centralized cousins. The basic model here is the *decentralized POMDP* (Dec-POMDP), which extends the basic single-agent POMDP to multiple agents with local information (Bernstein et al. [18, 16, 15]). We provide much more detail about this model in later chapters. Here, we note one important fact: Dec-POMDPs are highly complex: in fact, they are NEXP-complete, a result that leads to many difficulties, and which is a motivation behind much of our research. (Full discussion of this result and its implications can be found in Section 2.2.)

There are also a number of other models of these sorts of decentralized problem structures, including Goldman and Zilberstein's *Dec-POMDP with communication*

(Dec-POMDP-COM) [47], and the work of Pynadath and Tambe [105] on the *multi-agent team decision problem* (MTDP), which also has a variant with communication (MTDP-COM). Recently, Seuken and Zilberstein [113, 116] have shown that all of these variants are equivalent to the basic Dec-POMDP. Another variant, the *interactive POMDP* (I-POMDP) proposed by Gmytrasiewicz and Doshi [42], is more expressive, but possesses even more daunting complexity. (For this reason, we will not consider that model here, and will begin with the basic Dec-POMDP, before considering special sub-cases.)

1.3.3 Algorithms for Dec-POMDPs

The first known exact solution algorithm for finite-horizon Dec-POMDPs extends dynamic programming (DP) for single-agent POMDPs, as devised by Hansen [58]. The DP algorithm for Dec-POMDPs, introduced by Hansen, Bernstein, and Zilberstein [59], works bottom-up to generate finite-horizon policies, and then applies iterated pruning techniques to eliminate dominated strategies, reducing the number that must be considered to find an optimal solution. (We give full details of this approach in Section 2.3). However, such improvements are not sufficient to make the general problem tractable; even for a very simple problem, the method cannot generate policies beyond a handful of time-steps. Similar results have been reported with respect to top-down methods employing heuristic search. Again, as reported by Szer, Charpillet, and Zilberstein [124, 126], only the smallest problems can be solved. These sorts of practical limitations also apply to ϵ-optimal methods for infinite-horizon problems, which develop bounded-memory finite state controllers for Dec-POMDPs, as described in the work of Bernstein, Hansen, and Zilberstein [13]. An overview of results on infinite-horizon techniques can be found in the thesis of Bernstein [15].

Due to complexity bounds, approximation of Dec-POMDPs is very difficult. Rabinovich, Goldman, and Rosenschein [107] have shown that ϵ-approximation of Dec-

POMDPs is just as hard as optimal solution. As a result, approximation methods are either impractical, or provide only rough guarantees of quality. Among the latter, we find equilibrium-based methods for finite-horizon Dec-POMDPs, like the JESP algorithm of Nair et al. [86]. Other such techniques include linear and nonlinear programming methods for (Amato, Bernstein, and Zilberstein [3, 4], Aras, Dutech, and Charpillet [5]), and point-based approximate algorithms (Szer and Charpillet [125]).

Of particular interest to us here are a pair of approximate algorithms that generalize the optimal dynamic programming method, making it memory-bounded. First devised by Seuken and Zilberstein [114, 116], and then extended by Carlin and Zilberstein [23], these techniques place hard caps on the number of trees retained per agent at each DP backup step, using top-down heuristics to isolate those that seem most useful. (Again, we will give full details of these algorithms in Section 2.4.) A good overview of many of these results, and comparison with other algorithms can be found in the review papers of Seuken and Zilberstein [113, 116].

1.3.4 Reduced Interaction Models

Given the difficulty of solving the general Dec-POMDP problem class, whether optimally or approximately, there has been much interest in special cases, where structure is simplified or restricted enough that some computational traction can be gained. One obvious such subcase are those problems in which the various aspects of the system dynamics are independent of one another, with respect to individual agents. Becker et al. [7, 11, 12] consider models where state-transitions and observations are functionally independent, but rewards gained are not; in such cases, the problems are of reduced complexity, and specialized algorithms can come into play. A survey of many known complexity results for various problems with partially independent dynamics can be found in Goldman and Zilberstein [48]. An important and interesting generalization, proposed by Becker, Lesser, and Zilberstein, involves

event-based interactions, where only some small set of particular state-action transitions in the Dec-POMDP create dependencies between state outcomes or rewards [9]. Such models will be considered in detail in later chapters, where we establish for the first time that their worst-case complexity is identical to that of the general case.

Another possibility, found in the work of Guo and Lesser [55, 56], and the thesis of Guo [54], is the *state-based action-sets* framework, which allows agents to interact only by way of the set of possible actions available to them; one agent then influences another in cases where they force a state-transition that alters the other's available choices. Again, we are able to prove that the complexity of these problems is as high as for the general model.

1.3.5 Identifying Interactions and Their Effects

Shen, Becker, and Lesser [119] have suggested that the complexity of a decentralized problem increases in proportion to the "degree of interaction" between agents, a notion upon which our work here seeks to expand. In the game theory literature, furthermore, Kearns and Mansour [69] have shown that bounding the influence that any agent can have on overall reward simplifies the process of calculating equilibria solutions. In the context of single-agent MDPs, Ratitch and Precup [109] demonstrate that the performance of learning algorithms in MDPs is inversely proportional to certain information-theoretic measures of the system dynamics. Our work combines these ideas, examining how performance of solution techniques can be improved in cases where we restrict the effect agents can have on the overall expected value of joint policies, which in turn depend upon the probabilistic problem-dynamics.

Other general models of overall interaction have been proposed. For instance, Carver and Lesser [24, 25] consider *nearly monotonic* problems in information processing, where tasks are divided between multiple agents, and the final global solution is composed out of local sub-solutions. Kim *et al.* [70] have examined local interactions

in networked, distributed POMDP problems. Oliehoek, Spaan, et al. [88, 122] have used factored Markov Game techniques to isolate problems with reduced interactions.

1.3.6 New Solution Techniques

In research on single-agent planning and learning, there has been much interest in how Markov decision processes can be solved more efficiently, given structure in the way the state-space is set up. One interesting line of work has been the literature on *weakly coupled MDPs*, which are nearly decomposable into smaller sub-problems, but contain a number of "bottleneck" states governing transitions between one sub-process and another. In the presence of such states, various heuristic and learning algorithms have been proposed that attempt to solve the separate problems and compose the solutions into a global policy (Parr [95], Meuleau et al. [83], Bernstein and Zilberstein [17]). Although a firm characterization of the difference is not yet available, Littman [77] has suggested that such problems can be significantly easier to solve than full-blown MDPs of the same size, particularly where the number of bottleneck states is reasonably small.

Similar results have been established for some of the reduced-interaction models already mentioned. For instance, in the event-based models just discussed, algorithm performance is known to be a direct function of the number of event-based dependencies. In addition, there has been work on various heuristic approaches to such problems. One example, given by Guo [54], solves problems with state-based actions using policy pruning techniques that are guided by either *optimistic* or *pessimistic* assumptions about the extent to which action-sets will turn out to be restricted. Becker [10] has examined the use of *myopic* heuristics in decentralized planning for communication problems. In this work, complex decisions about when and what information to share between cooperating agents are simplified by restricting the time horizon over which planning is performed. Rather than considering all possible down-stream

effects of communication, that is, agents plan their actions and information-sharing only a step or two ahead, making heuristic assumptions about future eventualities.

Another interesting candidate class of methods involve problem decompositions. As the work in event-based interactions and weakly coupled problems has shown, many decision problems consist of decomposable sub-problems that are *very nearly* separate. Goldman and Zilberstein [46, 47, 49] present problems in which decompositions arise based on decisions about when otherwise decentralized agents should communicate in order to synchronize their local information-states. Xuan and Lesser [135] examine methods for transforming centralized policies into more decentralized ones, making trade-offs between communication and utility; this work provides some interesting measures of solution quality, and helps us understand key interactions in terms of their effect upon solution quality. Similarly, hierarchical agent-control models, such as the TAEMS structure (Lesser et al. [74]), introduce various ways of identifying agent influence and interaction. The work of Wagner and others [130, 131], on methods for scheduling and re-scheduling on the fly in the TAEMS architecture, highlights interesting ways of decomposing decentralized problems, and the interactions therein, in various hierarchical manners. The work of Zhang and others [139, 140, 141, 142] also considers hierarchies, based on *negotiations* as the key form of interaction.

1.4 Outline

The body of this dissertation is structured as follows.

Chapter 2 gives necessary background, defining Dec-POMDPs and Dec-MDPs, and outlining their associated solution concepts. We provide a sample Dec-POMDP, with discussion, to illustrate the ideas behind the formal model. We also briefly discuss the worst-case NEXP-complete theoretical complexity of the problems, and its implications. Finally, we outline the state of the art in solution algorithms, with

special attention paid to the exact and approximate variants of dynamic programming that we later put to use in our empirical studies.

Chapter 3 contains our central complexity results, applied to some interesting existing Dec-POMDP variants. These cases are distinguished by a concentration on distinct, often limited, forms of agent interaction, in order to simplify the problem, or to provide leverage for special-purpose algorithms. We detail these models, and establish their computational complexity for the first time. We review the fundamental result of Bernstein, et al. [16], outlining the proof by reduction that Dec-POMDPs are NEXP-hard. We are then able to extend these techniques to three other classes of restricted-interaction problems, in each case showing that this high level of difficulty is preserved, even when dynamics are apparently simplified. These new complexity results reveal much about the limitations to what restricting interactions can do. Rather than continuing to proliferate special-case models in hopes of finding generally simpler cases, we therefore move on to a new approach, seeking to identify ways of isolating the average-case difficulty of general Dec-POMDPs.

Before doing so, however, we show some new properties of one well-known sub-model. Chapter 4 concentrates upon *events*—defined as state-action pairs—and outlines their use in specifying certain reduced-interaction Dec-POMDP sub-cases. We establish an interesting connection between these events and the essential difficulty of problem-instances. Specifically, we show that the essential dimensionality of a problem, which factors significantly into the practical applicability of mathematical programming methods, is tied directly to the event-structure of the domain.

Chapter 5 moves from specific sub-models to a general treatment of agent influence and interaction. We provide a formal information-theoretic treatment of the effect an agent has on the dynamics of a Dec-POMDP, defining a series of *influence measures* that quantify this effect precisely. We then show that these measures correlate with real features of Dec-POMDPs, demonstrating how agents that possess some influence

must be taken into account while planning, while those that do not have influence may be ignored. Agent influence is also tied to the event-based framework, exposing further connections with existing work.

Chapter 6 examines agent influence empirically. Once influence has been quantified, interaction can be evaluated in a precise fashion. Our experiments then show that the *influence gap*—measuring the difference in the degree to which different agents affect the system, or its *centralization*—correlates with actual algorithm performance. In the case of exact algorithms, increasing influence gap is shown to lead to problems that are usually easier to solve. When using approximate algorithms, an increase in the influence gap also leads to much better performance, as solutions are much more valuable, and can be attained more efficiently.

Finally, Chapter 7 concludes, suggesting implications of the present work, and indicating where we may want to go from here.

CHAPTER 2

DECENTRALIZED CONTROL PROBLEMS: FORMULATION, SOLUTION, AND COMPLEXITY

The material in this chapter provides needed background, presenting the basic formal model of decentralized control problems that we employ throughout. We also discuss known solution techniques, optimal and approximate, outlining a set of *dynamic programming* algorithms—in both exact and memory-bounded variants—that we will use in empirical investigations of the effects of agent influence and centralization on problem difficulty (Chapter 6). Further, we outline the fundamental difficulties and limitations presented by practical and theoretical problem complexity.

2.1 Decentralized Markov Decision Processes

Our work is based upon *decentralized partially observable Markov Decision Processes* (Dec-POMDPs), and their close relatives, jointly observable Dec-MDPs. In this section, we outline the full formal model in detail, providing an example instance, and identify the key solution concept that applies to them, namely *joint policies*.

2.1.1 The Formal Dec-POMDP and Dec-MDP Models

The *decentralized partially observable Markov decision process* (Dec-POMDP) is a highly general and powerful framework, capable of representing a wide range of real-world problem domains. It extends the basic POMDP to multiple agents, operating in conjunction based on locally observed information about the world, and collecting a single source of reward.

Definition 2.1 (Dec-POMDP)**.** A *decentralized partially observable Markov decision process* (Dec-POMDP), $\mathcal{D}$, is specified by a tuple:

$$M = \langle \{\alpha_i\}, S, \{A_i\}, P, \{\Omega_i\}, O, R, T \rangle \tag{2.1}$$

with individual components as follows:

- Each α_i is an *agent*.
- S is a finite set of *world states* with a distinguished *initial state* s^0.
- A_i is a finite set of *actions*, a_i, available to α_i. A *joint action* is some tuple of individual actions, $\langle a_1, \ldots, a_n \rangle$, one per agent.
- P is the *Markovian state-action transition function*. $P(s, a_1, \ldots, a_n, s')$ is the probability of going from state s to state s', given joint action $\langle a_1, \ldots, a_n \rangle$.
- Ω_i is a finite set of *observations*, o_i, for α_i.
- O is the *joint observation function* for the set of agents, given each state-action transition. $O(a_1, \ldots, a_n, s', o_1, \ldots, o_n)$ is the probability of observing $\langle o_1, \ldots, o_n \rangle$, if joint action $\langle a_1, \ldots, a_n \rangle$ causes a transition to global state s'.
- R is the *global reward function*. $R(s, a_1, \ldots, a_n)$ is the reward obtained for performing joint action $\langle a_1, \ldots, a_n \rangle$ when in global state s.
- T is the (finite or infinite) *time-horizon* of the problem.

A Dec-POMDP has three main salient features, which characterize the class and contribute to its complexity. These are as follows:

Local observations lead to decentralization. Given any state-action transition, the observation function, O, assigns each agent its own observations on a probabilistic basis. While there may be probabilistic dependencies between the observations of different agents, these are not necessarily very informative, and in general each agent must act solely on the basis of its own local information.

Shared rewards leads to cooperation. The reward function, R, is a global one, and so agents who wish to maximize the objective will seek to coordinate.

Common state-space leads to interaction. There is a single environment underlying the system dynamics, with states that transition according to actions of all agents together and that govern the various rewards and observations received.

Together with the extended time-horizon over which agents must act, these features combine to create rich and complex problem-domains. The joint reward and necessity to plan over time distinguishes the problem from one-step competitive games, and means that intelligent planning methods must take multiple agents, and their joint actions over time, into full account.

In a general Dec-POMDP, the joint observations of agents are related to the underlying state in only a general probabilistic manner, and may provide little real information about that state, even taking all agent observations together. For instance, there is nothing in the definition that prevents all state-action transitions from leading to exactly the same joint observation. That is, we can specify a Dec-POMDP in which there exists some one observation-tuple, $\langle o_1^\star, \ldots, o_n^\star \rangle$, such that for any joint action, $\langle a_1, \ldots, a_n \rangle$, and state, s, $O(a_1, \ldots, a_n, s, o_1^\star, \ldots, o_n^\star) = 1$, and is zero (0) for all other combinations of observations.

In most cases, however, the joint observations provide more information than that about the underlying state.[1] The most important sub-instance of the Dec-POMDP model is thus the *decentralized MDP* (Dec-MDP), where this information is *perfect*, and knowing the joint observation tells us everything we need to know about the state into which the system has just transitioned.

[1]This is not to say that *individual observations* are very informative. We can also create problems for which the joint observations have strong probabilistic ties to the actual state, but some particular agent's own local information tells us next to nothing.

Definition 2.2 (Dec-MDP)**.** A *decentralized Markov decision process* (Dec-MDP) is a Dec-POMDP that is *jointly fully observable*. That is, there exists a functional mapping, $J : \Omega_1 \times \cdots \times \Omega_n \rightarrow S$, such that $O(a_1, \ldots, a_n, s', o_1, \ldots, o_n) \neq 0$ if and only if $J(o_1, \ldots, o_n) = s'$.

In a Dec-MDP, then, the sum total of the individual agent observations provides a complete picture of the state of the environment. It is important to note, however, that this *does not mean* that any individual agent actually possesses this information. Dec-MDPs are still fully decentralized in general, and individual agents, when choosing actions, cannot count on access to the global state. This distinguishes these problems from the *multiagent MDP* framework, in which agents work together in states that each observe equally and completely [19]. As we shall see, Dec-MDPs are considerably more interesting, and—somewhat unfortunately—far more complex.

2.1.2 An Example Instance

As we have discussed already, Dec-POMDPs have proven useful as models for a wide range of multiagent problem domains (Section 1.3.2). The general framework allows for many variations, given the wide range of possible relationships between states, actions, observations, and rewards. A simple example illustrates the sorts of decisions an agent may need to make in a decentralized environment, and how such a problem can be presented as a Dec-POMDP.

Experiments with automated, unmanned systems for detection, pursuit and evasion tasks [129] have generally allowed teams of those vehicles to share information freely among themselves over the course of their search or escape routes, for purposes of simplifying the problem. If such systems were deployed, however, security concerns would generally dictate that communication be kept to a minimum, to avoid divulging route and location details to the competing side. (Some work has attempted to create

security through policy randomization [97]; for the purposes of this example, however, we merely consider a traditional, non-random policy.)

Figure 2.1 shows an example of the sorts of decisions that might be necessary under such constraints. In this framework, two automated agents, a *Sensor* and a *Pursuer*, work together to track and capture a moving target. When the Sensor detects the trajectory of the target, it can use radio signals to relay this information to the Pursuer, who can then change course to intercept. However, there is a potential risk to communicating: there is some chance that the target intercepts communications between its pursuers and changes course, knowing that it is spotted. The question is then two-fold: (1) whether or not the Sensor should communicate any information it has gathered to the Pursuer; and (2) what the Pursuer ought to do given information received (if there is any).

The diagram illustrates the situation as follows. Circular nodes contain possible actions for the two tracking agents, taken in order of time from left to right. Square nodes contain values of variables for the courses taken by the target (CT) and the Pursuer (CP); we assume that these courses run either west (W) or east (E). Both agents begin in a starting state. At the second step, the Sensor detects the target trajectory ($CT = X$), and at the third, it can choose whether or not to communicate this fact to the Pursuer, as indicated by the dashed line connecting the agents.

The decision whether or not to communicate is complicated by the chance of detection, which we can model as uncertain, probabilistic effects of that decision. In order to determine whether communication is worthwhile, the decision-makers must take these effects into account.[2] The goal is to make the decision leading to the highest value for the tracking system, where this value is measured in terms of whether or

[2]Other ways of modeling such communication decisions as separate, distinct types of actions can be found in [85, 106]; trade-offs between communication and quality are also considered by Xuan, Lesser, and Zilberstein [136, 137].

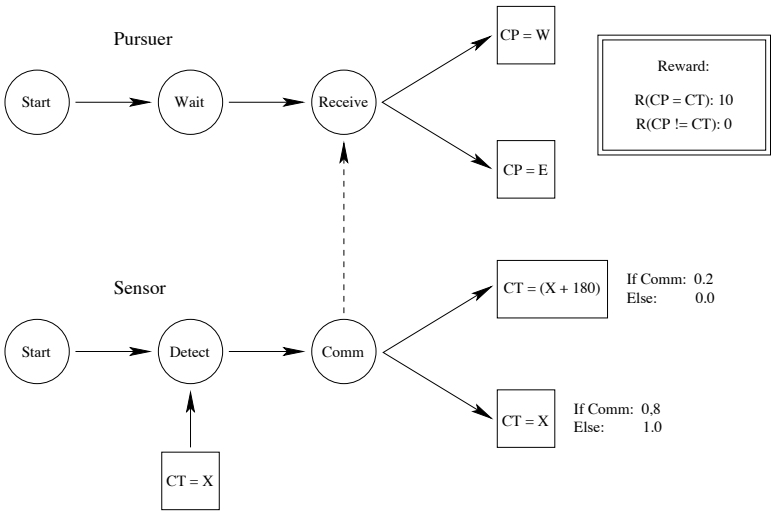


Figure 2.1: An example communication decision problem. *Sensor* and *Pursuer* agents must work together to track a target. The Pursuer must choose its own course (CP). The Sensor must choose whether or not to communicate the observed course taken by the target (CT); if it does, there is a 20% chance the target will alter course.

not the target is intercepted. The possible effects of communicating are given by the numbers along the right-hand edge of the diagram, indicating the probabilities of each outcome for the variables CT:

- If the Sensor does not communicate anything, then the target will remain on its current course ($CT = X$, probability 1.0).
- If the Sensor communicates with the Pursuer, there is a 20% chance the target intercepts the radio transmission and, realizing it has been detected, changes to the opposite trajectory ($CT = (X + 180°)$, probability 0.2); otherwise, the target continues on its original path ($CT = X$, probability 0.8).

Finally, rewards are given in the double box at upper right. As shown there, the system as a whole receives a reward of 10 units if the target is intercepted in the end ($CT = CP$), and a reward of 0 units otherwise ($CT \neq CP$). We can assume that all other states and actions lead to 0 reward. For simplicity, we let communication be cost-free in this instance.

This tracking example can be represented as a Dec-POMDP as follows.

(a) S: The global state at any time consists of the location of all mobile elements (Pursuer agents and targets) across the range of the sensor network, and their current trajectories, so that each state, $s \in S$, is a tuple of locations and course variables, one set per Pursuer/target.

(b) A: At each time-step, Sensor agents choose whether or not to broadcast information to Pursuers, and Pursuers choose their courses of motion, or can opt to capture a target whose present location they share.

(c) P: The transition function directs the situation according to actions taken; for instance, communication actions of Sensor agents have probabilistic effects upon the course of targets, as already described.

(d) R: The total reward is given as a function of whether or not individual targets are captured, and is zero (0) otherwise.

(e) Ω: Each Sensor agent observes some sub-region of the entire range of the network and detects target movement in that region; accordingly, its set of observations corresponds to either seeing a target (and its trajectory) or not, at each location in its range. Pursuers observe their immediate area, along with the proximal presence of targets, and their observation-sets correspond to possible locations and targets; in addition, since they can receive messages from Sensors, each possible observation includes the presence or absence of same.

(f) O: The observation function takes various target locations and courses, given by the state, and "parcels out" information to the various agents as appropriate.

(g) T: We may presume the problem has an infinite (effectively unlimited) horizon, with continuous tracking and pursuit of new targets over time.

This is but one example, and clearly the details could be altered to add further capacities, limitations, and complications. For instance, the communication cost-

function can be made non-zero, so that acts of information-sharing take into account factors like power usage; this can be modeled using the reward-function, R, adding a negative penalty to actions involving communication by the Sensor. Noisy communications can be accounted for by including the content of broadcast information as part of the local-state observation for individual Pursuer agents; communication actions by Sensors would then have some probability of sending incorrect or partial information. Similarly, the function O can be set to represent noisy or faulty sensors, providing only probabilistic guarantees about the actual course taken by a particular target. Sensor networks that are only expected to operate over fixed periods of time can be represented in models with finite time-horizons for planning purposes.

Even without such complications, this example already illustrates something of the complex nature of decentralized decision-making. When devising a policy governing what the Pursuer may do, clearly we must consider both the possibility that the Sensor communicates and that it does not, along with the information that may be communicated, and the effects of the communication as necessary. Furthermore, when determining whether or not the Sensor should communicate its observation, we must consider not only the effects of doing so on the target's trajectory, but also upon what the Pursuer will do in each case. Generating optimal courses of action for each agent requires considering not only possible local outcomes and state transitions, as in a single-agent decision problem, but also the set of possible policies for the other agent. Furthermore, as time passes, agents will accumulate an entire *history* of prior observations, which will provide more refined—or potentially more confounding—information about the current possible state of the system as a whole. As will be detailed below, it is this sort of ramified decision-making that contributes to the complexity of decentralized problems. Since agents will generally be unable to know the local states, observations, and actions of others, planning will often need to

take account of vast numbers of possibilities, as policies of action are prepared that maximize value over any and all contingencies.

2.1.3 Solutions to Decentralized Markov Processes

Solving a Dec-POMDP or Dec-MDP amounts to generating a *policy* for each agent, a mapping between their local observations and their possible actions. In fact, we will want these mappings to be functions of *observation histories*, since partial observability means that agents may gather increasing amounts of information about the current state over time (just as in single-agent POMDPs).

Definition 2.3 (Policies)**.** A *local policy* for an agent α_i is a mapping from sequences of that agent's observations, $\overline{o_i} = \langle o_i^1, \ldots, o_i^k \rangle$, to its actions, $\pi_i : \Omega_i^\star \rightarrow A_i$. A *joint policy for n agents* is a collection of local policies, one per agent, $\pi = \langle \pi_1, \ldots, \pi_n \rangle$.

In general, then, a solution method for a decentralized problem will seek to find some joint policy that maximizes expected value given the starting state (or distribution over states) of the problem. In order to evaluate policies, we must first define the probability of a given global state-transition, given a series of observations and a particular joint course of action.

Definition 2.4 (Transition-Probability for π)**.** Given a joint policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ and a length-m observation sequence $\overline{o}_i = \langle o_i^1, \ldots, o_i^m \rangle \in \Omega_i^\star$ for each agent α_i, we can define the *transition-probability for* π between states s and s', designated by $P^\pi(\cdot)$, recursively as follows. Let ϵ be the empty observation-sequence.

$$P^\pi(s, \underbrace{\epsilon, \ldots, \epsilon}_{\times n}, s) = 1.$$

$$P^\pi(s, \overline{o}_1 o_1, \ldots, \overline{o}_n o_n, s') = \sum_{s'' \in S} P^\pi(s, \overline{o}_1, \ldots, \overline{o}_n, s'') \times P(s'', \pi_1(\overline{o}_1), \ldots, \pi_n(\overline{o}_n), s') \times O(\pi_1(\overline{o}_1), \ldots, \pi_n(\overline{o}_n), s', o_1, \ldots, o_n).$$

Based upon these policy-driven transitions, we can compute the expected value of a policy, starting from a given state. For finite-horizon policies, this is straightforward.

Definition 2.5 (Value of a finite policy)**.** The *value of following a policy* $\pi = \langle \pi_0, \ldots, \pi_n \rangle$ for finite number of steps T, beginning in some state s, is as follows:

$$V^{\pi}(s) = \sum_{\langle \overline{o}_1, \ldots, \overline{o}_n \rangle} \sum_{s' \in S} P^{\pi}(s, \overline{o}_1, \ldots, \overline{o}_n, s') \times R(s', \pi_1(\overline{o}_1), \ldots, \pi_n(\overline{o_n})), \tag{2.2}$$

where each tuple of observation-sequences $\langle \overline{o}_1, \ldots, \overline{o}_n \rangle$ is such that every component sequence $\overline{o}_i$ is of the same length t, for any value $t \in [0, T)$.

Given a Dec-POMDP $\mathcal{D}$, with a finite time-horizon T, and distinguished starting state $s_0 \in S$, a *solution* for $\mathcal{D}$ is thus a T-step policy, π. An *optimal solution for* $\mathcal{D}$ is a T-step policy, $\pi^{\star}$, that maximizes expression (2.2).[3]

For infinite-horizon policies, things are somewhat more complicated, due to the fact that the naive definition (i.e., simply replacing T in Definition 2.5 with ∞) means that the expected value from any state is infinite whenever we have strictly positive rewards, and so all policies are identical. In practice, this is usually dealt with by concentrating on *discounted reward* over time, employing some weighting factor $\gamma \in (0, 1)$ to render rewards negligible over suitably long time-horizons. Since our work here will focus almost entirely on the finite-horizon case, we leave this aside for now. The thesis of Bernstein [15] covers the subject in detail.

2.2 Complexity of Dec-POMDPs

Here we outline the difficulties inherent in decentralized control problems. As it turns out, such problems are generally intractable, and present many practical

[3]When $\mathcal{D}$ does not have a distinguished starting state, but rather an initial distribution over states in S, the optimal policy is that one maximizing expected value over all possible starting states, given that distribution.

problems when it comes to exact solutions. Furthermore, approximate solutions to Dec-POMDPs also turn out to be extremely difficult, making the landscape even more challenging.

2.2.1 Worst-Case Theoretical Complexity

The fundamental touchstone here is the result of Bernstein, et al. [16], showing that the decision variant of finite horizon Dec-POMDPs (and Dec-MDPs) is complete for nondeterministic exponential time (NEXP), even with only two (2) agents. (The infinite-horizon version is undecidable, a result that follows directly from the fact that the same holds for single-agent POMDPs.) The essential hardness portion of the proof is based on a reduction from the tiling problem (see Lewis [75]). While we will not go into the details here, we outline some of its important implications.[4]

NEXP-hard problems possess extremely high complexity. To see how this is so, we consider the more familiar class NP (nondeterministic polynomial time). Simply put, problems in this class are such that, once a candidate solution has been presented, it can be verified in time that is polynomial in the size of the original problem specification. Nondeterminism, here, enters the picture in the process whereby the candidate solution is presented for verification: the "NP algorithm" essentially amounts to choosing some solution non-deterministically, and then performing verification. In practice of course, nondeterministic machines do not exist, and so an NP solution algorithm is in fact given simply in terms of the exact verification procedure.

Genuinely NP-hard problems, then, do not have feasible polynomial algorithms for *finding* solutions, only for *checking* them. Furthermore, these problems have a possible number of solutions that is exponential in the size of the given instance. Thus, the only known method that could possibly solve an NP-hard problem is one

[4]Later, in Chapter 3, we outline the proof in more detail, as we make use of variations on the technique to establish some novel complexity results.

that enumerates and then verifies each of the exponentially-many solutions one by one. Barring the reduction of NP to P—the discovery of a polynomial-time algorithm for actually finding solutions—such problems are considered to have worst-case exponential (EXP) time complexity for existing deterministic machines.[5]

Thus, NP-hard problems are intractable in the worst case, unless of course P = NP, in which case the problems are in fact polynomial, since now we can generate and verify solutions polynomially. With this in mind, the difficulty of NEXP-hard problems is evident. For such problems, once a candidate solution is presented, verification of that candidate requires worst case time that is *irreducibly exponential* in the size of the original problem. Thus, a serious bottleneck exists, even if nondeterminism were not an issue. At best, even if we had the complexity collapse NEXP = EXP (the best that can be possible, and not to be expected in any case), such problems would remain exponentially difficult. In the absence of such a collapse, practical complexity would in fact seem to be *doubly exponential*, since the only generally optimal method must search through an exponentially large space of potential solutions as before, *each of which* then requires an exponential amount of time to verify. The upshot is therefore that NEXP-hard problems, like Dec-POMDPs, can be expected to be largely intractable.

Thus, finite-horizon Dec-POMDPs and Dec-MDPs are extremely difficult. Furthermore, approximation is also highly complex. Many intractable problems have efficient algorithms for *bounded approximation*. That is, given some factor ϵ, it is possible to find a solution that it is within ϵ of optimal, using time that is polynomial in

[5]Strictly speaking, for a complexity class C, saying that a problem is *C-hard* is only to place a lower bound on its difficulty. That is, when we say something is NP-hard, or NEXP-hard, this does not rule out the possibility that the problem is actually harder than that. When we say a problem is *in the class C*, this puts an upper bound on complexity, meaning that it can be solved by an algorithm for that class, and may in fact be easier. Finally, when we say that a problem is C-complete, we have determined, among other things, that it is bounded above and below by C. Thus, to say that Dec-POMDPs are NEXP-complete means that they are in fact NEXP-hard in general, and no harder.

a combination of that approximation factor and the problem size. For Dec-POMDPs, this is unfortunately not the case, and as shown by Rabinovich, Goldman, and Rosenschein [107], ϵ-approximation is also NEXP-complete. As a result, approximation is just as hard as optimal solution, and bounded methods are also generally impractical. As we will discuss below, this means that many algorithms can only provide rough guarantees about the optimality of solutions, or provide no bounds at all.

2.2.2 Comparison with Similar Problems

Papadimitriou and Tsitsiklis [93] show that single-agent MDPs with a finite time horizon are P-complete. Strictly speaking, such problems are in fact *pseudopolynomial*, since the solution time is polynomial in the size of the problem description only if the time horizon is specified in non-compact unary form.[6] However, given the general restriction that the time horizon is sufficiently smaller than the size of the problem state-space ($T \lll |S|$), the problem class is strongly polynomial. In the infinite-horizon MDP case, value and policy iteration algorithms (for an overview, see Sutton and Barto [123]) converge to optimal solution policies in polynomial time, and linear programming techniques also provide polynomial solutions (see Kallenberg [67]).

Papadimitriou and Tsitsiklis [93] also show finite-horizon single-agent POMDPs to be PSPACE-complete. More recently, Lusena, et al. [79] have shown that unless we have P = PSPACE (which is unlikely), there is no polynomial time ϵ-approximation possible for POMDPs. In the infinite-horizon case, as shown by Madani, et al. [81], POMDPs are in fact undecidable, and have no finite general solution technique. (Intuitively, since policies for POMDPs are maps from possible observation-histories to

[6] Intuitively, such problems may require an iterative approach, so that the solution will require one step for each moment of time. Even if the amount of processing done per iteration is polynomial, in the size of the problem specification, a compact, logarithmic representation of a sufficiently long time-horizon would still mean that the overall time must be exponential in the problem size. This problem disappears if the unary representation of the time is used, since the size of the instance keeps pace with the iterations required.

actions, the possibility of infinitely-long histories leads to the undecidability result.) This of course implies that Dec-POMDPs are undecidable when the time horizon is infinite, as well.

The NEXP-completeness of finite-horizon decentralized MDPs and POMDPs is thus very strong, showing a significant increase in complexity over their single-agent, centralized relatives. The result also significantly strengthened known complexity properties for decentralized problems. Initially, Papadimitriou and Tsitsiklis [91] were able to show that a similar problem, the *static team decision problem* (STDP) of Radner [108], was NP-complete for the finite-horizon and single-agent case. They later pointed out that this implies the NP-hardness of Dec-POMDPs, as well [92]. Unfortunately, their actual complexity is much worse than this lower bound.

When agents can communicate freely and instantaneously with one another, and have vocabulary sufficient to express all their actions and local states (observations), it is easy to see that Dec-MDPs (Dec-POMDPs) reduce to multi-agent MDPs (multi-agent POMDPs), and are thus solvable as equivalent single-agent cases. In many such cases, even if agents begin without a fixed pre-understood communication protocol, they can in fact learn to communicate and coordinate properly, making the problems tractably solvable, as shown by Allen, Goldman, and Zilberstein [1, 43, 44, 45]. However, whenever communication is not cost-free—as indeed it is generally not—the decision about what and when to communicate must also be considered, and the overall problem is identical to the general Dec-POMDP. Goldman and Zilberstein [43] give an overview of complexity results for many subproblems, especially those involving various forms of communication.

2.3 Exact Algorithms for Finite-Horizon Dec-POMDPs

As noted, the NEXP-completeness of general Dec-POMDPs means that algorithms for solving such problems optimally can face doubly-exponential growth in

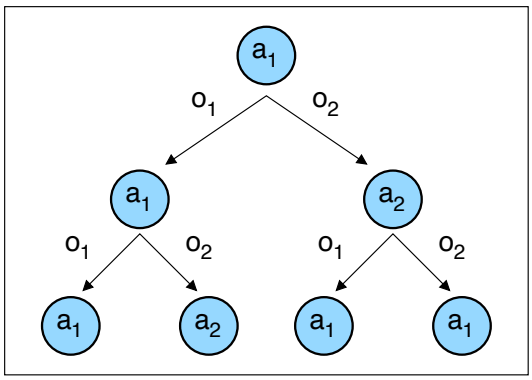


Figure 2.2: A simple policy-tree, showing a three-step policy for a domain with two possible observations (o_1 and o_2).

necessary space and time, rendering even simple problems potentially intractable. In some computational domains, such theoretical *worst-case* complexity does not reflect the *average-case* difficulty as encountered in practice, and many problem instances are in fact solvable using polynomial (or better) amounts of computational resources. In the case of Dec-POMDPs, however, this has not seemed to be the case. Existing research has primarily focussed upon the basic feasibility of running exact and approximate algorithms, and success has been measured in terms of the sizes of the sets $\{\alpha_i\}$, S, $\{A_i\}$, and $\{\Omega_i\}$ for the largest problems solved, along with the maximum finite time-horizons for which policies can be generated. Nothing systematic has been determined about the relationships between problem solutions and these values, but suffice to say that all existing cutting-edge methods solve only very small problem instances (where, for instance, we have but two agents and only a handful of possible actions and observations for each of them).

Part of the problem is the sheer profusion of possible policies. As we have already explained, a joint policy for a Dec-POMDP $\mathcal{D}$ is a tuple of individual agent-policies, where each of these is a mapping from sequences of observations to actions. One convenient means of representing such mappings in the finite-horizon case is in the form of *policy-trees*, as shown in Figure 2.2. Such a tree-structure consists of nodes

Horizon (t)	Policies (1 agent)	Joint policies (2 agents)
1	2	4
2	8	64
3	128	16,384
4	32,768	1,073,741,824
5	2,147,483,648	$4.612 \cdot 10^{18}$

Table 2.1: Policy-tree explosion: the number of possible policies for one or two agents, with 2 actions and 2 observations each.

labeled with action-choices, and branches labeled with possible observations; for a given agent α_i, each such tree will thus have a branching factor equal to the number of possible observations, $|\Omega_i|$, and a depth equal to the horizon of the policy. Thus, in our example, we see a three-step policy for an agent with two possible observations, o_1 and o_2. Each path down the tree from root to leaf then corresponds to an observation-sequence, and an agent pursues the policy by taking the root-action, receiving the ensuing observation, and following the associated branch to the next action, where the process is repeated as required.

Given such a representation, it is easy to compute the number of possible horizon-t policy-trees for an agent α. From the root node, with depth 0, to the leaves at depth $t-1$, we must choose labels for each node at that level; given the branching factor $|\Omega|$, the number of choices at some depth d is evidently $|\Omega|^d$. Therefore, given that there are $|A|$ choices for each action-label, the number of of possible distinct trees for the agent is then $|A|^{1+|\Omega|+|\Omega|^2+\cdots+|\Omega|^{t-1}} = |A|^{\frac{|\Omega|^t-1}{|\Omega|-1}}$, which is exponential in the number of possible observations, $|\Omega|$, and doubly so in the time-horizon, t. Finally, for an n-agent problem (assuming, without loss of generality, that all agents have the same number of possible actions and observations), the set of all n-agent, horizon-t joint policies is additionally exponential in n: $(|A|^{\frac{|\Omega|^t-1}{|\Omega|-1}})^n$, which is $\mathcal{O}(|A|^{n\cdot|\Omega|^t})$. The exact values for one and two agents, with only two actions and two observations each,

are given in Table 2.1, which makes the point clear: exhaustive enumeration and evaluation of policies for Dec-POMDPs is simply infeasible.

Algorithms for solving finite-horizon Dec-POMDPs optimally have tried to get around the problem of policy profusion in a number of ways. The first known exact solution algorithm for general Dec-POMDPs, was devised by Hansen, et al. [59], and is explained in detail below. Briefly, it uses dynamic programming (DP), working bottom-up to generate the finite-horizon policies from the leaves to the root; to mitigate the full combinatorial explosion, iterated pruning techniques are used to reduce the number of policies that must be considered overall. However, such improvements cannot generally make the problem tractable. While this of course follows by the complexity results, it is also evident in practice: even for a very simple problem, the method often cannot generate policies beyond a handful of time-steps. Similar results have been reported with respect to other known optimal methods, including those that proceed top-down, using heuristic search; again, as Szer, Charpillet, and Zilberstein [124, 126] have pointed out, only the smallest problems can be solved. A good overview of these results can be found in Seuken and Zilberstein [113, 116].

2.3.1 Outline of the Dynamic Programming Algorithm

Hansen, et al. [59] base their Dynamic Programming (DP) method for finite-horizon Dec-POMDPs on Hansen's own method of DP for single-agent POMDPs [58]. Incremental policy search and pruning are combined with the iterated elimination of dominated strategies, in a manner analogous to some work in game theory. While the algorithm cannot escape the basic worst-case hardness results for the problem class, the intention is that pruning of dominated strategies will sometimes reduce the search space of policies enough to allow us to solve more complex problems.

Since we use this method in some of our empirical results later on (Chapter 6), we go over it in detail here. Given finite time horizon t and n agents, the solution is

represented as a sequence of depth-t policy-trees, $\pi^t = \langle \pi_1^t, \ldots, \pi_n^t \rangle$, where each such tree π_i^t has labeled action-nodes and observation-branches, as described above. The sum of expected rewards under each of the component policy-trees gives the overall value of joint policy π^t.

These individual agent policy-trees are built in a bottom-up fashion, and the algorithm tries to avoid doing a full backup—creating the full space of trees—by selecting policies to prune away as it goes. (In effect, at each iteration, each agent keeps only those policies that are *best responses* to some possible policy of another.) The set of depth-$(t+1)$ policy-trees in π^{t+1} is built by adding all possible action/observation-transitions onto depth-t trees from π^t. Therefore, the elimination of some tree $\pi^\star$ at any iteration means that all possible later trees that would include $\pi^\star$ as a subtree will never actually be considered. Early pruning of trees thus promises exponential reduction of necessary work later on.

Each DP iteration consists of the following steps:

Input. For each agent α_i, a set Π_i^t of depth-t policy-trees π_i^t, with an associated set of value vectors $\mathcal{V}_i^t$.

Backup. Generate the set of next-depth trees $\pi_i^{t+1} \in \Pi_i^{t+1}$, with value vectors $\mathcal{V}_i^{t+1}$.

Pruning. Trees $\pi_i^{t+1} \in \Pi_i^{t+1}$ are pruned, given value vectors of other agents V_j^{t+1}.

Output. Sets of trees Π_i^{t+1} and vectors $\mathcal{V}_i^{t+1}$ for each agent α_i.

We give more detail of these various stages and their requirements below.

Backup and value calculation

The DP program is given a set Π_i^t of depth-t policy-trees for each agent α_i. Each agent also possesses a set $\mathcal{V}_i^t$ of value vectors, one for each tree $\pi_i^t \in \Pi_i^t$. For a Dec-POMDP $\mathcal{D}$ with state-set S, and letting $\Pi_{-i}^t = \{\Pi_j^t \mid j \neq i\}$ be the set of trees for other agents, each value vector $v_k \in \mathcal{V}_i^t$ is of dimension $|S| \times \left|\Pi_{-i}^t\right|$, and gives the value

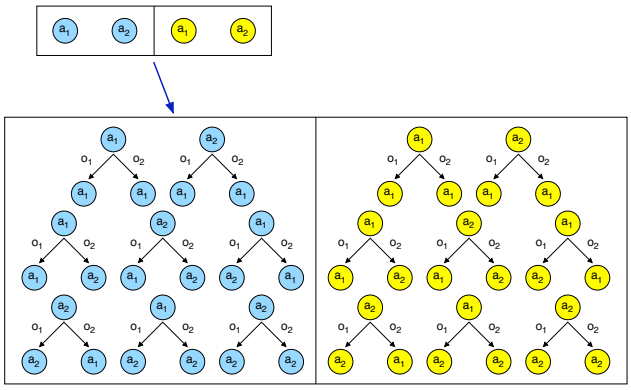


Figure 2.3: Expansion of the set of policy-trees from Π^1 to Π^2 for each of two agents. Each larger tree (right-hand side) consists of a root-node with one possible action label, and with observation-branches leading to one possible permutation of the trees from the prior step (left-hand) side.

of following policy-tree $\pi_k \in \Pi_i^t$, for each possible start-state $s \in S$ and sequence of policy-trees for other agents $\langle \pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n \rangle \in \Pi_{-i}^t$.

Based on the transition, observation, and reward models of the Dec-POMDP, the algorithm then exhaustively generates the next set of trees Π_i^{t+1} for each agent. Figure 2.3 shows the full expansion of the set of policy trees for a pair of agents, from initial horizon ($t = 1$, where the trees are just possible action leaf-nodes) to the next time-step, again for the 2-action, 2-observation case. Once Π_i^{t+1} has been generated for each i, the next-stage value vectors $\mathcal{V}_i^{t+1}$ are also calculated. Note that each such vector will now be of higher dimension than before, rising to size $\left|\mathcal{S} \times \Pi_{-i}^{t+1}\right|$.

Iterated pruning

The value of a policy-tree for agent α_i is determined by that agent's belief about the state of the environment and about the possible policies being followed by other agents. This belief is represented as the set of probability-distributions over this space of possibilities, $\Delta(S \times \Pi_{-i}^t)$. We then want to prune policy-trees which are *dominated* relative to any possible belief, in the sense that some other policy-tree does just as

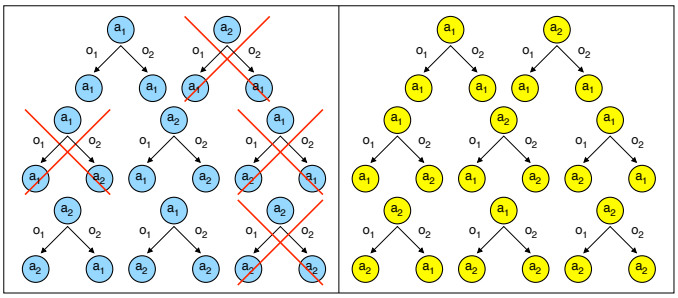


Figure 2.4: Policy-trees are pruned iteratively, beginning from the set created by the full backup from the prior step. First, we eliminate policies for the first agent, keeping only those that are dominated by others, given any possible policy of the second agent; then, we do the same for the second. Since the reduction in the second set may mean that further policies can be eliminated from the first, the process is repeated until convergence.

well or better against any of the other agents' current possible policy-trees. In order to prune each policy-set Π_i^{t+1}, the DP algorithm iterates as follows:

1. Choose agent α_i and find policy-tree $\pi_k \in \Pi_i^{t+1}$, with value vector v_k, such that:

$$\forall \delta \in \Delta(S \times \Pi_{-i}^{t+1}),\ \exists v_m \in (\mathcal{V}_i^{t+1}/v_k) \text{ s.t. } \delta \cdot v_m \geq \delta \cdot v_k.$$

2. Set $\Pi_i^{t+1} \leftarrow (\Pi_i^{t+1}/\pi_k)$.
3. Set $\mathcal{V}_i^{t+1} \leftarrow (\mathcal{V}_i^{t+1}/v_k)$.

That is, we trim out any policy that is less valuable than all others, given any possible belief. (This is computed in an efficient manner using linear programming.) Such a policy is eliminated from the policy-set, and its associated value vector is eliminated along with it, since we will no longer have to consider the value of that policy. This process is carried out iteratively for each agent α_i in turn, as shown in Figure 2.4. On the left-hand side of the diagram, we see that some policy-trees for one agent are trimmed away, since each is dominated by some other policy in the set. Then, on the right-hand side, the policy-trees of the second agent are trimmed,

where their value are calculated relative only to those possible policies of the first agent that remain. The process is then repeated: since the second agent has now trimmed its possible policies, too, the first may now find that some of its remaining policies are no longer necessary, since the target to which they are a best response no longer exists. The algorithm continues looping through policies of each agent, until it converges, and no more policies are eliminated.

Output policy

At the end of one iteration, the DP algorithm has generated a set of depth-k policy-trees for each agent, comprising all of those courses of action that may prove valuable under some possible belief. The process repeats until the time-horizon T is reached. The result is then a collection of depth-T policy-trees, each of which is provably optimal for some initial belief-distribution over possible initial states of the Dec-POMDP. Then, given each agent's starting belief about what that state may be, we can calculate the value of each policy-tree relative to that particular belief. The final output is then simply a sequence of depth-T policy-trees, $\pi^t = \langle \pi_1^t, \ldots, \pi_n^t \rangle$, each maximizing value for the given agent, and comprising an optimal solution-policy.

2.4 Approximate DP for Finite-Horizon Dec-POMDPs

In Chapter 6, we also examine how the performance of some approximate dynamic programming methods relates to defined measures of agent interaction. We thus describe these methods here.

2.4.1 Memory-Bounded Dynamic Programming

Seuken and Zilberstein [114, 115] present an extension of the basic DP method that places hard bounds on the number of policy-trees retained in each phase of the bottom-up construction. In normal DP, many trees are retained during construction that will turn out to be sub-optimal, once the full horizon has been reached, and

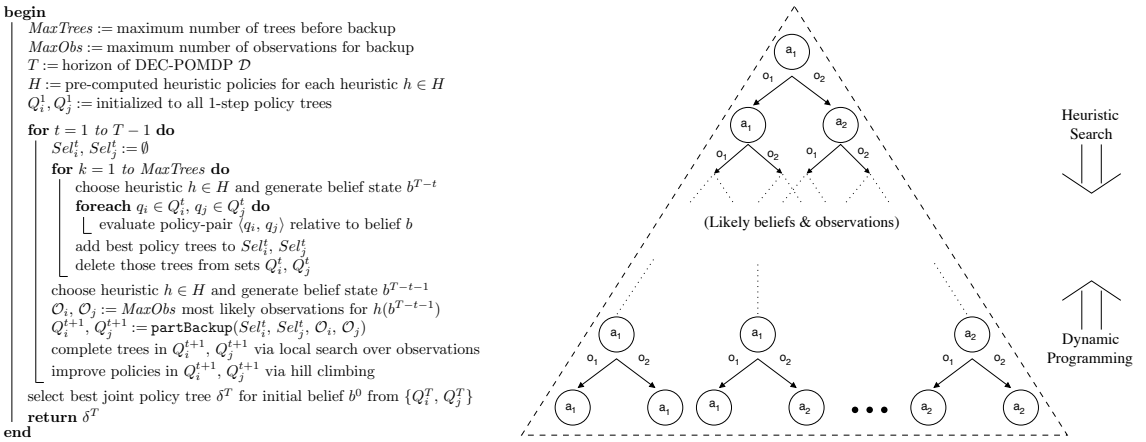


Figure 2.5: The Improved Memory-Bounded dynamic programming (IMBDP) algorithm combines the bottom-up policy-tree backup of regular DP with top-down heuristic search to find relevant beliefs and most likely observations. These are then used to prune down to a fixed maximum number of trees per agent at each iteration.

the various possible starting states are considered. Intermediate-stage beliefs about the possible policies of other agents, which determine whether a given policy-tree is dominated and can be eliminated, often turn out to be ill-founded. Thus, the *memory-bounded dynamic programming* (MBDP) method employs top-down heuristics, such as the assumption that the problem becomes a fully observable MDP after each step, to work top-down in an efficient manner, eliminating sub-trees that are in fact not reachable given the distribution over initial states. Combined with the parameter constant $maxTrees$, which restricts how many policy-trees are ever retained after pruning, the set of policies in original MBDP grows on the order of $|A| \times maxTrees^{|\Omega|}$. The algorithm performance is then linear with respect to the time horizon T, but still features exponential dependence on the number of possible observations.

To deal with this problem, the *improved MBDP* (IMBDP) algorithm also caps the maximum number of observations considered. As before, when considering sub-trees that begin at some time step $t \in T$, the top-down heuristics are used to generate a set of possible belief-states, and for each such state, the best policy-tree is selected.

Rather than doing a full DP backup over all such policies, however, a strictly partial backup is performed, using a set of *most likely observations*. This, for b^{t-1}, the most likely belief-state for horizon step $t-1$, and best joint action $\overline{a}$ (derived according to the heuristic model), we calculate the probability of any joint observation $\overline{o}$ as follows:

$$p(\overline{o}) = \sum_{s \in S} b^{t-1}(s) \times O(\overline{o} \,|\, \overline{a}, s).$$

Then, using a pre-set bound on total observations $m = maxObs$, the m most likely observations are used for backup of the policy-trees (with missing observation branches filled in heuristically using local search techniques). Figure 2.5 shows the algorithm, and its combination of bottom-up policy-tree backup of regular DP with top-down heuristic search to find relevant beliefs and most likely observations.

Seuken and Zilberstein [114] are able to provide a bound on the error introduced by the process that is quadratic in the time horizon T, and decreases as $maxObs$ grows, becoming 0 when $maxObs = |\Omega|$, and all observations are used in backing up trees as in original DP. Further, the algorithm is now polynomial in all parameters. Their empirical work establishes that the algorithm can solve problems that are much larger than can be managed using optimal methods of any known kind, with time-horizons that are an order of magnitude longer. However, there is still rapid growth in time and space requirements as $maxTrees$ and $maxObs$ increase. Thus, in Chapter 6, we show that this performance can be tuned using our defined measures of agent influence (Chapter 5). In particular, as problems grow more centralized, the memory-bounding parameters $maxTrees$ and $maxObs$ can be set lower while still retaining some assurance of quality approximation, allowing for far better solution times.

2.4.2 MBDP with Observation Compression

Carlin and Zilberstein [23] expand upon the previous work, taking a different approach to the elimination of observations. Based on IMBDP, the method uses the

same combination of bottom-up DP and top-down heuristic belief-state generation; as well, the number of policy trees retained for backup is bounded by the same fixed $maxTrees$ parameter. In original IMBDP, we recall, the $maxObs$ parameter bounds how many of the most likely possible observations are retained as well. Unlikely observations are pruned out, and the sub-tree branches under them are later filled in using heuristic local search techniques; therefore, policy-trees still retain their full branching factor. In the *observation compressions* (OC) variant, on the other hand, the algorithm will in fact reduce this branching factor by merging certain observations together, collecting them into a set so that all of them lead to the same policy-tree branch. The parameter $maxObs$ is thus used to bound the number of merged sets.

In order to decide which observations should be merged, the algorithm computes which possible combinations lead to the least loss in expected value. Illustrating this point for the two-agent case, and letting $q_i \in Q_i$ be a policy-tree for agent α_i, we can use basic POMDP methods to compute $V(q_i \,|\, q_j, b, a_i, a_j, o_i, o_j)$, which is the value of continuing to follow joint policy $\langle q_i, q_j \rangle$, after having taken joint action $\langle a_i, a_j \rangle$ and receiving joint observation $\langle o_i, o_j \rangle$ in belief-state b. The best policy for α_i, $q_i^\star$, is then the policy-tree q_i that maximizes this expression. We can then compute three quantities to determine what is sacrificed by following some other, non-optimal policy:

$$L_i(q_i \,|\, q_j, b, a_i, a_j, o_i, o_j) = V(q_i^\star \,|\, q_j, b, a_i, a_j, o_i, o_j) - V(q_i \,|\, q_j, b, a_i, a_j, o_i, o_j)$$

$$WL_i(q_i \,|\, b, a_i, \{o_i\}) = \sum_{o_i \in \{o_i\}} \max_{a_j \in A_j} \sum_{o_j \in \Omega_j} \max_{q_j \in Q_j} O(o_i, o_j \,|\, b, a_i, a_j)\, L_i(q_i \,|\, q_j, b, a_i, a_j, o_i, o_j)$$

$$WL_i^\star(Q_i \,|\, b, a, \{o_i\}) = \min_{q_i \in Q_i} WL_i(q_i \,|\, b, a_i, \{o_i\}).$$

That is, $L_i(\cdot)$ computes the value lost by pursuing policy q_i rather than the optimal $q_i^\star$. Then, $WL_i(\cdot)$ is the expected total loss for merging the set of observations $\{o_i\}$ and following the same policy for each, while $WL_i^\star(\cdot)$ is simply the minimum of these.

```
begin
    MaxTrees := maximum number of trees before backup
    MaxObs := maximum number of observations for backup
    T := horizon of DEC-POMDP 𝒟
    H := pre-computed heuristic policies for each heuristic h ∈ H
    Q_i^1, Q_j^1 := initialized to all 1-step policy trees
    A_i, A_j := the set of actions available to agents α_i, α_j
    for t = 1 to T do
        choose heuristic h ∈ H and generate belief state b
        for each agent α_i do
            Q_i^{t+1} := ∅
            for each action a ∈ A_i do
                Z := compressObs(α_i, b, a, Q_i^t, Q_{-i}^t, MaxObs)
                Q_i^{t+1} := Q_i^{t+1} ∪ partBackup(Q_i^t, a, Z)
        Sel_i^{t+1}, Sel_j^{t+1} := ∅
        for k = 1 to MaxTrees do
            choose heuristic h ∈ H and generate belief state b
            foreach q_i ∈ Q_i^{t+1}, q_j ∈ Q_j^{t+1} do
                evaluate each policy-pair (q_i, q_j) relative to b
            add best policy trees to Sel_i^{t+1}, Sel_j^{t+1}
            delete these trees from Q_i^{t+1}, Q_j^{t+1}
        Q_i^{t+1}, Q_j^{t+1} := Sel_i^{t+1}, Sel_j^{t+1}
    select best joint policy tree δ^T from {Q_i^T, Q_j^T}
    return δ^T
end
```

(a) The IMBDP-OC algorithm.

```
input  : Agent α_i, belief state b, root action a_i, subpolicies Q_i and Q_{-i},
         and observation bound MaxObs
output: 𝒪 ≡ a set of tuples ⟨{o_k}, ε_k⟩, where each {o_k} is a set of
         observations, and ε_k is the error introduced by merging that set
begin
    𝒪 := {⟨{o_1}, 0⟩, ⟨{o_2}, 0⟩, ...}
    for each observation o_k, each policy q_i,
    and each q_{-i} ∈ Q_{-i} & a_{-i} ∈ A_{-i} for the other agent do
        Precompute V(q_i | q_{-i}, b, a_i, a_{-i}, o_k, o_{-i})
        Keep track of best policies q_i^*
    while |𝒪| > MaxObs do
        ({o'_1}, {o'_2}) := argmin_{⟨{o_1},ε_1⟩,⟨{o_2},ε_2⟩∈𝒪} WL*(Q_i | b, a_i, {o_1}∪{o_2})
        ε := WL*(Q_i | b, a, {o'_1} ∪ {o'_2})
        𝒪 := 𝒪 − {⟨{o'_1}, ε_1⟩, ⟨{o'_2}, ε_2⟩} ∪ ⟨{o'_1} ∪ {o'_2}, ε⟩
    return 𝒪
end
```

(b) The `compressObs` method.

Figure 2.6: The IMBDP algorithm with observation compression. The method for compressing the observations into merged sets is also included.

At every backup, then, the IMBDP-OC algorithm divides observations in a way that minimizes the expected loss, under the constraint that at most *maxObs* sets be employed. Figure 2.6 presents the main method, and the `compressObs` routine for merging observations into combined sets.

Carlin and Zilberstein [23] show that when $maxObs \geq maxTrees$, the OC variant constructs the same best available policy-tree as original MBDP. In the worst case, the algorithm loses some fixed amount of value per iteration, and has running time that is again polynomial in all parameters. Their empirical investigation suggests that there is some small time penalty for adding observation compression, but overall runtime is comparable for IMBDP and their variant. In domains where there is some penalty for ignoring improbable observations, the original algorithm runs the risk of poor performance, and the OC variant does considerably better, since it does not completely eliminate such observations. On the other hand, where less-probable observations can safely be ignored, as they never arise in the course of actual optimal

courses of action, the algorithms are relatively comparable in terms of the values of policies generated. Our work in Chapter 6 shows that performance is relatively similar over a broad range of problems; again, the measures of influence defined in Chapter 5 correlate strongly with policy value under IMBDP-OC, and again allow us to set the bounding parameters intelligently.

2.5 Other Algorithmic Techniques

While optimal methods for Dec-POMDPs exist, in most cases only the very smallest such problems can actually be solvable practically. Generally, the amount of plan-space pruning afforded by a DP or heuristic search method is simply insufficient, and the number of policy trees under consideration explodes unmanageably after only very few iterative steps. Bernstein et al. [14, 13] have extended these methods, and investigate policy iteration methods for infinite-horizon problems as well. Recently, Aras *et al.* [5] have presented a mixed-integer mathematical programming method that solves Dec-MDPs via a sequence-form reduction. Nevertheless, these algorithms are primarily of theoretical interest, and have not significantly increased the practical scope of problem-solving. No efficient and implementable techniques exist that can solve the sorts of complex problems we are interested in here.

As a result of this fundamental difficulty, researchers have proposed a number of approximate solution algorithms. While we discussed two variants of DP that bound the memory used when seeking solutions, there are a number of other candidates. For example, Nair et al. [86] present the Joint Equilibrium-based Search for Policies (JESP) algorithm, which finds a *locally*-optimal joint solution (i.e. one that is optimal for individual agents considered on their own). While this is interesting, it presents difficulties for applications where systems can only be judged by their success or failure on the whole. In sensor networks, for instance, it would be hard to accept a

plan of action that allowed individual sensors to "succeed" without actually producing a coordinated tracking effort.

Peshkin et al. [98] study how to approximate the decentralized solution using on-line learning when the agents do not know the model of their environment. Schneider et al. [112] assume that each decision-maker is assigned a local optimization problem, and show how to approximate the global optimal value function when agents may exchange information about their local values at no cost. Neither convergence nor bounds have been established for this approach. Wolpert et al. [134] assume that each agent runs a predetermined reinforcement learning algorithm, transforming the coordination problem into one of updating local reward functions so as to maximize the global reward function. Again, this is an approximation algorithm for on-line learning that does not guarantee convergence. Guestrin et al. study off-line approximations, whether centralized [51] or distributed [50], where known dependencies between agents' actions induce a message-passing structure. In this context, agents choose their actions in turns and communication is again presumed to be free, limiting the work's applicability. Finally, Amato et al. [2, 4] study linear and non-linear programming techniques for generating fixed-size finite-state controllers for Dec-POMDPs. Such controllers, while at least locally optimal for their fixed size, yield generally sub-optimal solutions; some empirical work suggests that these are at least better approximations to the optimal than other approaches mentioned here.

CHAPTER 3

COMPLEXITY OF RESTRICTED MODELS

Given the difficulty of the general Dec-POMDP problem, much attention has been paid to special cases, where some restrictions are placed on the problem definition. In doing so, the hope is that we can reduce problem complexity, while still representing problems of real interest. Here, we concentrate on some special models that restrict the ways in which agents can possibly interact in a system. In some cases, this has been shown to reduce problem complexity, and specialized algorithms have been devised. In others, less has been shown about essential hardness.

In addition to reviewing some of the known results, we prove some new ones, and by doing so reinforce the challenges inherent in decentralized domains. In particular, we show that there are limits to the conclusions we can draw from the best-known theorem concerning the simplification of such problems. It has been established that the worst-case complexity of finite-horizon Dec-POMDPs goes down—from NEXP to NP—when agents interact only via a joint reward structure, and are otherwise independent of one another. Unfortunately, further reductions, based on other combinations of fully or partially independent system dynamics are unlikely, if not impossible.

Indeed, we show that if the situation were reversed, and rewards were wholly independent, whereas other dynamics were not, the problem would remain NEXP-complete. Further, we consider two important Dec-POMDP sub-classes from the literature, each of which tries to inject more independence into the transitions and observations: (a) those domains in which the local agent sub-problems are independent but for a (relatively small) number of *event-based interactions*, and (b) those in

which agents only interact by way of influence on the set of *currently available actions*. As it turns out, both of these types of problem are NEXP-complete as well—facts previously unknown. (In the latter case, this is in fact a substantial increase in the upper bound on worst-case complexity.)

As we will argue, these largely negative results—at least in terms of ease of expected solution—provide further impetus to examine new tools for the analysis and classification of problem difficulty in decentralized problem solving. While the introduction of increasing degrees of independence is an intuitive and elegant way to simplify the apparent structure of a Dec-POMDP, it turns out that this does not always correspond with real reduced complexity. Among other things, this motivates our work in later chapters, where we attempt to solve this classification problem in a novel way, by isolating new general properties of Dec-POMDPs that correlate more directly with empirical problem difficulty.

3.1 Independence Relations in Dec-POMDPs

In general form, the functions governing state transitions, observations, and rewards in a Dec-POMDP can involve any number of probabilistic dependencies. An obvious subcase of such a model is thus one in which these factors are subject to various independence relations. Becker et al. [7, 11, 12] have thus studied problems in which the global state-space consists of the product of local states, so that each agent has its own individual state-space. A Dec-POMDP can then be *transition independent*, *observation independent*, or *reward independent*, as each the local effects given by each corresponding function are independent of one another. Thus, for instance, the local state transitions for any single agent in a transition-independent problem do not depend upon those of any other agent, and similarly for the other models. If a Dec-POMDP is in fact characterized by all three types of independence, then it is trivially factorable into separate individual POMDPs, one for each agent. Thus,

where we have a fully factorable problem that is also originally a Dec-MDP, the result is a set of separate single-agent MDPs, each of which can be solved using any of the usual tractable methods.

A more interesting result comes when the transition and observation dynamics of a Dec-MDP are independent, but the overall reward structure remains global, depending upon interactions of multiple agent actions. Such problems are restricted Dec-MDP instances, but still arise often enough in practice to be of some real interest (see discussion in [113, §2.5.1], for examples). As first shown by Goldman and Zilberstein [48], such problems are in fact NP-complete. Thus, while they remain generally intractable, complexity is significantly reduced by the independence restrictions on agent interactions, and it is suggested that such problems may be often more feasibly solved. Indeed, Becker et al. [12] present an optimal method for such problems—the *Coverage Set Algorithm*—that is often tractable in practice.

3.1.1 Factored Dec-MDPs and Independence

We begin with the basic formal definitions of this special sub-class of Dec-MDP, adapted from existing literature, and review the NP-completeness claim.

Definition 3.1 (Factored Dec-POMDP)**.** A factored, n-agent Dec-POMDP is a Dec-POMDP such that the system state can be factored into $n+1$ distinct components, so that $S = S_0 \times S_1 \times \cdots \times S_n$, and no state-variable appears in any S_i, S_j, $i \neq j$.

As with the *local* (agent-specific) actions, a_i, and observations, o_i, in the general Dec-POMDP definition, we now refer to the *local state*, $\hat{s} \in S_i \times S_0$, namely that portion of the overall state-space that is either specific to agent α_i ($s_i \in S_i$), or shared among all agents ($s_o \in S_0$). Note that we insist that the agent-specific portions of local states be non-overlapping and distinct portions of the remaining state-space; however, this restriction is not very limiting, as we can easily show.

Proposition 3.1. Any n-agent Dec-POMDP has an equivalent factored version.

Proof. Consider an arbitrary n-agent Dec-POMDP, in accordance with Definition 2.1:

$$\mathcal{D} = \langle \{\alpha_i\},\, S,\, \{A_i\},\, P,\, \{\Omega_i\},\, O,\, R,\, T \rangle.$$

We define a factored version of the problem

$$\mathcal{D}^f = \langle \{\alpha_i\},\, S^f,\, \{A_i\},\, P^f,\, \{\Omega_i\},\, O^f,\, R^f,\, T \rangle,$$

with components:

- $\{\alpha_i\}$, $\{A_i\}$, $\{\Omega_i\}$, T are identical between $\mathcal{D}$ and $\mathcal{D}^f$.
- We add n state variables, $v_1, \ldots, v_n$, each of which has a single constant value, $v_i = 1, \forall i$. For each agent α_i, the purely local portion of the state space is $S_i = \{v_i\}$, and we let $S_0 = S \in \mathcal{D}$, the original state-space. Every state $s^f \in (S^f = S_0 \times S_1 \times \cdots \times S_n)$ is then simply some state $s \in S \in \mathcal{D}$, with n 1's appended; for any state s in the original state-space, we thus write $s1^n$ for the corresponding state in the new state-space. Note that $|S| = \left|S^f\right|$.
- $\forall s1^n$, $s'1n$, $\forall a_1, \ldots, a_n$, $\forall o_1, \ldots, o_n$,
  1. $P^f(s1^n, a_1, \ldots, a_n, s'1^n) = P(s, a_1, \ldots, a_n, s')$.
  2. $O^f(\forall a_1, \ldots, a_n, s'1^n, o_1, \ldots, o_n) = O(\forall a_1, \ldots, a_n, s', o_1, \ldots, o_n)$.
  3. $R^f(s1^n, a_1, \ldots, a_n) = R(s, a_1, \ldots, a_n)$.

It is easy to see that the new Dec-POMDP, $\mathcal{D}^f$ is equivalent to the original, since all state-transitions, observations, and rewards are based on precisely the same state-variables in both problems. It is also worth noting that the proof establishes the same result for Dec-MDPs. If the original problem $\mathcal{D}$ is a Dec-MDP, then any observation sequence determines the global state s; thus, in the derived version $\mathcal{D}^f$, the observation sequence for $s1^n$ will also determine s, and the rest of the state is fixed. □

It is more interesting, however, to consider problems with system dynamics that actually break down separately and meaningfully over the various local states. In doing so, for convenience of presentation, we will use the notation $\overline{s}_{-i}$ for the sequence of all state-components *except that* for agent α_i:

$$\overline{s}_{-i} = (s_0, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$$

(and similarly for action- or observation-sequences, $\overline{a}_{-i}$ and $\overline{o}_{-i}$).

Definition 3.2 (Transition Independence)**.** A factored, n-agent DEC-POMDP is *transition independent* if and only if the state-transition function can be separated into $n+1$ distinct transition functions $P_0, \ldots, P_n$, where, for any next state $s'_i \in S_i$,

$$P(s'_i \,|\, (s_0, \ldots, s_n), (a_1, \ldots, a_n), \overline{s}_{-i}) = \begin{cases} P_0(s'_0 \,|\, s_0) & \text{if } i = 0; \\ P_i(s'_i \,|\, \hat{s}_i, a_i, s'_0) & \text{else.} \end{cases}$$

In other words, the next local state of each agent is independent of the local states of all others, given its previous local state and local action, and the external system features (S_0). Further, the external features evolve independently of all local agent actions or states. Given these independence relations, the joint probability distribution over global states is a product of the relevant individual distributions.

Definition 3.3 (Observation Independence)**.** A factored, n-agent Dec-POMDP is *observation independent* if and only if the joint observation function can be separated into n separate probability functions $O_1, \ldots, O_n$, where, for any local observation $o_i \in \Omega_i$,

$$O(o_i \,|\, (a_1, \ldots, a_n), (s'_0, \ldots, s'_n), \overline{o}_{-i}) = O_i(o_i \,|\, a_i, \hat{s}'_i)$$

In such cases, the probability of each agent's individual observations is a function of their own local states and actions alone, independent of the states of other

agents, and of what those others do or observe. As with transition probabilities, the conditional independence given by this definition implies that the joint observation function O can be represented as a product of the individual-agent observation functions. Finally, we can extend the notion of independence to the Dec-POMDP reward function, separating it into distinct local functions.

Definition 3.4 (Reward Independence)**.** A factored, n-agent Dec-POMDP is *reward independent* if and only if the joint reward function can be represented using a set of local reward functions $R_1, \ldots, R_n$, such that:

$$R((s_0, \ldots s_n), (a_0, \ldots, a_n)) = f(R_1(\hat{s}_1, a_1), \ldots, R_n(\hat{s}_n, a_n))$$

and

$$R_i(\hat{s}_i, a_i) \geq R_i(\hat{s}_i,\, a_i') \;\Leftrightarrow\; f(R_1, \ldots, R_i(\hat{s}_i, a_i), \ldots, R_n) \geq f(R_1, \ldots, R_i(\hat{s}_i, a_i'), \ldots, R_n)$$

That is, the joint reward function can be represented as a function over local reward functions alone, under the constraint that we maximize the global reward if and only if we maximize each of the local rewards. A typical example of such an independent factorization is the additive sum of local rewards,

$$R((s_0, \ldots s_n), (a_0, \ldots, a_n)) = R_1(\hat{s}_1, a_1) + \cdots + R_n(\hat{s}_n, a_n),$$

where the system simply receives the total reward gathered by individual agents.

It is important to note that each of the three definitions of independence applies equally to Dec-MDPs, just as well as Dec-POMDPs. (The reader will recall that a Dec-MDP is a Dec-POMDP that is *jointly fully observable* [Definition 2.2], so that the global state is determined by the current tuple of agent-observations.) When we are

dealing with the latter type of problem, we are often interested in instances for which the joint full observability of the overall state is accompanied by full observability at the local level.

Definition 3.5 (Local Full Observability)**.** A factored, n-agent Dec-MDP is *locally fully observable* if and only if any agent's local observation uniquely determines its local state: $\forall o_i \in \Omega_i, \exists \hat{s}_i : P(\hat{s}_i \,|\, o_i) = 1$.

Local full observability is not simply equivalent to independence of observations. In particular, a problem may be locally fully observable without being observation independent (since agents may simply observe the results of non-independent joint actions). On the other hand, it is easy to show that an observation-independent Dec-MDP must also be locally fully observable.[1]

Proposition 3.2. If any factored, n-agent Dec-MDP is observation independent, then it is locally fully observable as well.

Proof. Suppose an arbitrary factored, n-agent Dec-MDP $\mathcal{D}$ is observation independent. Assume, for contradiction, that it is not locally fully observable. By Definition 3.5, this means that there exists some agent α_i, and some observation $o_i \in \Omega_i$, for which there is no local state $\hat{s}_i$ such that $P(\hat{s}_i \,|\, o_i) = 1$. Let $o_i^\star$ be that observation. Now, since the observation function for agent α_i, O_i, must define a proper probability distribution (and so it cannot be that all states $\hat{s}_i$ have 0 probability given o_i), there must exist at least two local states with positive probability given the observation in question; that is, there exist $\hat{\mathbf{s}}_i^1$, $\hat{\mathbf{s}}_i^2$ such that:

1. $\hat{\mathbf{s}}_i^1 \neq \hat{\mathbf{s}}_i^2$;

2. $P(\hat{\mathbf{s}}_i^1 \,|\, o_i^\star) = k$ and $P(\hat{\mathbf{s}}_i^2 \,|\, o_i^\star) = m$, with $0 \lneq k, m \lneq 1$.

[1] A version of this result was originally proven by Goldman and Zilberstein [48], for a somewhat more complicated case, and required the additional assumption that the Dec-MDP was transition-independent as well. Our result is new, and the proof distinct.

By Bayes' Rule, and conditionalization:

$$P(\hat{\mathbf{s}}_i^1 \mid o_i^\star) = \frac{P(o_i^\star \mid \hat{\mathbf{s}}_i^1)\, P(\hat{\mathbf{s}}_i^1)}{P(o_i^\star)}$$
$$= \frac{\sum_{a_i} P(o_i^\star \mid a_i, \hat{\mathbf{s}}_i^1)\, P(a_i \mid \hat{\mathbf{s}}_i^1)\, P(\hat{\mathbf{s}}_i^1)}{P(o_i^\star)}$$

and therefore, since $P(\hat{\mathbf{s}}_i^1 \mid o_i^\star) \neq 0$, we know that there exists some action, call it $\mathbf{a}_i^1$, such that $P(o_i^\star \mid \mathbf{a}_i^1, \hat{\mathbf{s}}_i^1) \neq 0$. By similar reasoning, there exists some action, call it $\mathbf{a}_i^2$, such that $P(o_i^\star \mid \mathbf{a}_i^2, \hat{\mathbf{s}}_i^2) \neq 0$. Now, by Definition 3.3, since $\mathcal{D}$ is observation independent, we can factor the observation function:

$$O((a_1, \ldots, a_n), (s_0, \ldots, s_n), (o_1, \ldots, o_n)) = \prod_{1 \leq i \leq n} O_i(a_i, \hat{s}_i, o_i)$$

Now consider any combination of observation-function values with non-zero value, featuring designated components for agent α_i, and otherwise identical:

$$O_1(a_1, \hat{s}_1, o_1) \times \cdots \times O_i(\mathbf{a}_i^1, \hat{\mathbf{s}}_i^1, o_i^\star) \times \cdots \times O_n(a_n, \hat{s}_n, o_n) \neq 0 \tag{3.1}$$

$$O_1(a_1, \hat{s}_1, o_1) \times \cdots \times O_i(\mathbf{a}_i^2, \hat{\mathbf{s}}_i^2, o_i^\star) \times \cdots \times O_n(a_n, \hat{s}_n, o_n) \neq 0 \tag{3.2}$$

(Such a combination must exist, trivially.) Since $\mathcal{D}$ is a Dec-MDP, by Definition 2.2, then by Equation (3.1) there exists a function J from observation-tuples to states such that: $J(o_1, \ldots, o_i^\star, \ldots, o_n) = (\hat{s}_1, \ldots, \hat{\mathbf{s}}_i^1, \ldots, \hat{s}_n)$. Similarly, by Equation (3.2), $J(o_1, \ldots, o_i^\star, \ldots, o_n) = (\hat{s}_1, \ldots, \hat{\mathbf{s}}_i^2, \ldots, \hat{s}_n)$. However, since $\hat{\mathbf{s}}_i^1 \neq \hat{\mathbf{s}}_i^2$, and J is a function, this is impossible. Therefore, by *reductio*, $\mathcal{D}$ must be locally fully observable. □

3.1.2 Examples of Independent Problem Domains

While the various independent sub-classes of the general Dec-POMDP or Dec-MDP framework are restricted, they remain useful, as they represent many problems encountered in practice. Transition-Independent problems cover any domains in

which individual agents can operate without interference between them; when we do not add additional presumptions of independence, these can still be quite rich. Decentralized computation, where multiple machines work on sub-parts of a problem that has been divided up ahead of time, can often be regarded as a transition-independent domain, for instance. Once each processor has received its own batch of jobs or instructions, these can be run entirely separately, before being returned to a central location. Such problems remain highly complex, since the final outcome—the reward accumulated once the diverse results are composed into a single solution—can vary widely, and there may be many interdependencies between the rewards and results gained in the end. Becker et al. [12] consider a Mars rover domain with similar features, in that rovers are able to move about their environment and gather scientific samples separately, without any dependent interactions; however, the final value of the samples gathered does involve interdependencies, since for example multiple samples of the same material is less valuable than a range of distinct samples. The same issues arise in the highly challenging area of vehicle routing, where individual elements of a fleet of delivery trucks, for example, navigate their environment from station to station absolutely independently, but the coordination of routing decisions for optimal pay-off in terms of time and quality of service remains difficult.

Observation independence is also common in these sorts of domains. When agents work separately, it is often the case that they observe nothing of the local variables and actions of others. In the rover example, it is generally assumed that the rovers have no access to one another's activity. Similarly, in distributed processing, one node will generally have no access to conditions and outcomes at other nodes. It is worth noting that such forms of independence contribute to the challenge of the problems. While dependent observations do add to the complexity of a problem—in that they provide additional information that must be accounted for in planning—the lack of

such information can also make optimal solution more difficult, since less may be known about how one's local action outcomes will affect final reward gained.

Reward independence is also common, since it often makes most sense to regard the global reward function for a problem as an additive function of individual agent rewards, ensuring that maximizing each agent's individual outcomes will provide globally maximal reward as well. When such problems do not include the other forms of independence, they can remain quite challenging. For instance, in modeling urban vehicle traffic, it is possible to treat reward as a function of individual rewards, such as the sum of inverse travel times during a commute. Because vehicle travel times and conditions do not evolve independently, such problems are generally not transition or observation independent, and so do not simply decompose into separate MDPs or POMDPs, as they would if all three forms of independent were present.

3.2 Shared Reward Structures

As just mentioned, if a Dec-MDP (or Dec-POMDP) has all three forms of independence given by Definitions 3.2–3.4, it can be decomposed straightforwardly into n separate problems, where each agent α_i works solely within the sub-environment given by local states in $S_i \times S_0$. Such single-agent problems are known to be P-complete, and can be generally solved to high degrees of optimality in efficient fashion.[2] to calling the problems polynomially solvable; indeed, when However, it is possible to create much more complex problem instances when nearly all of the independence relationships hold. In particular, it has been shown that Dec-MDPs that are both

[2]See Papadimitriou and Tsitsiklis [93]. Strictly speaking, MDPs are only solvable in *pseudopolynomial* time by most common dynamic programming methods, since the iterations of the algorithm depend directly upon the time horizon, which is typically given in concise (binary) form; thus T iterations of an algorithm for a problem whose size is $(O)(\log T)$ is actually exponential in the problem size. If T is given in prolix (unary) fashion, then strictly polynomial solving time results. (These points are discussed in detail by Littman, Dean, and Kaelbling [77].) This is not typically considered a serious objection to treating MDPs as feasible, polytime problems; often the stipulation is made that $T \ll |S|$, for convenience.

transition- and observation-independent, but do not separate their reward-structure, are NP-complete.

We review this result, found in Goldman and Zilberstein [48] and Becker et al. [12]. To do so, we first describe the *event-based* reward structure used in that work. While somewhat complicated, these ideas are of broader use to us. Later sections and chapters will also make use of these definitions, or versions of them, to show how other combinations of independent dynamics relate to complexity, and how the nature of the reward structure relates precisely to the essential dimensionality of problem instances. Again, we give all definitions in terms of Dec-POMDPs; they apply immediately to Dec-MDPs in particular.

Definition 3.6 (History)**.** A *history* for an agent α_i in a factored, n-agent Dec-POMDP $\mathcal{D}$ is a sequence of possible local states and actions, beginning in the agent's initial state: $\Phi_i = [\hat{s}_i^0, a_i^0, \hat{s}_i^1, a_i^1, \ldots]$. When a problem has a finite time-horizon T, all possible complete histories will be of the form $\Phi_i^T = [\hat{s}_i^0, a_i^0, \hat{s}_i^1, a_i^1, \ldots, \hat{s}_i^{T-1}, a_i^T, \hat{s}_i^T]$.

Definition 3.7 (Events in a History)**.** A *primitive event* $e = (\hat{s}_i, a_i, \hat{s}_i')$ for an agent α_i is a triple representing a transition between two local states, given some action $a_i \in A_i$. An *event* $E = \{e_1, e_2, \ldots, e_h\}$ is a set of primitive events. A primitive event e *occurs in the history* Φ_i, written $\Phi_i \vDash e$, if and only if the triple e is a sub-sequence of the sequence Φ_i. An event E *occurs in the history* Φ_i, written $\Phi_i \vDash E$, if and only if some component occurs in that history: $\exists e \in E : \Phi_i \vDash e$.

Events can therefore be thought of disjunctively. That is, they specify a set of possible state-action transitions from a Dec-POMDP, local to one of its agents. If the historical sequence of state-action transitions that the agent encounters contains any one of those particular transitions, then the history satisfies the overall event. Events can thus be used, for example, to represent such things as taking a particular action in any one of a number of states over time, or taking one of several actions at

some particular state. For technical reasons, namely the use of a specialized solution algorithm, these events are usually restricted in structure, as follows.[3]

Definition 3.8 (Proper Events)**.** A primitive event e is *proper* if it occurs at most once in any given history. That is, for any history Φ_i if $\Phi_i = \Phi_i^1\, e\, \Phi_i^2$ then neither sub-history contains e: $\neg(\Phi_i^1 \vDash e) \wedge \neg(\Phi_i^2 \vDash e)$. An event E is *proper* if it consists of proper primitive events that are mutually exclusive, in that no two of them both occur in any history:

$$\forall \Phi_i \neg\exists x,\, y :\, (x \neq y) \wedge (e_x \in E) \wedge (e_y \in E) \wedge (\Phi_i \vDash e_x) \wedge (\Phi_i \vDash e_y).$$

Proper primitive events can be used, for instance, to represent actions that take place at particular times (building the time into the local state $\hat{s}_i \in e$). Since any given point in time can only occur once in any history, the events involving such time-steps will be proper by default. A proper event E can then be formed by collecting all the primitive events involving some single time-step, or by taking all possible primitive events involving an unrepeatable action. These ideas can then be used to define a non-independent reward structure for a Dec-POMDP $\mathcal{D}$, in terms of combinations of events for various subsets of the agents involved. Intuitively, the overall reward will be given in terms of what is to be gained or lost when every event in one of these combinations actually occurs during some joint history of the process.

Definition 3.9 (Joint Reward Structure)**.** A *joint reward structure* maps various sets of events for distinct agents to reward values:

$$\rho = \{\langle E_i^k, \ldots, E_i^m,\, c_i\rangle \,|\, 0 \leq i \leq j,\, j \in \mathbb{N},\, c_i \in \Re\}$$

[3]Becker et al. [12] describe how these limitations on the structure or events can be overcome, allowing a wider class of possible event-types.

where each event E_i^k $(1 \leq k \leq n)$ is an event for agent α_k, and no agent appears twice in the sequence, so that there is no E_i^k and E_i^m such that $k = m$. We write ρ_i for the ith sequence in ρ, featuring events E_i^k and value c_i. We refer to $|\rho| = j$, the number of such tuples, as the size of the reward structure.

For any tuple $\langle E_i^k, \ldots, E_i^m, c_i \rangle \in \rho$, the system receives the reward c_i if each event in the tuple occurs; that is, reward is received if and only if, for each event E_i^k in the tuple, $\Phi_k \vDash E_i^k$. That is, the reward is received if every event it contains occurs somewhere in the joint history of the process. Note that, as defined, the sequence of events may involve a proper subset of agents, so that a given reward value c_i can depend only upon events featuring some, but not all, of the agents.

In principle, any reward-function R in a Dec-POMDP $\mathcal{D}$ can be defined in terms of such an event-based reward structure, and so this does not in any way specialize the general definition. However, such structures allow us to isolate some interesting properties of otherwise-independent problems. As we discuss in the next chapter, there is a direct connection between the size of the event-based reward structure and the essential dimensionality of a given problem instance. This is particularly interesting, given results in the literature that suggest that the operation of solution algorithms is directly affected by this size. Furthermore, restricting dependence to the reward function alone reduces the overall complexity of finding optimal solution.

Any agent α_i in a factored Dec-MDP that is both transition- and observation-independent can act optimally based upon a *local policy*, $\pi_i : (S_i \times S_0) \rightarrow A_i$, mapping local states to local actions. The probability that some primitive event occurs, given that the agent follows some local policy is then straightforward. Letting $e = (\hat{s}_i, a_i, \hat{s}_i')$, then $P(e \,|\, \pi_i) = 0$ if $\pi_i(\hat{s}_i) \neq a_i$, since the policy will guarantee that the agent never takes that particular action in that particular state; otherwise,

$$P(e \,|\, \pi_i) = \sum_{t=0}^{T} P_t(\hat{s}_i \,|\, \pi_i)\, P_i(\hat{s}_i' \,|\, \hat{s}_i, a_i), \tag{3.3}$$

where $P_t(\hat{s}_i \mid \pi_i)$ gives the probability of being in state $\hat{s}_i$ at time t, given policy π_i. This can be calculated straightforwardly from the Dec-MDP specification of state-transitions (and the other probability, $P_i(\hat{s}'_i \mid \hat{s}_i, a_i)$, is simply α_i's transition function). The expected value of a joint policy for all n agents follows immediately.

Definition 3.10 (Policy Value with Shared Rewards)**.** For a factored, n-agent Dec-MDP $\mathcal{D}$ with joint reward structure ρ, the *value of a joint policy* $(\pi_1, \ldots, \pi_n)$ is:

$$V(\pi_1, \ldots, \pi_n) = \sum_{i=1}^{|\rho|} c_i \prod_{E_i^k \in \rho_i} P(E_i^k \mid \pi_k).$$

The *optimal joint policy*, $(\pi_1, \ldots, \pi_n)^\star$, is one that maximizes the above equation.

3.2.1 Complexity of the Joint-Reward Case

As it turns out, finding a policy that maximizes value in such a problem is provably easier than the worst-case NEXP-completeness of general Dec-POMDPs. While we do not go through the proof in detail, we state this important result here.

Theorem 3.1. Solving Dec-MDPs with independent transitions and observations, and a joint reward structure is NP-complete.

Proof Sketch (after Lemma 4, Goldman and Zilberstein [48]). For the upper bound, we show that the problem of deciding whether a given policy has expected value greater than some constant r is in NP. This follows straightforwardly from the policy value as given in Definition 3.10. Given any joint policy $(\pi_1, \ldots, \pi_n)$ for consideration, each $\rho_i \in \rho$ can be evaluated in time that is polynomial in the state space and number of agents, since each proper event involves a single distinct agent. Furthermore, each of these events can be evaluated in time polynomial in the problem size, simply summing up the probability of constituent primitive events, as determined in Equation (3.3). Since the reward-structure, ρ, is part of the problem description, the

number of reward-components in the main sum, $|\rho|$, results in overall effort polynomial in the problem size.[4]

For the lower bound, we can reduce the NP-complete problem DTEAM (Papadimitriou & Tsitsiklis [91, 92]) to an instance of the given type of Dec-MDP. We omit this reduction, which is quite elementary; the interested reader is directed to the full version of the proof of Theorem 1, Becker et al. [12]. □

3.2.2 Conclusions and Discussion

Unlike the distinction between the complexity classes P and NP, which is generally considered to be absolute, but has never been proved so, the classes NEXP and NP are known to be distinct, with NEXP demonstrably harder than NP. Thus, the NP-completeness proof shows a significant reduction in complexity for the special class of Dec-MDPs under consideration. In practice, furthermore, NP-hard problems are those for which many instances exhibit exponential time-costs using typical solution techniques. NEXP-hard problems, on the other hand, exhibit *doubly exponential* time-costs, leading to a significant leap in observed run-time, so that, as we have repeatedly stressed, only very small instances can actually be solved to optimality. Therefore, even if the reduction to NP leaves these Dec-MDPs still generally infeasible, they often reduce the solution times seen in practice to the point where real problems of interest can be solved.

In this connection, Becker's *Coverage Set Algorithm* (CSA) has been demonstrated to solve shared-reward problems of reasonable size (for details, see Becker et al. [12], and his thesis [8]). While the operation of the algorithm exhibits exponential time-cost in the size of the joint reward structure, this is often quite manageable, given a sparsely-connected series of events. This relationship is expanded upon in the

[4]This argument, while distinct from the one given by Goldman and Zilberstein, establishes the equivalent point. We include this version for variety's sake.

next chapter, when we consider an approach devised by Petrik and Zilberstein [99] that extends the CSA method by employing powerful mathematical programming techniques. As we show there, the reward structure is importantly related to the ability to reduce the dimensionality of such problems, and optimize the solution time.

Unfortunately, there is a limit to the power of the NP-completeness result. As we shall show, other relatively obvious combinations of independence relationships in Dec-MDPs do not yield the same general complexity reduction. Before we prove these new results, we consider the original proof of NEXP-completeness in Dec-POMDPs, which we draw on later.

3.3 Bernstein's Proof of NEXP-Completeness

Before establishing our new claims, we review the proof of NEXP-completeness for finite-horizon Dec-MDPs, as given by Bernstein et al. [18, 16] (we draw here on perhaps the most complete version of the proof, from Bernstein's thesis [15]). While this is somewhat complicated, it is the basis for our next three main results, and is worth explicating in some detail.

First, we note that the upper bound, namely that finite-horizon Dec-POMDPs are in NEXP, meaning that they can actually be solved in nondeterministic exponential time, will immediately and without modification establish the same upper bound for all the problems that we will consider, since each is but a special sub-case of the general finite-horizon framework. As we showed in Proposition 3.1, any Dec-POMDP can be written in an equivalent factored form. Thus, Bernstein's upper bound immediately establishes the identical bound on the complexity of any factored finite-horizon Dec-POMDP; since all the problems we will look at are but special restricted cases (like

the transition- and observation-independent Dec-MDPs with shared rewards), the upper bound applies to all of them as well.[5]

Theorem 3.2 (Upper Bound)**.** The decision version of the finite-horizon, n-agent problem Dec-POMDP $\in$ NEXP.

Proof (after Theorem 4, Bernstein [15]). Consider arbitrary finite-horizon, n-agent Dec-POMDP $\mathcal{D}$, constant k, and relevant joint policy π. We show that we can evaluate whether the value of π is at least k in exponential time. Policy π will, by definition, consist of n mappings from agents' observation-histories to actions. Any Dec-POMDP, taken together with such a joint policy, can be treated as a single-agent POMDP with individual policy, by treating each n-tuple of observations as a single, complex observation, and likewise for joint action-tuples. In exponential time, each of the exponentially-many possible sequences of such complex observations can be converted into a belief-state over possible states of the underlying system. Similarly, state-transitions and expected rewards over belief-states can be calculated in exponential time, turning the problem into a belief-state MDP, with state-space exponential in the size of the original Dec-POMDP. Finally, dynamic programming can be used to evaluate policy π over the MDP, in time that is polynomial in the size of the new state-space (or exponential overall), to determine whether its value is at least k. □

Since this result gives us an upper bound on the complexity of each problem-type we examine, we will simply refer back to it as needed. More challenging (and more interesting) is the process of establishing lower bounds on these problems. That is, in each case, we shall want to show that the problems are in fact *NEXP-hard*, and

[5]Note that, as is standard in such complexity proofs, results are stated in terms of the "*decision version*" of the problem. For the class of Dec-POMDPs, for instance, the decision version is the problem of determining, for a given Dec-POMDP $\mathcal{D}$, and constant value k, whether there exists any policy for $\mathcal{D}$ with value at least k. This suffices to establish the complexity of the optimal-policy problem, since any method that could produce an optimal policy could answer the decision question in the affirmative for suitably chosen values of k.

actually require nondeterministic exponential time to solve. The theoretical tool that allows us to establish this result is the *reduction*, whereby any instance of a problem already known to have that complexity is transformed (without expanding the size of the problem representation beyond a polynomial factor) into an instance of the new problem of interest. If it can then be shown that a solution to the latter form of the problem is in fact a solution to the former, we have established that the latter form is no easier than the original. Furthermore, if the problem that is reduced is known to be *complete* for its complexity class—meaning that any problem in that class can be reduced to it—then the new problem class is also complete for the class, since reduction is a transitive operation.

3.3.1 The TILING Problem

The problem used in our reductions is the NEXP-complete TILING problem (see Lewis [75], Papadimitriou [90]). A TILING problem instance consists of a board size n, given concisely in $\log n$ binary bits, a set of tile-types $L = \{t_0, \ldots, t_k\}$, and a collection of binary and vertical compatibility relations between tiles H, $V \subseteq L \times L$. A *tiling* is a mapping of board locations to tile-types, $t : \{0, \ldots, n-1\} \times \{0, \ldots, n-1\} \rightarrow L$; such a tiling is *consistent* just in case (i) the origin location of the board receives tile-type 0 ($t(0,0) = tile_0$); and (ii) all adjoint tile assignments are compatible:

$$(\forall x, y)\ \langle t(x,y), t(x+1,y)\rangle \in H \quad \& \quad \langle t(x,y), t(x,y+1)\rangle \in V.$$

The TILING problem is thus to decide, for a given instance, whether such a consistent tiling exists. Figure 3.1 shows an example instance and consistent solution.

In what follows, we will sketch the main details of the reduction of TILING to a 2-agent Dec-MDP (this will establish the hardness result for the more general Dec-POMDP case). A key detail to keep in mind going forward is that such a reduction must not increase the size of the original TILING problem beyond a polynomial factor.

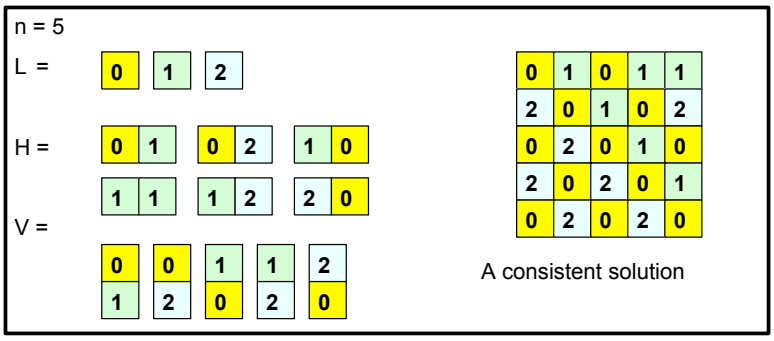


Figure 3.1: An example of the TILING problem, and a consistent solution.

Since the TILING instance is given with the board size n represented logarithmically, it will be necessary that any reductions likewise produce problems with size that is $\mathcal{O}(\log n)$. (For the purposes of proof, we can presume that n is suitably large, so that the description of the set of tile-types and compatibility relations is also possible in size logarithmic in n.)

3.3.2 The Basic Reduction

The essential idea behind the reduction is to begin with some arbitrary instance of TILING, and create a finite-horizon, 2-agent Dec-MDP such that the latter has a policy with non-negative value if and only if the former has a consistent tiling. Thus, we will have shown that the problem of determining whether a Dec-MDP has a policy with value at least $k = 0$ is as hard as TILING, which, together with the upper bound result, will establish NEXP-completeness.

To do this, the Dec-MDP is designed to choose two random tile positions from the board, which are then revealed separately to each agent, bit-by-bit. (The state-set of the Dec-MDP can not directly include the chosen positions, else it would have a state-set of impermissible size $\mathcal{O}(n)$, i.e., exponential in $\log n$.) Following the revelation of the tile locations, each agent chooses a tile to place there. The agents are then required to repeat the bits they have seen back to the system, which calculates the

relative locations of the tiles, and checks them for compatibility as necessary. In order to ensure that the agents are honest in repeating the bits they have seen, the system takes certain precautions to remember one such bit for each agent, without letting the agent know the precise one that is being remembered. Deviations from the remembered bit, or tile-types that are incompatible, lead to negative reward; therefore, an optimal (0-value) policy is only possible if the agents already possess a consistent solution for the original TILING problem, i.e., such a tiling must exist.

To accomplish this, the reduction thus creates a Dec-MDP with dynamics that work in the following 5 phases.

Stage 1: Select. The system randomly chooses two possible indices, and one bit-value at each of those indices, to encode one bit of each of the grid-locations given to the two agents. Each of these is remembered for later.

Stage 2: Generate. Under the constraints given by the bits chosen in the prior phase, two otherwise-random locations in the TILING grid, revealing one to each agent, bit-by-bit.

Stage 3: Query. Each agent supplies a tile-type, which is remembered for later.

Stage 4: Echo. The agents repeat back the bits they have seen, allowing the system to calculate the relative positions of the two tiles. The location-bits remembered from the **Select** phase are used to keep the agents honest.

Stage 5: Test Having calculated the location of the two tiles, they are evaluated to see whether they are consistent, according to the given TILING problem.

We give more detail of this reduction below.

3.3.3 The Proof of Correctness

In later sections, we present modifications of Bernstein's reduction, extending the proof technique to new sub-classes of the Dec-MDP framework. Thus, in the next

section, we go over the nature of his reduction in detail, in order to make the extent of our own alterations clear. Before doing so, however, we outline Bernstein's *correctness results*, which demonstrate that his reduced problem has a policy with non-negative value if and only if the original TILING instance has a consistent solution. Like his upper-bound result (Theorem 3.2), we will rely upon these claims in our own proofs; we therefore describe their salient features here, so that we may refer back to them.

Proposition 3.3. If a consistent solution to the arbitrary TILING problem given exists, then there exists a policy for the two agents with non-negative expected value.

Sketch of proof (after Lemma 1, Bernstein [15]). In the reduced problem, it is easy to show that the sole policy with non-negative expected value involves two main elements: (1) during the **Query** phase, providing a tile-type for the location provided that is identical with that in an agreed-upon consistent tiling; and (2) during the **Echo** phase, faithfully repeating the observed location-bits, so that the system can calculate the compatibility relationships accurately. □

In our own reductions, we will rely upon this result, arguing that the sole policy with non-negative expected value must have the same features. Thus, we can use Bernstein's proof directly.

Proposition 3.4. The existence of a policy with non-negative value for the reduced problem implies the existence of a consistent solution to the TILING instance given.

Sketch of proof (after Claims 1–3, Bernstein [15]). In the reduced problem, if either agent deviates from the policy that faithfully echoes the location-bits seen during the earlier **Generate** phase of the problem, then there will exist observation sequences for which they have some positive probability of providing an incompatible pair of tile-types. That is, since the agents do not know which bit of the location provided them is being memorized by the system, they have no way of "gaming the system" reliably. In particular, they have no way of knowing whether the system has revealed the same tile

location (or adjacent locations) to each agent, and cannot avoid the possibility that they are providing an incompatible pair of tile-types, leading to negative expected reward in the final **Test** phase. Only by following the strictly faithful policy can they avoid this possibility, and thus they must provide a consistent tiling. □

As for the previous proposition, we will rely upon Bernstein's proof directly. Our own reductions will also require that agents provide faithful echoes of the locations given, and choose compatible tile-types at any such location. The main differences will be in the specific structure of the reduced problem, not in the nature of the non-negative policies for same.

Together with Theorem 3.2, and the fact that the finite-horizon, 2-agent Dec-MDP is a special case of the general finite-horizon Dec-POMDP, these propositions establish Bernstein's main complexity result:

Theorem 3.3 (NEXP-Completeness)**.** The finite-horizon Dec-POMDP problem is NEXP-complete.

In this connection, we note that the infinite-horizon version of the problem is undecidable, a fact that follows directly from the undecidable nature of infinite-horizon, single-agent POMDPs (Madani, et al. [81]).

3.3.4 Details of the Reduction: State Space

We now outline the specifics of the Dec-MDP problem used in the reduction result. Given a TILING instance $\langle n, L, H, V \rangle$, we create a finite-horizon, 2-agent Dec-MDP as follows. A key element of the model derives from the fact that in a Dec-MDP, unlike a Dec-POMDP, we must be able to determine the global state, once we know the observations of each agent. This, along with the need to ensure that neither agent can observe the remembered bit that the system uses to keep them honest during the **Query** phase, leads us to divide the set of variables defining a given state into 3 parts, according to which of the agents observes them.

Both Agents. Four (4) variables are observed at each step. First, there is variable $phase \in \{sel, gen, query, echo, test\}$, indicating the current phase of the process. Second, there is variable $index \in \{0, \ldots, 2 \log n\}$, indicating the next (x, y)-location bit to be generated or echoed back. Third, there is $origin \in T, F$, indicating whether or not we are at the origin point $(0, 0)$ of the tiling grid. Lastly, there is $q \in Q$, tracking the state of a finite-state automaton that calculates the relative position of the locations echoed back by each agent (as will be described briefly below, we ensure that $|Q| = \mathcal{O}(\log n)$, as required).

Agent 1. This agent observes four (4) variables of its own, as well. First, it observes the index and value of the location bit that the system remembers for the *other agent*, $index_2 \in \{0, \ldots, 2 \log n - 1\}$ and $value_2 \in \{0, 1\}$. Second, it observes the current position-bit used in the **Generate** phase to provide a tiling-location, $pos_1 \in \{0, 1\}$, and its own choice of a tile-type $tile_1 \in L$.

Agent 2. This agent likewise observes (4) variables of its own, each of which is directly analogous to those observed by the other agent: $index_1$, $value_1$, pos_2, and $tile_2$.

The set of all states, S, is thus any combination of these variables, and thus $|S| = \mathcal{O}(\log n)$; we write any such state as a tuple of possible variable values, using a semicolon to separate each of the three classes of observed variables, such as:

$$s = \langle query, 5, F, q_0; 3, 1, 0, t_3; 4, 0, 1, t_5 \rangle$$

As a convention, we can abstract over a set of variables by only partially specifying their values, using "$*$" where we are indifferent. For instance, we can identify the set of all variables during the **Echo** phase, writing $s \in \langle echo, *, *, *; *, *, *, *; *, *, *, * \rangle$ to indicate such a state.

The Dec-MDP begins in the **Select** phase, with neither agent having chosen a tile-type, and the FSA at its initial state: $s_0 = \langle sel, 0, T, q_0; 0, 0, 0, t_0; 0, 0, 0, t_0 \rangle$

3.3.5 Details of the Reduction: Action Transitions

Each agent possesses two possible actions for repeating bits back to the system, and an action for selecting each possible tile-type; therefore, $A_1 = A_2 = \{0, 1\} \cup L$.[6] The state-action transitions are specified as follows, for each phase of the problem dynamics (any combinations not covered by the description are not reachable in the problem, and can be treated arbitrarily).

Phase 1: Select. The process selects one bit of the location to be revealed to each agent uniformly at random, and reveals it solely to the other agent.

$$\begin{aligned} P(s, a_1, a_2, s') = \frac{1}{(4 \log n)^2} \quad & \text{whenever:} \\ s = s_0 &= \langle sel, 0, T, q_0; 0, 0, 0, t_0; 0, 0, 0, t_0 \rangle, \\ s' &= \langle gen, 0, T, q_0; i_2, v_2, 0, t_0; i_1, v_1, 0, t_0 \rangle, \\ i_1,\, i_2 \in \{0, \ldots, (2 \log n) - 1\}, & \text{ and } v_1,\, v_2 \in \{0, 1\}. \end{aligned}$$

Phase 2: Generate. Two tile positions are chosen and revealed to each agent, bit-by-bit; this is done randomly, under the constraint that the bit chosen in the prior phase (**Select**) must be honored. After the entire bit-sequence has been revealed to each agent, the problem transitions deterministically to the next phase (**Query**).

---

[6] Here, we simplify Bernstein's proof somewhat. In the original, agents also possessed a "wait" action, allowing them to do nothing during some phases of the problem. However, as we will see, all state-transitions during such "idle" phases are independent of any actions taken, so no separate waiting action is required (the agents can choose any of their other possible actions whatsoever with no effect on possible outcomes).

$$P(s, a_1, a_2, s') = \frac{1}{h} \text{ whenever:}$$

$$s = \langle gen, k, T, q_0; i_2, v_2, *, t_0; i_1, v_1, *, t_0 \rangle \quad (0 \leq k < 2 \log n),$$

$$s' = \langle gen, k+1, T, q_0; i_2, v_2, b_1, t_0; i_1, v_1, b_2, t_0 \rangle, \text{ where}$$

$$b_1 = v_1 \text{ if } k = i_1, \text{ else } b_1 \in \{0, 1\},$$

$$b_2 = v_2 \text{ if } k = i_2, \text{ else } b_2 \in \{0, 1\},$$

$$h \text{ is the number of allowed combinations of } b_1 \text{ and } b_2.$$

$$P(s, a_1, a_2, s') = 1 \text{ whenever:}$$

$$s = \langle gen, 2 \log n, T, q_0; i_2, v_2, *, t_0; i_1, v_1, *, t_0 \rangle,$$

$$s' = \langle query, 0, T, q_0; i_2, v_2, 0, t_0; i_1, v_1, 0, t_0 \rangle.$$

Phase 3: Query. This is a one-step phase, in which each agent chooses a tile-type, and we transition to the **Echo** phase.

$$P(s, a_1, a_2, s') = 1 \text{ whenever:}$$

$$s = \langle query, 0, T, q_0; i_2, v_2, 0, t_0; i_1, v_1, 0, t_0 \rangle,$$

$$s' = \langle echo, 0, T, q_0; i_2, v_2, 0, \mathbf{t_1}; i_1, v_1, 0, \mathbf{t_2} \rangle, \text{ where}$$

$$\mathbf{t_i} = \begin{cases} a_i & \text{if } a_i \in L \\ t_0 & \text{else.} \end{cases}$$

Phase 4: Echo. The system now has the agents repeat back the location-bits that they previously observed. Using these bits, a finite-state automaton calculates the relative positions of the tile-locations. For any state $q \in Q$ of the FSA, and two echoed bits b_1, b_2, we write $FSA(q, b_1, b_2)$ for the output of the automaton at that stage. Full details of the FSA construction can be found in Bernstein [15, §3.4.4]; here, we simply note again the important fact that it can be constructed within the

proper space bounds, $|Q| = \mathcal{O}(\log n)$. Once all the bits have been echoed back, the system transitions to the final phase (**Test**).

$$P(s, a_1, a_2, s') = 1 \text{ whenever:}$$

$$s = \langle echo, k, o, q; i_2, v_2, 0, \mathbf{t_1}; i_1, v_1, 0, \mathbf{t_2} \rangle,$$

$$s' = \langle p, k', o', FSA(q, b_1, b_2); i_2, v_2, 0, \mathbf{t_1}; i_1, v_1, 0, \mathbf{t_2} \rangle, \text{ where}$$

$$b_i = \begin{cases} 1 & \text{if } a_i = 1 \\ 0 & \text{else} \end{cases}, \quad p, k' = \begin{cases} echo, k+1 & \text{if } 0 \leq k < (2 \log n) - 1 \\ test, 0 & \text{if } k = (2 \log n) - 1 \end{cases},$$

$$o' = T \Leftrightarrow (o = T \ \& \ a_1 = a_2 = 0).$$

Phase 5: Test. The final phase is a single-step process in which the system terminates.

$$P(s, a_1, a_2, s') = 1 \text{ whenever:}$$

$$s = \langle test, 0, *, *; *, *, 0, *; *, *, 0, * \rangle,$$

$$s' = \langle test, 0, T, q_0; 0, 0, 0, t_0; 0, 0, 0, t_0 \rangle.$$

3.3.6 Details of the Reduction: Rewards

The Reward function for the Dec-MDP is simple: agents receive reward of -1 for any state-action pair, except for a special designated set, for which they receive reward of 0. This set corresponds to the key features of the desired policy: faithfully repeating the bit that the system has remembered from their location during the **Echo** phase, and ending up with a compatible pair of tiles in the **Test** phase.[7] This

[7]Again, we simplify Bernstein somewhat: since we have no need of a "wait" action, and since the state-transitions have the effect of choosing a default tile-type (t_0) or bit (0) should the agents use an inappropriate action at the relevant phases, we can allow agents to act without penalty in any fashion they choose during the "idle" phases, where transitions do not depend upon those actions.

last reward depends upon the final state of the FSA that calculates relative tile-locations; it will be drawn from the set {**equal**, **hor**, **ver**, **apart**}, as the locations are respectively identical, horizontally adjacent, vertically adjacent, or otherwise.

$R(s, a_1, a_2) = 0$ if and only if one of the following is true:

Select: $s = \langle sel, *, *, *; *, *, *, *; *, *, *, * \rangle$.

Generate: $s = \langle gen, *, *, *; *, *, *, *; *, *, *, * \rangle$.

Query: $s = \langle query, *, *, *; *, *, *, *; *, *, *, * \rangle$.

Echo: $s = \langle echo, k, *, *; i_2, v_2, *, *; i_1, v_1, *, * \rangle$,

$a_1, a_2 \in \{0, 1\}$ and ($a_1 = v_1$ or $k \neq i_1$) and ($a_2 = v_2$ or $k \neq i_2$).

Test (i): $s = \langle test, *, o, \mathbf{equal}; *, *, *, \mathbf{t_1}; *, *, *, \mathbf{t_1} \rangle$,

$o = F$ or $\mathbf{t_1} = t_0$.

Test (ii): $s = \langle test, *, *, \mathbf{hor}; *, *, *, \mathbf{t_1}; *, *, *, \mathbf{t_2} \rangle$,

$\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in H$.

Test (iii): $s = \langle test, *, *, \mathbf{ver}; *, *, *, \mathbf{t_1}; *, *, *, \mathbf{t_2} \rangle$,

$\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in V$.

Test (iv): $s = \langle test, *, *, \mathbf{apart}; *, *, *, *; *, *, *, * \rangle$.

3.3.7 Details of the Reduction: Observations and Time Horizon

Finally, we simply set the observation function for each agent to be deterministic: given a state, agent α_i observes the four variables that make up its portion of the state space, along with the global variables that make up the first portion. The observation-set for α_1 is thus $\Omega_1 = \{phase, index, origin, Q, index_2, value_2, pos_1, tile_1\}$, and similarly for α_2. The time horizon is $T = (4 \log n) + 4$, consisting of one step each for the **Select**, **Query**, and **Test** phases, $(2 \log n) + 1$ steps to **Generate** the bits of the pair of (x, y) locations, and $2 \log n$ steps for the agents to **Echo** these bits back.

3.3.8 Discussion of the Reduction

It is intuitively easy to see that the Dec-MDP dynamics force agents to faithfully repeat the bits they have seen back, and to choose tile-types from a known consistent tiling, if they are to receive expected non-negative reward. In the **Select** and **Generate** phase, the agents are idle, as any actions they choose do not affect the system dynamics, and the only variables that change are those related to the initial selection of bits, and the revelation of their own tile-location. This has three important effects. First, since the transitions are otherwise random, the agents can not know, given what they have observed, what bit of their own location is being memorized by the state (recall, they know only the bit for the *other agent*). Second, since they can not affect the state-dynamics during these phases, nothing they can do in the way of action can serve as a signal, allowing them to communicate any revealed bits to one another, explicitly or implicitly.

Finally, since the FSA state-variable Q—which is observed by both—does not change at all in these phases, and is only affected by the bits they echo back later on, they have no information about the relative positions of the locations revealed, and have no choice but to use the known consistent tiling when picking a tile-type in the **Query** phase, lest they risk a negative reward later on. Further, during the **Echo** phase, since the agents still can not know which of their own location-bits has been memorized, the only policy guaranteed not to produce a negative-reward, incompatible-tiling situation is the one that repeats the bits back exactly as given to them. Any attempt to lie about these bits, even based upon an interim state of the FSA, risks negative reward along the way. Thus, Bernstein's reduction ingeniously forces any policy with non-negative value to correspond directly to a consistent tiling, as required. We will make use of similar reductions in our next results.

3.4 Other Subclasses of Interactions

We now prove our new results, establishing the complexity of certain other subclasses of the Dec-POMDP framework, including two apparently restricted problem instances that have been of some interest in the existing literature. From one point of view, our results are wholly negative; that is, we show that the NP-completeness result of the prior section is specific to the fully transition- and observation-independent problems considered there. When these independence properties are not present to the full extent, the worst-case complexity is once again in NEXP.

3.4.1 Reward-Independent-Only Models

We begin with a result that is rather simple, although it has not, to the best of our knowledge, been established before, and is worth noting. Namely, we consider the *inverse* of the NP-complete problem of Theorem 3.1; that is, rather than independent transitions and observations with shared rewards, we look at the complexity of problems for which rewards are independent, but transitions and observations are not. As we show, this returns complexity to the general case.

Theorem 3.4. Factored, reward-independent Dec-MDPs with n agents are NEXP-complete.

For our upper bound, as already described, we simply cite Theorem 3.2, which immediately establishes that such problems are in NEXP. For the lower bound, we modify the Dec-MDP from Bernstein's reduction proof so as to ensure that the reward-function factors appropriately into strictly local rewards.

Proof of Lower Bound. For our reduction, we create a Dec-MDP that is identical to that of the Bernstein reduction, in terms of its state-set S, transition function P, observation sets and functions O_i and Ω_i, and time horizon T. We are then required to show how to factor the state-space appropriately, as required by Definition 3.1, and re-design the reward function to be independent, in accords with Definition 3.4.

State Space Factorization. We divide the state space into three distinct (3) parts:

1. S_0 consists of the jointly observable variables, $\{phase, index, origin, Q\}$.

2. S_1 consists of all those variables indexed to agent α_1, with the addition of the tile-type variable for the other agent, α_2, $\{index_1, value_2, pos_1, tile_1, tile_2\}$.

3. S_2 consists of all those variables indexed to agent α_2, with the exception of its tile-type variable, $\{index_2, value_2, pos_2\}$.

Clearly, then $S = S_0 \times S_1 \times S_2$, as required. To avoid confusion, we stress that we have not changed which portions of the state-space are observable to each agent. In particular, each α_i still *does not observe* the location-bit values $index_i$ and $value_i$, instead observing those for the other agent. Thus, the system still requires them to echo back location bits that they have seen faithfully. Since we do not require that the problem be transition- or observation-independent, it is permissible for an agent to fail to fully observe their own local state; so long as *some* agent's observations determine that portion of the state space, all is well. Furthermore, it does not matter that the actions of α_2 can affect the local state-space of α_1 (via the tile-type selection process), since the problem is not assumed to be transition-independent.

As before, we write an agent's local state as a tuple, with semicolons separating variables according to how they are observed. Thus, the example global state:

$$s = \langle query, 5, F, q_0; 3, 1, 0, t_3; 4, 0, 1, t_5 \rangle$$

is the combination of state $\hat{s}_1 \in S_0 \times S_1$ for α_1: $\langle query, 5, F, q_0; 0, t_3; 4, 0, t_5 \rangle$, and state $\hat{s}_2 \in S_0 \times S_2$ for α_2: $\langle query, 5, F, q_0; 3, 1; 1 \rangle$. Once again, we use the symbol "$*$" as a wildcard, allowing us to write, e.g., $\hat{s}_1 = \langle sel, *, *, *; *, *; *, *, * \rangle$ to designate the any state for α_1 during the **Select** phase.

Reward Factorization. For each agent, we define a local reward function as follows. Each agent receives reward of -1 for all local state-transitions, except for a special designated set of transitions, for which they receive reward 0. Each agent remains idle in the same phases as before, and so receives reward 0 in those phases, independent of their action-choices. In the **Echo** phase, each reward function gives 0 reward if and only if the agents faithfully echo their own stored bit (the exact nature of which they are still unaware). In the **Test** phase, the second agent α_2 receives 0 reward, no matter what; however, the first agent α_1 receives 0 just in case the tiling pair selected by the two of them is compatible according to the defined TILING instance.

	$R(\hat{s}_1, a_1) = 0$ iff any of:	$R(\hat{s}_2, a_2) = 0$ iff any of:
Select:	$\hat{s}_1 = \langle sel, *, *, *; *, *; *, *, * \rangle$	$\hat{s}_2 = \langle sel, *, *, *; *, *; * \rangle$
Generate:	$\hat{s}_1 = \langle gen, *, *, *; *, *; *, *, * \rangle$	$\hat{s}_2 = \langle gen, *, *, *; *, *; * \rangle$
Query:	$\hat{s}_1 = \langle query, *, *, *; *, *; *, *, * \rangle$	$\hat{s}_2 = \langle query, *, *, *; *, *; * \rangle$
Echo:	$\hat{s}_1 = \langle echo, k, *, *; *, *; i_1, v_1, * \rangle$, $(a_1 = v_1$ or $k \neq i_1)$	$\hat{s}_2 = \langle echo, k, *, *; i_2, v_2; * \rangle$, $(a_2 = v_2$ or $k \neq i_2)$
Test (i):	$\hat{s}_1 = \langle test, *, o, \mathbf{equal}; *, \mathbf{t_1}; *, *, \mathbf{t_1} \rangle$, $o = F$ or $\mathbf{t_1} = t_0$	$\hat{s}_2 = \langle test, *, *, *; *, *; * \rangle$.
Test (ii):	$\hat{s}_1 = \langle test, *, *, \mathbf{hor}; *, \mathbf{t_1}; *, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in H$	
Test (iii):	$s = \langle test, *, *, \mathbf{ver}; *, \mathbf{t_1}; *, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in V$	
Test (iv):	$s = \langle test, *, *, \mathbf{apart}; *, *, ; *, *, * \rangle$.	

Finally, let our joint reward function simply be the sum of the local reward functions:

$$R(\langle s_0, s_1, s_2\rangle, a_1, a_2) = R_1(\langle s_0, s_1\rangle, a_1) + R_2(\langle s_0, s_2\rangle, a_2) = R_1(\hat{s}_1, a_1) + R(\hat{s}_2, a_2).$$

Correctness of the Reduction. It is relatively easy to see that any policy in this new reduced problem has an expected non-negative (0) value precisely when the same policy has expected non-negative value in Bernstein's original reduced problem (since all actions are the same, and rewards are negative or not in exactly the same circumstances). Thus, the correctness proof for the Bernstein reduction applies directly here: there exists such a non-negative policy if and only if the TILING instance has some consistent solution. □

Thus we see that in some respects, transition and observation independence are fundamental to the reduction of worst-case complexity from NEXP to NP. When only the rewards depend upon the actions of both agents, the problems become easier; however, when the situation is reversed, the general problem remains NEXP-hard. As we remarked before, this is not entirely surprising: much of the complexity of planning in decentralized domains stems from the necessity to take account of how one's action-outcomes are affected by the actions of others, and from the complications that ensue when observed information about the system is tied to those actions as well. The structure of rewards, while obviously key to the nature of the optimal (or otherwise) solution, is not as vital—even if agents can separate their individual reward-functions, making them entirely independent, other dependencies can still make the problem extremely complex.

We therefore turn to two other interesting special-case Dec-MDP frameworks, in which independent reward functions are accompanied by restricted degrees of transition- and observation-based interaction. These models attempt to create simpler problem instances by imposing further structure on the degree to which agents can affect one another, short of full independence. While some empirical evidence has suggested that these problems may be easier on average to solve, nothing has previ-

ously been shown about their worst-case complexity. We fill in these gaps, showing that even under such restricted dynamics, the problems remain NEXP-hard.

3.4.2 Event-Driven Interactions

The first model we consider is one of Becker et al. [9], which generalizes the notion of a fully transition-independent Dec-MDP, in a manner analogous to that used in the event-based reward structures discussed in Section 3.2. In this model, as before, a set of *primitive events*, consisting of state-action transitions, is defined for each agent. Such events can be thought of as occasions upon which that agent takes the given action to generate the associated state transition. *Dependencies* are then introduced in the form of relationships between one agent's possible actions in given states and another agent's primitive events; essentially, that is, an agent's actions are enabled (or not) due to the pre-existing occurrence (or not) of an event upon which it depends, modelled as a boolean variable. State-transitions are thus dependent upon the actions of other agents only insofar as some such dependence exists.

This model thus allows a simple counting approach to agent interactions, and the authors show that the Coverage Set algorithm introduced for fully transition-independent problem instances [9] in fact also applies to their less-restricted model in the presence of separate additive rewards. While no precise worst-case complexity results have been previously proven, the authors do point out that the class of problems has an upper-bound deterministic complexity that is exponential in the size of the state space, $|S|$, and doubly exponential in the number of defined interactions (which suggests nondeterministic exponential time complexity as an upper bound). This potentially bad news is mitigated by noting that if the number of interactions is small, then reasonably-sized problems can still be solved. Here, we examine this issue in detail, showing that, in fact these problems are NEXP-hard (indeed, NEXP-

complete); however, when the number of dependencies is a log-factor of the size of the problem state-space, NP-hardness is achieved.

We begin with the formal framework of the model. *Histories* (Definition 3.6), *events* (Definition 3.7), and *proper events* (Definition 3.8), are all as before. The model is a Dec-MDP with:

1. Two (2) agents.[8]
2. A factored state-space: $S = S_0 \times S_1 \times S_n$.
3. Local full observability: each agent α_i can determine its own portion of the state-space, $\hat{s}_i \in S_0 \times S_i$, exactly.
4. Independent (additive) rewards: $R(\langle s_0, s_1, s_2 \rangle, a_1, a_2) = R_1(\hat{s}_1, a_1) + R_2(\hat{s}_2, a_2)$.

Interactions between agents are given in terms of a set of *dependencies* between certain state-action transitions for one agent, and events featuring transitions involving the other agent. Thus, if a history contains one of the primitive events from the latter set, this can have some direct effect upon the transition-model for the first agent, introducing probabilistic transition-dependencies.

Definition 3.11 (Dependency)**.** A *dependency* is a pair $d_{ij}^k = \langle E_i^k, D_j^k \rangle$, where E_i^k is a proper event defined over primitive events for agent α_i, and D_j^k is a set of state-action pairs $\langle \hat{s}_j, a_j \rangle$ for agent α_j, such that each pair occurs in at most one dependency:

$$\neg(\exists\, k, k', s_j, a_j)\ (k \neq k')\ \ \&\ \ \langle s_j, a_j \rangle \in D_j^k \in d_{ij}^k\ \ \&\ \ \langle s_j, a_j \rangle \in D_j^{k'} \in d_{ij}^{k'}.$$

Such a dependency is thus a collection of possible actions that agent α_j can take in one of its local state, each of which depends upon whether the other agent α_i has made one of the state-transitions in its own set of primitive events. Such structures can be

[8]The model can be extended to n agents with little real difficulty. Since we will show that the 2-agent case is NEXP-hard, however, this will suffice for the general claim.

used to model, for instance, cases where one agent cannot successfully complete some task until the other agent has completed an enabling sub-task, or where the precise outcome depends upon the groundwork laid by the other agent.

Definition 3.12 (Satisfying Dependencies)**.** A dependency $d_{ij}^k = \langle E_i^k, D_j^k \rangle$ is satisfied when the current history for enabling agent α_i contains the relevant event: $\Phi_i \vDash E_i^k$. For any state-action pair $\langle \hat{s}_j, a_j \rangle$, we define a Boolean indicator variable $b_{\hat{s}_j a_j}$, which is true if and only if some dependency that contains the pair is satisfied:

$$b_{\hat{s}_j a_j} = \begin{cases} 1 & \text{if } (\exists\, d_{ij}^k = \langle E_i^k, D_j^k \rangle)\ \langle \hat{s}_j, a_j \rangle \in D_j^k \ \ \& \ \ \Phi_i \vDash E_i^k, \\ 0 & \text{otherwise.} \end{cases}$$

The existence of dependencies allows us to factor the overall state-transition function into two parts, each of which depends only on an agent's local state, action, and the status of the relevant indicator variable.

Definition 3.13 (Local Transition Function)**.** The transition function for our Dec-MDP is factored into two functions, P_1 and P_2, each defining the distribution over next possible local states: $P_i(\hat{s}_i' \mid \hat{s}_i, a_i, b_{\hat{s}_i a_i})$. We can thus write $P_i(\hat{s}_i, a_i, b_{\hat{s}_i a_i}, \hat{s}_i')$ for this transition probability.

When agents take some action in a state for which dependencies exist, they observe whether or not the related events have occurred; that is, after taking any action a_j in state s_j, they can observe the state of indicator variable $b_{\hat{s}_j a_j}$. (As Becker et al. [9] note, this can be changed so that agents observe the status of the indicator *before* they choose an action, if desired.)

With these definitions in place, we can now show that the worst-case complexity of the event-based problems is the same as the general Dec-POMDP class.

Theorem 3.5. Factored, finite-horizon, n-agent Dec-MDPs with local full observability, independent rewards, and event-driven interactions are NEXP-complete.

Again, the upper bound is immediate from the general case (Theorem 3.2). The event-based structure is just a specific case of general reward-dependence in a Dec-MDP, and such models can always be converted into Dec-MDPs where we define a joint transition function P, without using events in particular. Thus, if the more general Dec-POMDP class can be solved in nondeterministic exponential time, so can this special class. For the lower bound, we provide a reduction, again based on the Bernstein model, that builds in the special properties required.

Proof of Lower Bound. Unlike the previous reduction, in the proof of Theorem 3.4, we will need to add some variables to our state-space in this case, and alter the dynamics of the problem accordingly. In the original proof, each step of **Echo** consisted of both agents providing one of their location-bits, after which the problem advanced a step, and the FSA calculated its next state as it worked to determine the relative locations of the tiles given by the agents. The reward function provided a non-negative (0) reward just in case both agents were faithful in echoing back the memorized bits (observed by the other agent), and if the final result was a compatible pair of tiles. Such a reduction will not work directly for this problem for two reasons. First, the rewards given in the **Echo** phase depend upon how the actions of one agent relate to features (memorized location-bits) observed only by the *other* agent. Thus, our problem cannot be locally fully observed *and* locally reward independent, if we retain this feature. Second, the final reward, in the **Test** phase, the reward gained was based on the final state of the FSA, something observed by *both* agents, again defying local reward independence.

To solve the first of these problems, the intuitive idea will be to break each step in the **Echo** phase into sub-steps. Individual sub-steps will allow agents to act in sequence, after which the event-structure will ensure that there are local state features that lead to appropriate reward for the *other* agent. The second problem can be solved

similarly, by adding extra sub-steps to the **Test** phase, and making the FSA state a locally observable part of the individual state-space of only one agent.

States of the Problem. Our reduction Dec-MDP will feature the following state variables, again broken down according to how they are observed.

S_0: Both Agents. Three (3) variables are observed at each step. First are variables $phase \in \{sel, gen, query, echo, test\}$, and $index \in \{0, \ldots, 2 \log n\}$, indicating the state of the system and the next (x, y)-location bit to be generated or echoed, just as before. Third, there is $sub \in A, B, C, D$, indicating possible sub-steps of each part of the **Echo** or **Test** phases (as explained below).

S_1: Agent 1. This agent observes seven (8) variables of its own, as well. First, it observes $origin \in T, F$, and $q \in Q$, which are just the origin-point and FSA variables as before, although now only observed by the single agent. Second, it observes the index and value of the other agent's remembered location bit, $index_2 \in \{0, \ldots, 2 \log n - 1\}$ and $value_2 \in \{0, 1\}$, and its own position-bit and chosen tile-type, $pos_1 \in \{0, 1\}$, and $tile_1 \in L$, again as before. Last, it observes a variable $act_2 \in \{0, 1, \bullet\}$, that corresponds to possible bit-echo actions the other agent may take during the relevant phase and variable $choice_2 \in L \cup \{\circ\}$, corresponding to possible tile-choice actions by the other agent, to be used in the **Test** phase (with $\bullet$ and $\circ$ being "null" values, in each case).

S_2: Agent 2. This agent observes five (5) variables of its own, each of which is directly analogous to one of those observed by the first agent: $index_1$, $value_1$, pos_2, $tile_2$, and act_1. (Note that it does not observe the origin or FSA variables, nor does it have a recorder variable for the tile-choice of agent 1.)

Again, since the new variables each have a constant number of values, the set of all states $S = S_0 \times S_1 \times S_2$, is composed of factors, and has required size $|S| = \mathcal{O}(\log n)$, for suitably large n. Further, the problem is a locally fully observable

Dec-MDP, since each agent observes its own local portion in entirety. Again, we write any such state as a tuple of possible variable values, using a semicolon to separate each of the three classes of observed variables, and use the "$*$" convention to abstract over features to which we are indifferent. As before, the Dec-MDP begins in the **Select** phase, with neither agent having chosen a tile-type, and the FSA at its initial state; the *sub*, act_i, and $choice_2$ variables are set to default values: $s_0 = \langle sel, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ; 0, 0, 0, t_0, \bullet \rangle$

State Transitions. As in our prior reduction, agent actions can either echo location-bits, or choose tile-types, so that $A_1 = A_2 = \{0, 1\} \cup L$. For the first three phases, **Select**, **Generate**, and **Query**, the problem dynamics are essentially identical to the original version (Section 3.3.5). That is, **Select** chooses memorized location-bits at random, and **Generate** produces two otherwise-random sequences of location-bits for each agent to memorize, and all other variables remain the same. For these two phases, the transition-models are thus the same as before, with the addition of the *sub*, act_i, and $choice_2$ variables, which do not change throughout from the initial state. Since those transitions are indifferent to the actions that agents choose, the joint transition function given in the original can be trivially factored into the separate local transition functions, P_1 and P_2 as required. Similarly, in the **Query** phase, the action-choices affect only local state-variables, and so factorization of the transition function is straightforward. (In both cases, no events or dependencies are required, and so the indicator variables are false by default, meaning that all transitions are entirely independent of them, and they need not be specified at all.) Therefore, we do not present all the details here, as they are unnecessary; we simply note that the **Query** phase ends with a transition to the first state in the **Echo** phase, namely:

$$s' = \langle echo, 0, A; T, q_0, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle.$$

Each step in the previous **Echo** phase is now broken up into four sub-steps, A through D, as follows:

A: In this sub-step, the agents can choose whichever actions they like, and the state transitions to the next sub-step, regardless. (Note, however, that the action of agent 2 will be "recorded" in the next sub-step, based on a set of events that will be specified in the next section.)

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle,$

$\hat{s}'_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle.$

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle,$

$\hat{s}'_2 = \langle echo, k, B; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle.$

Again, there are no event-dependencies for these state-action pairs, so the boolean indicator variables need not be specified.

B: We define a set of dependencies, one per step $k \in \{0, \ldots, (2 \log n) - 1\}$ of the **Echo** phase. Letting $a_1^1, \ldots, a_1^m \in A_1$ be the possible actions for agent 1, each such dependency will be of the form $d_{21}^k = \langle E_2^k, D_1^k \rangle$ (that is, an event for agent 2, and a set of state-action pairs for agent 1), with:

$$E_2^k = \{(\hat{s}_2, 1, \hat{s}'_2)\}, \text{ and}$$
$$D_1^k = \{(\hat{s}_1, a_1^1), \ldots, (\hat{s}_1, a_1^m)\},$$

where $\hat{s}_2$, $\hat{s}'_2$ are the two local states of agent 2 as given in sub-step **A**, above, and $\hat{s}_1$ is the state for agent 1 as in the following:

$P(\hat{s}_1, a_1, b_{\hat{s}_1 a_1}, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,

$\hat{s}'_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, b_{\hat{s}_1 a_1}, \circ \rangle$,

where: $\mathbf{a_1} = 1$ if $a_1 = 1$, and 0 otherwise.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, B; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,

$\hat{s}'_2 = \langle echo, k, C; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$.

That is, agent 1 transitions to a state that depends upon the state of the indicator variable: if agent 2 chose action "1" at the prior sub-step **A**, then this will be recorded in agent 1's local variable act_2, otherwise a zero (0) will be recorded. In this step, we now employ the variable pos_1, previously unused in this phase, to record the action taken at sub-step **B** by agent 1 (in the next step we will also use dependencies to record this choice in the other agent's state-space, just as we have recorded agent 2's here). We note that the dependency-structure defined fits the requirements of Definitions 4.2 and 3.11. Since there is only a single atomic event for agent 2, indexed by a k-value that ensures that it does not repeat, in each dependency-event E_2^k, it is obviously proper. In addition, since each state-action pair for agent 1 in D_1^k also features an indexed k-value, each these appear in no more than one dependency.

C: In this sub-step, we use the same trick to record the prior sub-step's action for agent 1, using the local state-variable for agent 2, act_1. Again, we define a set of dependencies, one per step $k \in \{0, \ldots, (2 \log n) - 1\}$. Letting $a_2^1, \ldots, a_2^m \in A_2$ be the possible actions for agent 2, each such dependency will be of the form $d_{12}^k = \langle E_1^k, D_2^k \rangle$, with:

$$E_1^k = \{(\hat{s}_1, 1, \hat{s}'_1), (\hat{s}_1, 1, \hat{s}''_1)\}, \text{ and}$$

$$D_1^k = \{(\hat{s}_2, a_2^1), \ldots, (\hat{s}_2, a_2^m)\},$$

where $\hat{s}_1$, $\hat{s}'_1$, $\hat{s}''_1$ are the local states of agent 1 as given in sub-step **B**, above ($\hat{s}'_1$ and $\hat{s}''_1$ correspond to the two possible outcomes for boolean variable $b_{\hat{s}_1 a_1}$), and $\hat{s}_2$ is the state for agent 2 as in the following:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,

$\hat{s}'_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$.

$P(\hat{s}_2, a_2, b_{\hat{s}_2 a_2}, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, C; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,

$\hat{s}'_2 = \langle echo, k, D; i_1, v_1, 0, \mathbf{t_2}, b_{\hat{s}_2 a_2} \rangle$.

D: In this last sub-step, the system transitions to the next step of **Echo**, or to **Test**, as required. The variables for the origin (o) and FSA state (q) are updated in the local state of agent 1, based on the recorded actions of both agents.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,

$\hat{s}'_1 = \langle p, k', A; o', FSA(q, \mathbf{a_1}, \mathbf{act_2}), i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,

$$p, k' = \begin{cases} echo, k+1 & \text{if } 0 \leq k < (2 \log n) - 1 \\ test, 0 & \text{if } k = (2 \log n) - 1 \end{cases}$$

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, D; i_1, v_1, 0, \mathbf{t_2}, \mathbf{act_1} \rangle$,

$\hat{s}'_2 = \langle p, k', A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,

$o' = T \Leftrightarrow (o = T \ \& \ \mathbf{a_1} = \mathbf{act_2} = 0)$.

Thus, we have used our dependencies to allow each agent to record the other agent's actions in certain sub-steps of the **Echo** phase (which will allow us, later on, to separate the reward function in a proper local manner). We note that there are a total of $4 \log n$ dependencies in the formulation, each of which is of constant size (depending only upon the number of actions each agent possibly has). Again, then, the size of this additional information is $\mathcal{O}(\log n)$, for suitable values of n, the size of the TILING grid.

Finally, we extend the **Test** phase. Where before the process the system would take a single step and terminate, we here take three (3) separate sub-steps.

A. This sub-step is deterministic, independent of choices of agent actions. However, we will set up a dependency in the next step in order to record the action of agent 2; later, we will set up the reward functions to force that agent to repeat the same choice of tile-types it made in the **Query** phase in order to gain non-negative joint reward.

$P(\hat{s}_1,a_1,\hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle test, 0, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ\rangle,$

$\hat{s}'_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \circ\rangle.$

$P(\hat{s}_2,a_2,\hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle test, 0, A; i_1, v_1, 0, \mathbf{t_2}, \bullet\rangle,$

$\hat{s}'_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet\rangle.$

B. For this sub-step, we create a set of dependencies, one for each possible tile-choice action for agent 2. Letting $m = |L|$, we let $a_2^1, \ldots, a_2^m$ be this set of actions. Then, for each such action a_2^k, we create a dependency $d_{21}^k = \langle E_2^k, D_1^k\rangle$, with:

$$E_2^k = \{(\hat{s}_2, a_2^k, \hat{s}'_2)^1, \ldots, (\hat{s}_2, a_2^k, \hat{s}'_2)^r\}, \text{ and}$$
$$D_1^k = \{(\hat{s}_1, a_1^1), \ldots, (\hat{s}_1, a_1^u)\},$$

where each primitive event for agent 2, $(\hat{s}_2, a_2^k, \hat{s}'_2)^j$, is one of the possibilities given in the previous step (there will be different ones, given possible combinations of variables i_1, v_1, and $\mathbf{t_2}$, but again no two can ever occur in a single history, so the event will be proper). Also, $a_1, \ldots, a_1^u$ are the set of all possible actions for agent 1, and the single state $\hat{s}_1$ is as in the following:

$P(\hat{s}_1,a_1,\hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \circ\rangle.$

$\hat{s}'_1 = \langle test, 0, C; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2}\rangle,$

where: $\mathbf{t_2} = a_2^k \Leftrightarrow b_{\hat{s}_1 a_1} = 1.$

$P(\hat{s}_2,a_2,\hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet\rangle,$

$\hat{s}'_2 = \langle test, 0, C; 0, 0, 0, t_0, \bullet\rangle.$

Thus, this step will record the value of agent 2's prior action, allowing the reward function to give the right reward in the final sub-step of **Test**, based only on agent 1's state. Again, the number of necessary dependencies is bounded by the number of tile-types in L, and so will not require undue amounts of space to specify during the reduction.

C. Finally, the **Test** phase terminates, moving to an absorbing final state.

$$P(\hat{s}_1,a_1,\hat{s}'_1)=1 \text{ iff:} \qquad P(\hat{s}_2,a_2,\hat{s}'_2)=1 \text{ iff:}$$
$$\hat{s}_1=\langle test,0,C;o,q,0,0,0,\mathbf{t_1},\bullet,\mathbf{t_2}\rangle, \qquad \hat{s}_2=\langle test,0,C;0,0,0,t_0,\bullet\rangle,$$
$$\hat{s}'_1=\langle test,0,A;T,q_0,0,0,0,t_0,\bullet,\circ\rangle. \qquad \hat{s}'_2=\langle test,0,A;0,0,0,t_0,\bullet\rangle.$$

Local Reward Functions. We now present the factored, local reward function, similar to that one in the prior reduction (Section 3.4.1). In this problem, each agent again receives reward of -1 for all local state-transitions, except for a special designated set of transitions, for which they receive reward 0. Each agent remains idle in the same phases as before, and so receives reward 0 in those phases, independent of their action-choices. In the **Echo** phase, each reward function again gives 0 reward if and only if the agents faithfully echo their own stored bit (the exact nature of which they are still unaware). However, there is one catch, namely that the use of the sub-steps and recorder variables act_1 and act_2 allow us to give the reward for these choices to the *other agent*. Thus, agents can receive 0 reward even if they lie, but the do so knowing that they may cause the other agent to receive negative reward in future. The only joint policy with non-negative expected value is therefore one that forces both agents to tell the truth during **Echo**, just as before. As in the previous proof for the Reward-Independent-Only case, the **Test** phase only really applies to the first agent, who receives 0 just in case the tiling pair selected by both agents is compatible according to the defined TILING instance (again forcing the second

agent to choose from a consistent tiling solution, even if it personally accrues no local reward for doing so).

	$R(\hat{s}_1, a_1) = 0$ iff any of:	$R(\hat{s}_2, a_2) = 0$ iff any of:
Select:	$\hat{s}_1 = \langle sel, *, *; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle sel, *, *; *, *, *, *, * \rangle$
Generate:	$\hat{s}_1 = \langle gen, *, *; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle gen, *, *; *, *, *, *, * \rangle$
Query:	$\hat{s}_1 = \langle query, *, *; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle query, *, *; *, *, *, *, * \rangle$
Echo (A):	$\hat{s}_1 = \langle echo, *, A; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle echo, *, A; *, *, *, *, * \rangle$
Echo (B):	$\hat{s}_1 = \langle echo, *, B; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle echo, *, B; *, *, *, *, * \rangle$
Echo (C):	$\hat{s}_1 = \langle echo, k, C; *, *, i_2, v_2, *, *, \mathbf{act_2}, * \rangle$, ($\mathbf{act_2} = v_2$ or $k \neq i_2$)	$\hat{s}_2 = \langle echo, *, C; *, *, *, *, * \rangle$
Echo (D):	$\hat{s}_1 = \langle echo, *, D; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle echo, k, D; i_1, v_1, *, *, \mathbf{act_1} \rangle$, ($\mathbf{act_1} = v_1$ or $k \neq i_1$)
Test (A):	$\hat{s}_1 = \langle test, *, A; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle test, *, A; *, *, *, \mathbf{t_2}, * \rangle$, $a_2 = \mathbf{t_2}$
Test (B):	$\hat{s}_1 = \langle test, *, B; *, *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle test, *, B; *, *, *, *, * \rangle$
Test (C.i):	$\hat{s}_1 = \langle test, *, C; o, \mathbf{equal}, *, *, *, \mathbf{t_1}, *, \mathbf{t_1} \rangle$, ($o = F$ or $\mathbf{t_1} = t_0$)	$\hat{s}_2 = \langle test, *, *; *, *, *, *, * \rangle$.
Test (ii):	$\hat{s}_1 = \langle test, *, C; *, \mathbf{hor}, *, *, *, \mathbf{t_1}, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in H$	
Test (iii):	$s = \langle test, *, C; *, \mathbf{ver}, *, *, *, \mathbf{t_1}, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in V$	
Test (iv):	$s = \langle test, *, C; *, \mathbf{apart}, *, *, *, *, *, * \rangle$.	

Finally, let our joint reward function simply be the sum of the local reward functions:

$$R(\langle s_0, s_1, s_2\rangle, a_1, a_2) = R_1(\langle s_0, s_1\rangle, a_1) + R_2(\langle s_0, s_2\rangle, a_2) = R_1(\hat{s}_1, a_1) + R(\hat{s}_2, a_2).$$

Observations and Horizon. Again, we simply set the observation function for each agent to be deterministic: given a state, agent α_i simply observes the four variables that make up its portion of the state space, along with the global variables that make up the first portion. This also takes care of the requirement that agents be able to observe whether or not dependencies are satisfied after taking relevant actions, since each agent's state-space contains variables that determine the value of the indicator variables in the right spots. The time horizon is $T = (10 \log n) + 6$, consisting of one step each for the **Select**, and **Query** stages, three steps for **Test**, $(2 \log n) + 1$ steps to **Generate** the bits of the pair of (x, y) locations, and $4 \times (2 \log n)$ steps for the multi-step **Echo** phase.

Correctness of the Reduction. Again, it is straightforward that any policy in this new reduced problem has an expected non-negative (0) value precisely when the same policy has expected non-negative value in Bernstein's original reduced problem. The first three phases are exactly the same, since agents "idle" and accumulate 0 reward no matter what. Then, in the **Echo** phase, agents are again required to faithfully repeat observed location-bits: if they do not, then there is some positive probability that the other agent will receive a negative reward on a later sub-step. Finally, during the **Test** phase, sub-step **A** requires that agent 2 repeats the same tile-type it chose during **Query**. Finally, the last step simply replicates the final step of Bernstein's reduction, giving non-negative reward only for compatible tile choices. Thus, the correctness proof for the original reduction again applies directly: there exists such a non-negative policy if and only if the TILING instance has some consistent solution. □

3.4.2.1 A Special, NP-Hard Case

The reduction just presented relies on the fact that we allow the number of dependencies in the problem to grow as a factor of $\log n$, where n is the size of the grid in the original TILING instance. (We do not allow the set to grow any larger, lest the reduction no longer suffice for the complexity proof.) Since the overall size of the state-space S in the reduced Dec-MDP state-space is also $\mathcal{O}(\log n)$, this means that the number of dependencies is $\mathcal{O}(|S|)$. Thus, the NEXP-completeness result will hold for any Dec-MDP with event-based transitions where the number of dependencies is polynomial in the size of the state-space.

When we are able to restrict the number of dependencies further, however, we can do better in terms of the upper bound on worst-case complexity.

Theorem 3.6. A factored, finite-horizon, n-agent Dec-MDP with local full observability, independent rewards, and event-driven interactions are solvable in nondeterministic polynomial time (NP) if the number of dependencies is $\mathcal{O}(\log |S|)$, where S is the state-set of the problem.

Proof. Let $\mathcal{D}$ be a Dec-MDP as described, and let S be its state-set. Let $\mathbf{d}$ be the set of dependencies; by assumption, $|\mathbf{d}| = \mathcal{O}(\log |S|)$. We let $d_1, \ldots, d_{|\mathbf{d}|}$ be a set of new variables, one per dependency. For any agent α_i, we have that the state-set $S_i = S_0 \times S_i$ is some portion of the global state-set S. Further, for any α_i, we let $\mathbf{d}_i \subseteq \mathbf{d}$ be those dependencies $d^k_{ji} = \langle E^k_j, D^k_i \rangle$ that involve possible state-action pairs of that agent. Since agents can observe whether or not dependencies are fulfilled, we can augment the local state-space of each α_i with the dependencies for that agent, so that $S'_i = S_0 \times S_i \times \mathbf{d}_i$.

The result is a Dec-MDP with state-space $S' = S_0 \times S_1 \times \mathbf{d}_1 \times \cdots \times S_n \times \mathbf{d}_n$, where any local policy for an agent α_i is a mapping from local states $\hat{s}'_i \in S'_i$ to actions $a_i \in A_i$. (Since the local states now contain all information to which each agent has access during the course of any policy run, these now suffice. Given any joint

policy for the augmented problem, we can use dynamic programming to evaluate it in time that is polynomial in the size of the new state-space. Since the sum-total of all dependencies is $\mathcal{O}(\log n)$, we have that the size of the augmented state-space is polynomial in the size of the original: $|S'| = \mathcal{O}(|S|)$, and therefore that we can evaluate the policy in time that is polynomial in the size of the original, as well. Thus, the problem class can be solved in nondeterministic polynomial time. □

3.4.2.2 Discussion of the Results

These results are interesting for several reasons. First, the fact that the general event-based case is NEXP-complete, even in the presence of fully independent rewards and local full observability (Theorem 3.5), suggests that many interesting decentralized problems are potentially intractable. Becker et al. [9] show how the event-dependency model can be used to represent common problem structures in the hierarchical task modeling language TAEMS [37, 74, 132], which has been used in a number of real-world domains. The TAEMS framework allows us to represent a variety of interactions between sub-tasks in multiagent domains, including relationships where the actions and outcomes for one agent *enable* (make possible) or *facilitate* (ensure quality of) the possible outcomes for another agent. Since TAEMS is used in some practical systems, complexity analysis that can be extended to that framework is welcome.

In addition, the isolation of cases where complexity is likely to be lower (albeit still potentially NP-hard, as Theorem 3.6 shows) can help in determining what sort of task structures and agent interrelationships lead to intractability. In domains where the dependency structure can be kept simple relative to the overall state-space size, it may possible to derive optimal solutions in a reasonable amount of time. Both of these subjects are worthy of further study. In our final chapter, we will return to this issue briefly, suggesting how we might proceed from here.

3.4.3 State-Dependent Actions

Guo [54, 55, 56] considers another specialized subclass Dec-MDPs based on apparently even more restricted types of interaction. In this model, we again deal with separate agent state-spaces, and all agent action-transitions and rewards are independent of the actions of other agents. Such problems are not wholly decoupled, however, as the actions available to each agent at any point depend upon the global state of the system. Thus, agents interact by making choices that restrict or broaden the range of actions available to others.

Definition 3.14 (Dec-MDP with State-Dependent Actions)**.** An n-agent *Dec-MDP with state-dependent actions* is a tuple $\mathcal{D} = \langle S_0, \{S_i\}, \{A_i\}, \{B_i\}, \{P_i\}, \{R_i\}, T\rangle$, where:

- S_0 is a set of shared states, and each S_i is the state-space of agent s_i, with the global state space $S = S_0 \times S_1 \times \cdots \times S_n$. We let $s^0 \in S$ be the initial state.
- Each A_i is the action-set for α_i.
- Each $B_i : S \to 2^{A_i}$ is a mapping from *global states* of the system to some set of available actions for each agent α_i. For all $s \in S$, $B_i(s) \neq \emptyset$.[9]
- $P_i : (S_0 \times S_i) \times A_i(S_0 \times S_i)$ is the state-transition function over local states for α_i. The global transition function is simply the product of individual P_i.
- $R_i : (S_0 \times S_i) \to \Re$ is a local reward function for agent α_i. We let the global reward function be the sum of local rewards.[10]
- $T \in \mathbb{N}$ is the finite time-horizon of the problem.

---

[9]This requirement—that agents have some action-choice in every state—is a basic presumption of most Markov decision models. We note that it can always be relaxed by simply providing a "no-op" action that causes self-transitions in some state, having the same practical effect as an absence of actions. Guo also requires that $B_i(s) \subset S$, for all s, so that at every state there is some action that is not available. Since this requirement can be trivially satisfied in any problem by simply adding some new action that is never available anywhere, we choose to ignore it here.

[10]In the original formulation, this additive feature is not present, and the global reward function can be any arbitrary combination of local rewards. Since we will show our complexity for this more restricted version of the reward function, this will suffice for the more general case as well.

We note that there need be no observations in such a problem; since we presume local full observability, each agent's observations are just their local states. Furthermore, it is presumed that each agent can observe its own available actions in any state; a local policy is thus a mapping from local states to available actions.

For such cases, Guo presents a planning algorithm based on heuristic action-set pruning, along with a learning algorithm that combines pruning with Q-learning (on the latter, see [123, chap. 6]). While empirical results show that these methods are capable of solving potentially large instances, we again know very little analytically about the actual difficulty of solving problems with state-dependent actions. An NP-hardness lower bound is given [54] for the overall class, by reducing a normal-form game problem to the state-dependent but this is potentially quite weak, since no upper bound has been shown, and even the operative algorithmic complexity of the solution methods given is not well understood. We rectify this situation, showing that in fact the problem of determining whether a given instance of such a problem has a policy with non-negative value is also just as hard as the general case.[11]

Theorem 3.7. Factored, finite-horizon, n-agent Dec-MDPs with local full observability, independent rewards, and state-dependent action-sets are NEXP-complete.

Once more, we can rely upon the general upper bound on the complexity of Dec-POMDPs (Theorem 3.2). These sorts of Dec-MDPs are obviously special cases of the general model, which can easily incorporate state-based actions simply by allowing every action in every global state.

Proof of Lower Bound. This reduction will be very like the prior one, for event-driven interactions (Theorem 3.5). Again, we will use the trick of "recording" certain actions

[11] Again, in the original literature, the reward function for the problem is an arbitrary function of local rewards. Thus, the problem is more generally a *partially observable stochastic game* (POSG), and the initial complexity results are formulated in terms of equilibria, rather than optimal cooperative policies. Here, we show that the more restricted, purely additive, solution concept suffices.

of each agent in the state-space of the other, allowing us to have purely local rewards and local full observability. This time, however, we will use the idea of action-sets that are dependent upon the global state, rather than event-based dependencies, to enforce the dynamics we desire.

State-Space. We again give a reduction of any TILING problem to a 2-agent instance of the Dec-MDP with state-dependent actions. The state-space is exactly as for the prior reduction, and is divided the same way into the local, fully observable states for each agent. We do not repeat the description here, noting only that we again start with state $s_0 = \langle sel, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ; 0, 0, 0, t_0, \bullet \rangle$.

Actions and Transitions. As before, both agents possess actions $\{0, 1\} \cup L$. Unless otherwise specified, these actions are available for both agents in every state; exceptions, along with a set of additional actions for each agent are detailed below.

Again, our reduction works identically to the previous one through the first three phases (**Select**, **Generate**, and **Query**) of the problem, leading to the initial state of the **Echo** phase, which is again $s' = \langle echo, 0, A; T, q_0, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$. This phase is again divided into four sub-steps, A through D, with details as follows.

A: In this sub-step, the agents can choose whichever actions they like (all are available), and the state transitions to the next sub-step deterministically. The action of agent 2 is "recorded" by its own pos_2 variable, which in previous reductions was unused in this phase of the problem, remaining fixed ($pos_2 = 0$).

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,

$\hat{s}'_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,

$\hat{s}'_2 = \langle echo, k, B; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$,

$\mathbf{a_2} = 1$ if $a_2 = 1$ and is 0 otherwise.

B: We add four (4) actions to agent 1's action set A_1, $a^{00}, a^{01}, a^{10}, a^{11}$. Intuitively, we will use a^{ij} to allow us to record the situation where agent 1 echoes back bit-value i, given that agent 2 has echoed back bit-value j (in sub-step **A**). Formally, we enforce this by restricting its action-set function B_1:

$$B_1(\langle echo, *, B; *, *, *, *, *, *, *, *, *; *, *, 0, *, * \rangle) = \{a^{00}, a^{10}\}$$
$$B_1(\langle echo, *, B; *, *, *, *, *, *, *, *, *; *, *, 1, *, * \rangle) = \{a^{01}, a^{11}\}$$

Given this limitation, we can then record agent 1's own choice of bit in its own pos_1 variable, and agent 2's choice in agent 1's act_2 variable:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,

$\hat{s}'_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,

$\mathbf{a_1} = 1$ if $a_1 \in \{a^{10}, a^{11}\}$, and 0 otherwise,

$\mathbf{act_2} = 1$ if $a_1 \in \{a^{01}, a^{11}\}$, and 0 otherwise.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, B; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$,

$\hat{s}'_2 = \langle echo, k, C; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$.

C: In this sub-step, we use the same trick to record the prior sub-step's action for agent 1, using the local state-variable for agent 2, act_1. We add two actions, $a^{\bullet 0}$ and $a^{\bullet 1}$ to agent 2's action set A_2, and force it to choose exactly one of them, based on the variable in agent 1's own local state-space that records its prior action (from sub-step **B**).

$$B_2(\langle echo, *, C; *, *, *, *, 0, *, *, *; *, *, *, *, * \rangle) = \{a^{\bullet 0}\}$$
$$B_2(\langle echo, *, C; *, *, *, *, 1, *, *, *; *, *, *, *, * \rangle) = \{a^{\bullet 1}\}$$

Given this restriction, we can then force the recording of agent 1's choice of bit in agent 2's act_1 variable:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle,$

$\hat{s}'_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle.$

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, C; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle,$

$\hat{s}'_2 = \langle echo, k, D; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \mathbf{act_1} \rangle,$

$\mathbf{act_1} = 1$ if $a_2 = a^{\bullet 1}$, and 0 otherwise.

D: Finally, the system transitions to the next step of **Echo**, or to **Test**, as before. The variables for the origin (o) and FSA state (q) are again updated in the local state of agent 1, based on recorded actions of both agents.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle,$

$\hat{s}'_1 = \langle p, k', A; o', FSA(q, \mathbf{a_1}, \mathbf{act_2}), i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle,$

$$p, k' = \begin{cases} echo, k+1 & \text{if } 0 \leq k < (2 \log n) - 1 \\ test, 0 & \text{if } k = (2 \log n) - 1 \end{cases}$$

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle echo, k, D; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \mathbf{act_1} \rangle,$

$\hat{s}'_2 = \langle p, k', A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle,$

$o' = T \Leftrightarrow (o = T \ \& \ \mathbf{a_1} = \mathbf{act_2} = 0).$

Thus, just as before we used event-based dependencies, we have now used restrictions on the sets of actions to allow (indeed, *to force*) each agent to record the other agent's actions in certain sub-steps of **Echo**, again allowing us to later produce a fully factored local reward function. To do so only requires the addition of a fixed number of actions, independent of the state-space size, and cannot upset the reduction process by increasing specification size too much.

We use similar techniques in the **Test** phase of the problem, this time expanding each step to two separate sub-step (one less than in the prior reduction), and recording necessary information using action-set selection techniques.

A. In this sub-step, we record the tile-choice of agent 2 into the $choice_2$ variable, which is part of agent 1's local state-space. To do so, we first limit agent 1's

action-set to exactly one tile-type selection actions from L, directly corresponding to the tile chosen by the other agent. (Note that since we use actions that already exist in the original problem specification, this step does not require any increase in the problem size at all.)

$$B_1(\langle test, 0, A; *, *, *, *, *, *, *, *, *; *, *, *, \mathbf{t_2}, * \rangle) = \{a \in L \,|\, a = \mathbf{t_2}\}.$$

Then, we have state transitions as follows:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle test, 0, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle,$

$\hat{s}'_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2} \rangle,$

where: $\mathbf{t_2} = a_1$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle test, 0, A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle,$

$\hat{s}'_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet \rangle.$

B. As before, the **Test** phase terminates, moving to an absorbing final state.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:

$\hat{s}_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2} \rangle,$

$\hat{s}'_1 = \langle test, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ \rangle.$

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:

$\hat{s}_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet \rangle,$

$\hat{s}'_2 = \langle test, 0, A; 0, 0, 0, t_0, \bullet \rangle.$

Rewards, Observations, and Time Horizon. The reward function for this problem is essentially identical to that for the previous reduction (page 88), as the state-spaces are the same. The only difference is that we only have two sub-steps in the **Test** phase, rather than three. Our new reward function simply replaces the rewards previously given in sub-step **B** with those previously given in sub-step **C**, and eliminates the reference to an action for agent 2 in sub-step **A**, since that agent is now idle. As it is otherwise identical, we do not repeat it here. Observations are likewise as before

(under the general presumption that agents always know what action-choices they have). The time horizon $T = (10 \log n) + 5$, which is one step less than the prior problem, simply because we have eliminated a single sub-step in **Test**.

Correctness. Given the functionally identical reward functions, this reduction produces exactly the same non-negative policies as before. Thus, agents must still echo bits back reliably or risk negative rewards, and there is a policy with non-negative expected value just in case there is a consistent solution to the original TILING instance. □

3.4.3.1 Discussion of the Result

Guo and Lesser [56, 54] were able to show that the problem of deciding whether a problem with state-based actions had an equilibrium solution with value greater than k was NP-hard. (Again, this is because they allow a more general reward function, so that the problem may be competitive, and is in fact a form of partially observable stochastic game, or POSG.) It was not ascertained whether or not this lower bound was tight, however; since it was recognized that the general POSG class was NEXP-complete, this remained an important open question, given that NEXP-hard problems are considerably more difficult. Our result shows that this bound was indeed too low. Since an optimal joint policy will be an equilibrium for the special case of additive rewards, the general problem can be no easier.

This is interesting, for reasons that go beyond the satisfaction of a formal proof. These sorts of decentralized problems indeed appear to be quite simple in structure, requiring wholly independent rewards and action-transitions, so that agents can only interact with one another via choices that affect which actions are available. A typical example of such a problem is one in which two persons act completely regardless of one another, except for the fact that there is only a single rowboat, used for crossing a stream; if either agent uses the rowboat to get to the other side, then that action

is no longer available to the other. Another example involves two persons who share a work-space, although only the first of them has keys to the door: if that person unlocks the door, then both agents can freely make use of the space independently. In the first case, involving the rowboat, one agent's action can *eliminate* possible actions for the other; in the case of the door, unlocking it can *make possible* some actions that were not available before. These sorts of problems are intuitive, and common, and not all of them are hard to solve, obviously. Our results show, however, that the same structures can be intractable in the worst case. Since we can use the actions that are made available to encode all the information we need to pass between agents in the TILING reduction, even this seemingly simple form of interaction can lead to great complexity in solving.

3.4.4 Other Interaction Models

Here, we briefly mention a few other models and ideas from the literature. While we do not deal with them in detail, each is worth examining further in future.

Thomas [127, 128] defines an extension of the Dec-MDP model based on the idea of *direct interactions* between agent pairs. In this model, global states are again factored into local states of each agent. Then, along with the usual action-transitions and rewards, each pair of agents is furnished with a special subset of actions, namely the direct interactions. Such interactions affect only the local states of the pair of agents engaged in them, and take place only after that pair communicates in order to coordinate this local action-choice. Such a problem model thus introduces a restricted form of agent interaction, alongside a modicum of centralized control and coordination. Thomas presents problem domains—such as fire-bucket brigade coordination—that allow for the application of a specialized reinforcement-learning scheme, whereby agents pass on portions of a local reward to others during these direct interactions. Such a learning method effectively solves large multiagent problems that

are infeasible when fully decentralized. Only the most general complexity results are known for such a framework. Since the direct-interaction model is built upon the usual decentralized problem, worst-case complexity is trivially identical (NEXP-complete): if we leave the set of direct interactions empty, the resulting problem is just a Dec-MDP. Nothing is known about cases in which these sets are non-empty; of special interest is the case in which all interactions are of the direct variety.

Nearly Monotonic problems arise in distributed problem-solving domains, where information-processing tasks are divided between multiple agents, and the final global solution is composed out of local sub-solutions [24, 25]. Such problems are entirely monotonic when the composition problem is trivial, since local solutions are guaranteed to be globally valid. More interesting results are found when the local solutions do not so smoothly translate into global outcomes. Empirical results have shown that various measures of the degree of problem monotonicity can serve as heuristics for the performance of certain algorithms for distributed sensor interpretation and distributed diagnosis. Many nearly monotonic problems can be solved quite well using techniques which divide up a complex distributed problem and solve its simpler sub-parts using only local information; however, some nearly monotonic problem instances resist such solutions, and more analysis is necessary to understand why.

We would also like to examine more closely recent models of local interaction proposed by Oliehoek, Spaan, et al. [88, 122], where degrees of interaction are determined according to the connectivity properties of certain graphical models. While they have used techniques based on factored Markov Games to solve some relatively complicated instances, little is known about the real complexity of the domains.

3.5 Conclusions and Discussion

In this chapter, we have given an overview of a number of existing models for decentralized problem-solving. In each case, the models restrict the forms of agent

interaction in some way, in order to produce a special sub-case of the general Dec-POMDP problem. It has been known for some time that systems where agents act entirely independently apart from sharing rewards have reduced worst-case complexity. Somewhat unfortunately, we have shown that this does not apply to other variants, where we relax the independence requirements even only a little. In all of the cases we looked at, the new problem variants are as hard as the general case. This fact, when combined with results that show that many other contemporary models of decentralized problem solving are equivalent to the general Dec-POMDP model (as shown by Seuken and Zilberstein [116]), reveals the essential difficulty of optimal planning in decentralized settings.

At the same time, it must be stressed that the NEXP-complexity demonstrated here is a *worst-case* measure. Not all decentralized domains are going to be intractable, and indeed the event-based and action-set models we examine have been shown in many cases to yield to specialized solution methods, providing us with the ability to solve interesting instances in reasonable amounts of time. When the number of action-dependencies is small, or there are few ways that agents can affect available action-sets, it may well be possible to provide optimal solutions effectively. That is, the high worst-case complexity is no guarantee that *average-case* difficulty is likewise high. In later chapters, we expand upon this idea. On the one hand, it would be extremely difficult to provide precise theoretical bounds on average-case complexity for Dec-POMDPs; indeed, this sort of complexity measure is very hard to pin down in nearly any domain, since it is often difficult even to say what the "average" case looks like. On the other hand, we have developed some general measures of agent interaction that apply to all Dec-POMDPs, regardless of structural assumptions. As we will show later on, this measure has predictive power with respect to the actual empirical difficulty of solving problems, optimally or approximately.

CHAPTER 4

REWARDS, EVENTS, AND DIMENSIONALITY

The previous chapter showed that, outside of fully transition- and observation-independent Dec-MDPs, the worst-case complexity of even quite restricted models of interaction remains very high. As a result, in later chapters we will want to provide new tools for analyzing and identifying the average-case difficulty of problems, as encountered in practice. Before we move on to that part of the project, however, we provide some new results concerning the simpler (NP-complete) case. Our work here shows that in such problems, the structure of the shared-reward function corresponds precisely with essential problem dimensionality, as revealed when using mathematical programming techniques.

Such techniques extend the power to solve shared-reward Dec-MDPs. Becker's Coverage Set Algorithm (CSA), the first method established specifically to solve such problems, can work quite well when the reward-structure is specified concisely, but exhibits resource requirements that are exponential in the number of events used [12]. This can be an even greater problem, given the fact that while an event-based model may be highly intuitive, it can also be very inefficient, since it may be difficult to find the minimal number of events necessary to specify the system completely. Mathematical programming methods first explored by Petrik and Zilberstein [99] go a long way towards solving this problem. As we will show, these techniques allow us to compactify problems, reducing them only to their essential dimensions; furthermore, the size of the compactified problem directly determines the minimal number of events, and vice-versa.

Shen *et al.* [119] have suggested that complexity of a decentralized problem increases with the "degree of interaction" between agents. We develop these ideas in one particular possible direction, for the transition- and observation-independent problems with shared reward previously considered in Section 3.2. We describe one method of isolating the essential dimensionality of such Dec-MDPs via formulation as separable bilinear programs. This leads to some new results. We show how the bilinear programming version of the problem can be converted back into the event-based structure, and that doing so often reduces (and provably never increases) the number of events needed to describe the reward-structure, a key factor governing solution algorithm performance.

As we have somewhat simplified the presentation of the event-based model from Becker et al. [12], we begin by defining the specific type of Dec-MDP framework used in this work, and outlining the shared reward constraint structure that is the main source of problem complexity for such domains. Following that, we describe how such problems can be formulated and solved as bilinear programs, and reviews a method for compacting the problem to its essential dimensions. Next, we present proofs that demonstrate connections between the constraint structure and the essential dimensionality of a problem instance; it is shown how dimension compactification can be used to generate a compact constraint structure, reducing the hardest aspect of the problem as much as possible.

4.1 Decentralized MDPs

The class of problems first introduced by Becker, et al. [12], where the process is independent, but for shared influence on the joint reward, can alternatively be defined based on single-agent MDPs.

Definition 4.1. A *Markov decision process* is a tuple $\mathcal{M} = \langle S, A, P, R, \Delta_S, T \rangle$, with individual components as follows:

- S is a finite set of world states.
- A is a finite set of available actions.
- $P(s, a, s')$ is a state-transition function.
- $R : (S \times A) \to \Re$ is the reward function.
- Δ_S is the initial state-distribution.
- T is the finite time-horizon of the problem.

To define the shared reward structure of the multiagent Dec-MDP version, we require the following further notions.

Definition 4.2. For any MDP $\mathcal{M}$, an *event from* $\mathcal{M}$ is some set of state-action pairs,

$$\mathcal{E} = \{\langle s, a\rangle_1, \langle s, a\rangle_2, \ldots, \langle s, a\rangle_m\} \subseteq (S \times A).$$

When $\mathcal{E}$ is a singleton $\{\langle s, a\rangle\}$ we call $\mathcal{E}$ a *primitive event*, and we also refer to $\langle s, a\rangle$ itself as a (primitive) event.

This definition is a novel simplification of the original (Definitions 3.7 & 3.8), as we do not require the uniqueness conditions on proper events present there (although such conditions could be accommodated without affecting the results given here). Our notion of event is to be considered disjunctive, i.e. the event

$$\mathcal{E} = \{\langle s_1, a_1\rangle, \langle s_2, a_2\rangle, \ldots, \langle s_m, a_m\rangle\} \subseteq (S \times A)$$

can be thought of as a statement to the effect that an agent *performs action* a_1 *in state* s_1 OR *performs action* a_2 *in state* s_2 ... OR *performs action* a_m *in state* s_m.

Definition 4.3. For a pair of MDPs $\mathcal{M}^1$, $\mathcal{M}^2$, a *reward-constraint on* $\mathcal{M}^1$, $\mathcal{M}^2$ is a triple $c = \langle \mathcal{E}^1, \mathcal{E}^2, r_c\rangle$, where each $\mathcal{E}^i$ is an event from $\mathcal{M}^i$, and $r_c \in \Re$.

A reward-constraint is the basis for defining a shared dependency between the two processes $\mathcal{M}^1$ and $\mathcal{M}^2$. Again, such a constraint $\langle \mathcal{E}^1, \mathcal{E}^2, c \rangle$ can be regarded as a statement to the effect that *if event* $\mathcal{E}^1$ *occurs* AND *event* $\mathcal{E}^2$ *occurs, then the system receives additional reward* r_c. Such structures provide an intuitive definition of shared reward, and naturally describe many domains in which agents are engaged, for instance, in complementary or redundant subtasks. These problems allow separate execution, but can still make the overall system reward a complex function of combined agent behaviors, requiring coordination.

To properly define such a problem, the reward-constraints must obey a particular simple condition, however.

Definition 4.4. Let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be a set of reward-constraints on some pair of MDPs $\mathcal{M}^1$, $\mathcal{M}^2$. $\mathcal{C}$ is *feasible* iff all distinct reward-constraints are non-intersecting:

$$(\forall \langle s, a \rangle^1, \langle s, a \rangle^2)(\forall c_i, c_j)\big[\langle s, a \rangle^1 \in \mathcal{E}_i^1 \wedge \langle s, a \rangle^1 \in \mathcal{E}_j^1 \\ \wedge \langle s, a \rangle^2 \in \mathcal{E}_i^2 \wedge \langle s, a \rangle^2 \in \mathcal{E}_j^2\big] \;\Rightarrow\; (r_{c_i} = r_{c_j}).$$

That is, a feasible set of reward-constraints can never assign more than one reward to a pair of primitive events $(\langle s, a \rangle^1, \langle s, a \rangle^2)$. Note that such sets need not assign values to *all pairs* of primitive events; only that each such pair be assigned at most one supplementary shared reward. Such pairs define the problem's interaction structure.

Definition 4.5. For two agents x and y, a two-agent *factored and decentralized Markov decision process (Dec-MDP)* is a triple

$$\mathcal{D} = \langle \mathcal{M}^x, \mathcal{M}^y, \rho \rangle$$

where $\mathcal{M}^x$ and $\mathcal{M}^y$ are MDPs and ρ, the *shared-reward structure for* $\mathcal{D}$, is a feasible set of reward-constraints.

An optimal solution to a such a factored Dec-MDP is a pair of deterministic policies, π^x, π^y, one per agent, maximizing the expected sum of individual rewards ($R^i \in \mathcal{M}^i$) and joint reward (ρ).

Note that this type of factored and decentralized MDP is more properly called a *transition and observation-independent, locally and jointly fully observable Dec-MDP*; for convenience, in this chapter we simply refer to them as factored Dec-MDPs.

As we have pointed out before, these restricted versions of the general class are still useful for representing many real-world problems in which agents can work separately, without interfering with one another, but overall value of actions is a function of all the agents together. Examples include domains in which tasks can be divided into components that can be accomplished separately; given uncertainty about progress and outcome of subtasks, such problems still prove challenging. As we already outlined, Becker et al. [12] have shown that the problem of solving these factored Dec-MDPs optimally is NP-complete. While this significantly reduces worst-case complexity from NEXP-hardness, solution can still be quite difficult in practice. They apply a specialized method, the *Coverage Set Algorithm* (CSA), to such problems; they show that it can perform quite well on some cases, although it is not applicable to Dec-MDPs in general. The main hurdle in using the CSA on a given factored Dec-MDP comes from the shared-reward structure ρ: while most of the algorithmic heavy lifting is performed efficiently using linear programming and hill-climbing methods, the algorithm iterates exponentially in the number of reward-constraints, $|\rho|$.

This fact motivates the current results. As we will show, the structure of ρ is tightly bound to the *dimensionality* of a factored Dec-MDP, where this refers to the essential size of matrices necessary for solving the problems via mathematical programming methods. As ρ grows, so generally will the dimensionality. We give bounds on this growth, and then show how techniques for dimensionality compactification reduce the problem to only its essential (or dominant) dimensions. Further, we show

that such techniques can be used to generate new, often much smaller, shared-reward structures. These results provide firm connections between dimensionality and reward interactions in a Dec-MDP, and can reduce the complexity of the constraint structure, thus improving performance for algorithms like CSA that are highly sensitive to $|\rho|$.

4.1.1 An example of a Factored Dec-MDP

We present a simple example of a factored problem instance, to help make things clear, and for use in our later description of solution methods. In this domain $\mathcal{D}$, two agents x and y must make some delivery of goods of type a and b to one of two customers, c_1 and c_2. For each agent, the individual action-outcomes and rewards are given by two MDPs, $\mathcal{M}^x$ and $\mathcal{M}^y$. The particular details are unimportant; we simply note that specifying the MDPs separately, with separate transition and reward functions, means that they can be regarded as wholly independent from the point of view of each agent. Further, the techniques we present are able to easily solve each agent's independent sub-problem, based on the transition probabilities and reward functions of the individual MDPs.

However, the delivery problem contains one important source of dependency: the first customer, c_1, is willing to (1) pay \$2 extra for receiving two items, and (2) will give an additional \$2 bonus if it actually receives two different types of items. This shared bonus can be given in terms of the following feasible set of events (writing $\langle c_i, dl_j \rangle^k$ for the event of agent k delivering item type j to customer c_i).

$$\begin{aligned}\rho = \big[& \langle \langle c_1, dl_a \rangle^x, \langle c_1, dl_a \rangle^y, 2 \rangle, \\ & \langle \langle c_1, dl_a \rangle^x, \langle c_1, dl_b \rangle^y, 4 \rangle, \\ & \langle \langle c_1, dl_b \rangle^x, \langle c_1, dl_a \rangle^y, 4 \rangle, \\ & \langle \langle c_1, dl_b \rangle^x, \langle c_1, dl_b \rangle^y, 2 \rangle \big]\end{aligned}$$

$x \backslash y$	$\langle c_1, dl_a \rangle^y$	$\langle c_2, dl_a \rangle^y$	$\langle c_1, dl_b \rangle^y$	$\langle c_2, dl_b \rangle^y$
$\langle c_1, dl_a \rangle^x$	2	0	4	0
$\langle c_2, dl_a \rangle^x$	0	0	0	0
$\langle c_1, dl_b \rangle^x$	4	0	2	0
$\langle c_2, dl_b \rangle^x$	0	0	0	0

Table 4.1: The shared-reward structure for the delivery problem for agents x and y.

The shared-reward structure is therefore as shown in Table 4.1, which tracks the extra reward to be gained for the various state-action pairs for each agent. In general, any shared-reward structure can be represented in such a matrix form, where each entry corresponds to the shared-reward bonus for the corresponding pair of primitive events. Obviously, for any pair of MDPs $\mathcal{M}^x$ and $\mathcal{M}^y$, the size of this matrix representation is at most $|S^x|\,|A^x| \times |S^y|\,|A^y|$. Note also that this matrix only describes the shared reward for the relevant states and actions; there may be many more state-action pairs that play no role in the joint reward, but are part of the independent single-agent MDP planning problems. The policy for such a problem will involve actions for each agent in its own sequential planning problem—which might involve such things as planning local routes to various deliveries, for instance—while also maximizing overall reward based on the shared constraints.

4.2 Bilinear Programs and Factored Dec-MDPs

Petrik and Zilberstein [99] show how these restricted-interaction instances can be represented and solved as separable bilinear programs (for more details on separability, see Horst & Tuy [63]). We simplify their presentation somewhat here. For factored Dec-MDP $\mathcal{D} = \langle \mathcal{M}^x, \mathcal{M}^y, \rho \rangle$, we define the equivalent bilinear program:

$$\begin{aligned} \text{maximize} \quad & r_1^T x + x^T R y + r_2^T y \\ \text{subject to} \quad & A_x x = \Delta_{S^x} \qquad x \geq 0 \\ & A_y y = \Delta_{S^y} \qquad y \geq 0 \end{aligned} \tag{4.1}$$

Such a program is defined similarly to the dual linear program form for single-agent MDPs (see Puterman [104]). The vectors x and y are composed of variables corresponding to the possible state-action pairs from the two MDPs; we write $x(s, a)$ for the state-action pair corresponding to $s \in S^x$ and $a \in A^x$, for instance. Each linear reward-vector r_i in the objective function is simply the individual reward, taken from $R^i \in \mathcal{M}^i$. The matrices A_i encode state-visitation information, so that the multiplication in the constraints generates the original state distribution Δ_{S^i}, preserving total flow in the system for each state; for instance, multiplying vector x by A_x yields, for any $s \in S^x$,

$$\sum_{a \in A^x} x(s,\, a) - \sum_{s' \in S^x} \sum_{a' \in A^x} P(s \mid s', a')\, x(s', a') = \Delta_{S^x}(s).$$

Note that all elements so far are linear. However, we get generally non-linear behavior in the objective function via the matrix R, encoding the shared-reward structure of the factored Dec-MDP. This leads to NP-hardness in solving the overall problem, although methods have been found that work quite well in practice. Once the mathematical program has been solved, the agent policies for agent x can be extracted by letting $\pi^x(s) = a$ iff $x(s, a) > 0$, and similarly for y.

While any general Dec-MDP can be represented bilinearly in principle, it is only practical for either very small general problems, or for the special nearly-independent form given here. Koller and Megiddo [71, 72] consider the representation of extensive-form games in the form of linear complementarity problems (LCP, see [35]). Mangasarian [82] shows how such LCPs can in turn be represented as a separable bilinear program. Unfortunately, for the general problem class, this two-stage reduction is of

little practical use: variables take the form of possible *action-observation sequences* for each agent, and thus the resulting bilinear formulation is exponentially large in the size of the original Dec-MDP. (This is not a failure of the method, per se; evidently, given the NEXP-hardness of the original general class, this is unavoidable by any method in the worst case). Still, such reductions are possible in principle, and may lead to useful methods in some general cases. A similar approach is used by Aras et al. [5], who perform a similar sequence-form reduction in order to solve Dec-MDPs via mixed integer programs. Similarly, Amato et al. [2, 4] employ quadratically-constrained linear and non-linear methods to solve the general problem. These methods extend the ability to solve some general-form Dec-MDPs (and Dec-POMDPs), but are still limited by their inherent complexity.

The bilinear approach is particularly useful in the special case described here, where R is simply a reward matrix on state-action pairs. Especially interesting is the possibility for dimensionality reduction. As we show, this technique allows us to develop automated methods for reducing the size of reward-constraint formulations in factored Dec-MDPs, providing new hope for methods like CSA that scale poorly.

4.2.1 Dimensionality Reduction

We will refer to the *dimensionality* of a bilinear program for a factored Dec-MDP as in equation (4.1), by which we mean n, the size of the y-dimension of shared-reward matrix R. As we will describe, this dimensionality has been observed to dominate the complexity of solving such programs, and we will prove that it is tightly bound to the shared reward constraint structure. Note that in what follows, we assume that the original matrix R is a square $(n \times n)$ matrix, i.e. that x and y are both of length n; for two MDPs with differently sized state or action-sets, this can be enforced by padding out the smaller MDP with null actions and null states. This is trivial and

convenient. Note also that we could as easily perform all described operations along the x-dimension of R; nothing depends upon y.

Petrik and Zilberstein [99] prove that any given factored Dec-MDP can easily and automatically be reduced to its essential dimensions, based on shared-reward matrix R. That is, we can perform the following elementary matrix operations to eliminate all constant dimensions of y (along which the best response for agent y is the same for anything x does):

Eigenvector Generation: Generate the $(n \times n)$ matrix $R^T R$, and calculate the eigenvectors of $R^T R$. Since $R^T R$ is always a symmetric square matrix, these eigenvectors can be written in their orthonormal form.

Divide the Eigenvectors Let F be the matrix with columns formed by all the eigenvectors of $R^T R$ that have non-zero eigenvalues; let G be the zero-value eigenvectors. Let $[F; G]$ be the matrix of all eigenvectors, with all of F first (otherwise order of columns does not matter). Note that since $R^T R$ is symmetric and $(n \times n)$, $[F; G]$ is also an $(n \times n)$ matrix.

Generate the Inverse: Let $D = [F; G]^{-1}$. It is an elementary fact that such an inverse always exists, for any collection of the eigenvectors of a symmetric, square matrix, like $R^T R$. Let k be the number of columns in F (i.e., the number of non-zero eigenvectors of $R^T R$), let matrix D_k^T be the first k rows of D^T (i.e., the transposed inverse corresponding to those non-zero eigenvectors), and let matrix D_{k+1}^T be the remaining rows.

Separate Dimensions: Let $y_1 = D_k^T y$ and $y_2 = D_{k+1}^T y$. This separates out those dimensions of y that "matter" in our problem (y_1), from those that do not (y_2). Let $\langle y_1, y_2 \rangle$ be the vector composed of y_2 appended to y_1. (Note that the size of $[y_1; y_2]$ is just the same as the original, $n = |y|$.)

It is now elementary that the following form of mathematical program is equivalent to the original (4.1):

$$\begin{aligned} \text{maximize} \quad & r_1^T x + x^T RF y_1 + r_2^T [F;G]\langle y_1, y_2\rangle \\ \text{subject to} \quad & A_x x = \Delta_{S^x} \\ & A_y [F;G]\langle y_1, y_2\rangle = \Delta_{S^y} \\ & x \geq 0 \quad y_1 \geq 0 \quad y_2 \geq 0. \end{aligned} \tag{4.2}$$

It is easy to verify, given the construction of y_1 and y_2 by inverse multiplication, that $[F;G]\langle y_1, y_2\rangle = y$, and so this formulation respects the original individual reward function r_2 for agent y, and the original constraints on distribution of states, Δ_{S^y}. What is interesting, however, is that we can replace the original $(n \times n)$ joint-reward matrix R in (4.1) with the $(n \times k)$ matrix RF here. Further, when k, the dimensionality of F, is small, and $R^T R$ has few non-zero eigenvectors, this can be a substantial savings. Additionally, we can then go back to the original problem formulation, and replace the reward-constraint structure with a new one, often smaller, as we describe below. This means that we can preserve the often more intuitive structure, based on events, and use algorithms exploiting this sort of structure.

4.2.2 Application to Our Example Problem

To see how this works in practice, let us consider again our simple delivery problem, with a 4-dimensional shared-reward matrix as found in Table 4.1:

$$R = \begin{bmatrix} 2 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad R^T R = \begin{bmatrix} 20 & 0 & 16 & 0 \\ 0 & 0 & 0 & 0 \\ 16 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The non-zero eigenvectors of $R^T R$ are thus the columns of:

$$F = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 \end{bmatrix}$$

In this case, further, the inverse-row-matrix $D_k^T = F^T$; it is important to note that this will not hold in general, although D_k^T always exists and is easily calculated.

Thus, we have the following (using $y(i,k)$ to abbreviate the event of y giving item k to customer i):

$$RF = \begin{bmatrix} 3\sqrt{2} & -\sqrt{2} \\ 0 & 0 \\ 3\sqrt{2} & \sqrt{2} \\ 0 & 0 \end{bmatrix} \quad y_1 = D_k^T y = \begin{bmatrix} \frac{y(1,a)+y(1,b)}{\sqrt{2}} \\ \frac{y(1,a)-y(1,b)}{\sqrt{2}} \end{bmatrix}$$

Our new joint-reward matrix RF is now 2-dimensional, and has only two y_1-variables, each a linear combination of pre-existing variables. One can easily confirm that the minimized reward function is identical to the original (that is, $x^T R y = x^T RF y_1$), and so the resulting objective function is equivalent to the original. It is also easy to generate remaining components G and y_2, and confirm that all other operations preserve equivalent problem input and output.

For an example like this, the dimensionality reduction is not very surprising; clearly, in the original specification of R, deliveries to the second customer play no role in maximizing the shared reward. No extra reward is received for deliveries to c_2, and the columns $y(2,*)$ and rows $x(2,*)$ are all empty (0). This is not generally the case, however; the method does not amount to simply ignoring columns that are all 0. There will be many cases in which no columns or rows of the original R are empty, and yet we can still compactify.

Furthermore, this example shows an important, and less obvious, new feature of the dimension reduction process, namely the compactification of the overall reward-constraint structure. While prior work has been interested in converting event-based Dec-MDPs into the bilinear formulation solely as a means of solving them, we can now go a step further. That is, we can convert the reduced bilinear form *back into* the reward-constraint formulation, with the possibility of a substantive savings in the size of the structure ρ.

Even though our original problem was very small, we are still able to re-write it using a smaller ρ than was possible before. Comparing the two matrices for the reward function:

$$R = \begin{bmatrix} 2 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad RF = \begin{bmatrix} 3\sqrt{2} & -\sqrt{2} \\ 0 & 0 \\ 3\sqrt{2} & \sqrt{2} \\ 0 & 0 \end{bmatrix}$$

we see that in original matrix R, no non-zero column or row contains any repeated values. Thus, the original shared-reward structure ρ is minimal with respect to its elementary events. That is, ρ needs four distinct entries ρ to specify R. In the case of RF, however, this is not true, since the first column, corresponding to new variable $(y(1,a) + y(1,b))/\sqrt{2} \in y_1$, contains only a single value, $3\sqrt{2}$. It follows that we can write a new reward-constraint structure in terms of the new, compound event-variables in y_1, featuring only 3 entries. Even on this small, nearly minimal example, then, we have reduced the minimum number of constraints necessary to describe the joint-reward structure. For algorithms like CSA, exponential in this value, this can significantly improve performance.

4.3 Constraints and Dimensionality

As we now show, these reductions in the overall dimensionality and size of the reward-structure are not accidental: we can relate basic properties of the minimal constraint structure for a problem to the dimensionality of its reduced bilinear form, to establish that the reduced form will always be no larger than the original.

We first establish an upper bound upon the *essential dimensionality* of a factored Dec-MDP, by which we mean the dimensionality of the reward matrix RF in the compactified bilinear form; we write $k[\mathcal{D}]$ for the essential dimensionality, with k equal to the number of columns of matrix RF.

Theorem 4.1. Let $\mathcal{D} = \langle \mathcal{M}^x, \mathcal{M}^y, \rho \rangle$ with

$$\rho = \left[\langle \mathcal{E}_1^x, \mathcal{E}_1^y, r_1 \rangle, \langle \mathcal{E}_2^x, \mathcal{E}_2^y, r_2 \rangle, \ldots \langle \mathcal{E}_m^x, \mathcal{E}_m^y, r_m \rangle \right]$$

and let $Y_\rho = \cup_{i=1}^m \mathcal{E}_i^y$. Then $k[\mathcal{D}] \leq |Y_\rho|$.

That is, the essential dimensionality of the problem is bounded on top by the size of the set formed from the union of all y-events in ρ. While this bound may be loose, it can often provide a good guide to the overall essential complexity of a given factored Dec-MDP $\mathcal{D}$.

Proof. Let the length of our original y-vector be n (so the dimensionality of unreduced matrix R is also n). Now consider any primitive event $\langle s, a\rangle_j^y \notin Y_\rho$; since this event is not featured in any constraint in ρ, we have that column j of R is all 0's. Thus, column j of $R^T R$ is all 0's, and so there exists a unitary vector

$$v_j^0 = [0_1\ 0_2\ \cdots\ 0_{j-1}\ 1_j\ 0_{j+1}\ \cdots\ 0_{n-1}\ 0_n]^T$$

(i.e. 0's in all places, and 1 in place j), which is a 0-value eigenvector of $R^T R$. For each such $\langle s, a\rangle_j^y \notin Y_\rho$, such a distinct v_j will exist; each will be orthogonal, and the entire collection can be put into orthonormal form. Thus the size of the set G of all 0-value eigenvectors of $R^T R$ will be at least the size of the complement of Y_ρ, $|G| \geq (n - |Y_\rho|)$. Therefore, since $|F| = (n - |G|)$, $k[\mathcal{D}] = |F| \leq |Y_\rho|$ □

Thus, for any factored Dec-MDP $\mathcal{D}$, we can bound the dimensionality of the reduced form in advance. In the worst case, all primitive events are featured in the reward-structure, and $Y_\rho = \{y(s, a) \mid s \in S^y,\ a \in A^y\}$, so this bound will simply be n. Of course, the worst case for compactification is that all eigenvalues of $R^T R$ are non-zero, and dimensionality is in fact n. (This is equivalent to invertibility of R).

A more interesting result concerns the opposite direction, namely bounding the size of the minimal constraint structure for a factored Dec-MDP based upon the reduced bilinear representation. As noted, in Section 4.2.2, our example can be written using 3 constraints once in compact bilinear form, rather than the original 4. This point can easily be made general.

Fact 1. Let $\mathcal{D}$ be a factored Dec-MDP written in reduced form (4.2), with compactified shared-reward matrix RF. For any column i of RF, let $u_i[RF]$ be the number of unique values occurring in that column. Then $\mathcal{D}$ can be written in an equivalent form $\mathcal{D}^-$, using reward structure ρ^- with size:

$$\left|\rho^-\right| = \sum_{i=1}^{|RF|} u_i[RF].$$

We can see this from our example problem, where the reward structure will be:

$$\begin{aligned}\rho = \big[&\langle \{x(1,a),\, x(1,b)\},\, \frac{y(1,a)+y(1,b)}{\sqrt{2}},\, 3\sqrt{2}\rangle, \\ &\langle x(1,a),\, \frac{y(1,a)-y(1,b)}{\sqrt{2}},\, -\sqrt{2}\rangle, \\ &\langle x(1,b),\, \frac{y(1,a)-y(1,b)}{\sqrt{2}},\, \sqrt{2}\rangle\big]\end{aligned}$$

In general, for any column of RF corresponding to a compound event variable $y^- \in y_1$, and any unique value u in that column, reward structure ρ^- requires one constraint. Each such constraint will be of the form

$$c = \langle \mathcal{E}^x,\, y^-,\, u\rangle$$

where $\mathcal{E}^x$ is the set of all state-action pairs $x(s,a)$ corresponding to rows of RF in which value u appears. This allows us to easily bound the general size of the reduced shared-reward structure.

Fact 2. Let factored Dec-MDP $\mathcal{D} = \langle \mathcal{M}^x,\, \mathcal{M}^y,\, \rho\rangle$, be written in equivalent form $\mathcal{D}^-$ as just described. We have an upper bound on the size of the reduced shared-reward structure for the re-written problem:

$$\left|\rho^-\right| \leq |S^x| \times |A^x| \times k[\mathcal{D}].$$

Proof. This is a straightforward application of Fact 1. Since the size of the structure is $|\rho^-| = \sum_{i=1}^{|RF|} u_i[RF]$, and the number of unique values in any column of RF is at most $n = |S^x| \times |A^x|$ (i.e., simply the number of rows in RF, equal to the size of vector x). The result is then obvious, since the number of columns in RF is simply the number of columns in F, i.e. the essential dimensionality $k[\mathcal{D}]$. □

Along with these basic bounds, we can also prove something far more significant about the shared-reward structure of a compactified, factored Dec-MDP $\mathcal{D}$. In particular, we show that putting $\mathcal{D}$ in the reduced form (4.2), and then rewriting it in terms of the induced reward structure, can only reduce the number of necessary constraints.

Theorem 4.2. Let $\mathcal{D}$ be a factored Dec-MDP with $|\rho| = n$; let $\mathcal{D}^-$ be the compactified bilinear form of the problem, and ρ^- be the resulting constraint structure, as described above. Then we have the following:

$$\left|\rho^-\right| \leq |\rho|$$

The full proof of Theorem 4.2 requires two parts. We must show that the original formulation of $\mathcal{D}$ must contain at least one distinct constraint for (i) every column of RF, and (ii) every distinct value in that column. The first is easily shown; here, we prove the second, since it is more interesting.

Proof. Consider any column c of RF, and suppose it contains two distinct values $c_i \neq c_j$. Let $v_c \in F$ be the column eigenvector of F that generated column $c \in RF$ (i.e., $c = Rv_c$). Thus, since

$$c_i = \sum_{k=1}^{n} r_{ik} v_c \quad \text{and} \quad c_j = \sum_{k=1}^{n} r_{jk} v_c$$

there must exist column k^* of R such that $r_{ik^*} \neq r_{jk^*}$ (else $c_i = c_j$). Therefore, in the original problem formulation of $\mathcal{D}$, the specification of R in terms of events will require two separate and distinct constraints:

$$c_1 = \langle \mathcal{E}_1^x = \{\langle s, a \rangle_i^x, \ldots\}, \mathcal{E}_1^y = \{\langle s, a \rangle_{k^*}^y, \ldots\}, r_{ik^*} \rangle,$$
$$c_2 = \langle \mathcal{E}_2^x = \{\langle s, a \rangle_j^x, \ldots\}, \mathcal{E}_1^y = \{\langle s, a \rangle_{k^*}^y, \ldots\}, r_{jk^*} \rangle.$$

Thus, each distinct value in any column of RF corresponds to at least one constraint in the original problem. □

Thus, the reduction in number of necessary constraints observed for our example problem is no accident. Rather, the three-stage process of (1) conversion into bilinear form, (2) dimensionality reduction, and (3) re-conversion into event-based reward-constraint form, will never increase the size of the problem specification (since it only ever shrinks ρ, and leaves all else alone).

4.4 Practical Applications

These techniques are of more than formal interest. Ongoing research by Petrik has applied the presented techniques to a number of domains, including the multiagent broadcast-channel and tiger problems—standard benchmarks used, for example, in recent work by Aras et al. [5]—and a common Dec-MDP formulation of a Mars rover robot exploration problem, as used in Becker, et al. [9]. The reduction method has been shown to reduce the number of events necessary to specify a wide range of these domains. In the broadcast domain, dimensionality (and the number of necessary events) is reduced to 3 no matter what the original problem size, providing a potentially very large reduction from the event-based specification. In the rover case, many irrelevant events are eliminated, reducing to one (1) for each site that two

rovers both explore, out of many initial events involving all possible sites; additionally, in particular instances the number of events may further be reduced even more significantly, with very little resulting error. Finally, when applied to instances of the decentralized tiger problem, the number of events is reduced by about a factor of 5, from 108 to 20, with a reward loss of at most 2%. Since even linear reductions in the number of events provides exponential possible speed-ups for algorithms like the CSA, this transforms such problem instances from ones that are simply infeasible to those that can be practically solved after all.

4.5 Conclusions and Discussion

As we have shown, the reduction process allows us to potentially eliminate constant dimensions for one agent's actions, and also rewrite the problem in terms of a smaller reward structure. While the method of converting into bilinear program and doing dimensionality reduction was already known, this work is the first to show how to move back to the original form, and how that affects problem size. This is of both theoretical and practical interest.

In analytical terms, this method allows us to reveal the essential structure of dependencies between agents in a factored Dec-MDP. By converting to the reduced form, we can find a minimal set of events suitable for representing a domain. The event-based formulation is very convenient and intuitive, but can be highly inefficient. While simple techniques for merging events exist, they are limited. In fact, as we have shown, problems can be such that there is simply no way of reducing the size of the event formulation, so long as we use state-action pairs. This poses a serious roadblock to the use of methods like the Coverage Set Algorithm, which explicitly iterates based on separate constraints. Our process of reduction allows problems to reduce this size, often dramatically.

Of course, it may be hard to look at a linear combination of state-action pairs, as generated by our method, and see how this relates to the structure of the original problem. That is, it is difficult to interpret the weighted combination of elementary events produced by compactification. One possibility for future work concerns factored Dec-MDPs for which this problem of interpretation is much easier. In such cases, the partial inverse matrix D_k^T is of a special form, and our new reduced problem can be expressed in terms of simple events from the original problem, while still reducing the maximum number of constraints generated. These sorts of extensions have many possible practical applications, since they can provide ways of automatically reconfiguring large and complex multiagent system specifications, eliminating unnecessary events and reward-constraints from consideration.

Finally, we note that is straightforward to extend this approach to problems with more than two agents, if rewards depend on pairs of agents and the dependency graph is bipartite. In this case, the problem is again formulated bilinearly. An extension to general multiagent problems is more problematic. A possible approach may rely on a multilinear program formulation, and then applying a tensor version of singular value decomposition (SVD). The problem is that in general, these methods are often NP-complete, unlike two-dimensional SVD, which can be done in polynomial time.

We do not pursue these other ideas at the present time, since they are not the main focus of our work here. Instead, we move on from the use of restricted models, to a more general account of agent interaction. This allows us to tackle the problem of complexity more directly, by looking for measures of expected practical difficulty that do not depend upon assuming special kinds of structures to the system dynamics.

CHAPTER 5

AN INFORMATION-THEORETIC TREATMENT OF INTERACTION AND PROBLEM DIFFICULTY

The previous chapters have presented a number of recent attempts to simplify the solution complexity of Dec-POMDP models by restricting the ways in which agents may interact with one another. We are now interested in a more unified picture of agent interactions. Such an account would look at interactions in general terms, considering (among other things) the following questions:

1. How do we *identify* interactions at all?
2. How do we *quantify their effect* on
   (a) the *quality* of a solution outcome?
   (b) the *difficulty of finding* a solution outcome?

The various special models already considered each identify interactions differently, Now, we want to move beyond that more piecemeal approach, looking instead for a widely applicable means of evaluating how agents influence both one another and the overall problem dynamics. Our approach is both analytical and empirical. On the one hand, we consider general properties of agent interaction measures. On the other, we examine exactly how these measures correlate with the practical difficulty of applying various optimal and approximate algorithms to Dec-POMDPs.

5.1 The Effect of Interactions in General

All existing theoretical and empirical work points to the fact that the practical complexity of solving Dec-POMDPs depends upon interactions between the state-

transitions, observations, and rewards of different agents; the precise nature of this connection is the focus of our current line of investigation. As we have pointed out, if a problem is completely separable into individual agent problems, complexity is no longer a fundamental problem. Work on problems that are partially independent (Section 3.1) shows that limiting actions can in fact reduce elementary complexity, but our further complexity proofs have shown how restricted these results can be.

This raises another interesting question about agent interactions, namely how they affect a problem's overall complexity. We seek a general framework, or at least a set of properties, applicable to all Dec-POMDPs, and want to know how that general account relates to the difficulty of solving them. Prior work has suggested that the complexity of a decentralized problem increases in proportion to the "degree of interaction" between agents, although a fully precise definition of this measure has not yet been proposed [119]. In the game theory literature, it has been shown that bounding the influence any agent can have on overall reward simplifies the process of finding equilibria [69]. In the context of MDPs, it has been shown that the performance of learning algorithms is inversely proportional to information-theoretic measures of the overall system dynamics [109]. Combining these ideas, a more complex hypothesis is that performance of solution techniques can be improved in cases where we restrict the effect agents can have on the overall expected value of joint policies, which in turn will depend upon the probabilistic dynamics of the Dec-POMDP system. In this chapter, we make a first attempt at providing a more general account of such restricted interaction, and in the next we provide evidence that the hypothesized correlation exists: as problems grow more centralized, they tend to get easier.

5.2 Information and Action in Decentralized Planning

The need to account for the actions of other agents presents one of the primary sources of complexity and problem difficulty for decentralized planning. Methods—

optimal or approximate—that compare and evaluate one agent's policies against those of others soon run into difficulties, as the number of necessary comparisons grows too large. This is in some respects unavoidable: since the outcomes of one agent's actions interact with those of others, intelligent planning cannot avoid paying attention to multiple policies at once. In previous chapters, we considered event-based models of interaction, which essentially amount to ways of counting how often agents interact, isolating particular intersections between their courses of action to find (potentially) simpler problem instances. Here, we look at another way of quantifying interaction in a multiagent problem. Instead of focussing on particular single interactions, we investigate a way of measuring *overall interaction*, in terms of *information relations*.

Information is central to problem complexity. When we plan in a decentralized environment, we only ever need to take other agents into account in those cases where our course of action would be different, depending upon what those others were doing. We base our plans on the possible actions of those who can affect our own outcomes, something that becomes especially difficult when we do not have full knowledge of what they are actually doing. That is, devising and executing plans is complicated by a lack of information about the actions of others. Of course, a *lack* of information can only be felt where the *presence* of information would actually be helpful: where knowing the actions of others might make some kind of difference to what we do.

Here, we present a precise way of measuring this kind of informational relationship, evaluating how much knowledge of a given agent's actions can tell us about the dynamics of the overall system. We show how to quantify the total possible influence an agent has on the system in such information-theoretic terms, and demonstrate, both analytically and empirically, how this sort of influence is related to the difficulty of solving a multiagent problem. As shall become clear, these sorts of information measures allow us to quantify such otherwise vague or general notions as "influence"

or "centralization," and correlate well with the performance of both optimal and approximate solution algorithms.

5.3 Quantifying Agent Influence

In order to measure the degree of influence and interaction between agents, we have devised a means for grading the difference between agent effects upon system dynamics, called the *influence gap*. First, we determine the total effect each agent has on the Dec-POMDP system, in terms of the amount of information about those dynamics that is carried by that agent's actions. Then, we measure the difference between these various levels of influence (the "gap"). As we shall show, the individual information-measure proposed has some interesting formal properties. Furthermore, the gap measurement generally correlates with the empirical performance of both optimal and approximate algorithms. In particular, as this gap shrinks to zero, for cases where agents have equal (non-zero) influence over the problem dynamics, problems are generally harder to solve. Conversely, as the gap grows, and one agent comes to dominate the system—that is, as a single agent has more and more control over the outcomes in the Dec-POMDP—problems generally prove easier to solve.

5.3.1 Entropy and Mutual Information

We begin with a few standard definitions and results from Information Theory. Originally defined by Shannon [117, 118], these allow us to measure both the amount of potential information carried by a random variable, and the amount of information shared between two such variables. We do not discuss the motivation for measuring information as given here, but note that the Shannon measure proves to be a natural one; an interested reader may turn, for instance, to the textbook of MacKay [80] (any standard treatment of the subject will contain the same definitions and propositions).

Definition 5.1 (Shannon information content)**.** Let x be a possible outcome for some random variable X, occurring with probability $P(x)$. The *Shannon information content of* x is given by:

$$h(x) = \log \frac{1}{P(x)}. \tag{5.1}$$

When the logarithm used to define information is the base-2 log, we refer to the information content $h(x)$ (and all other measures given in what follows) as being measured in *bits*; when the natural logarithm is used, the measure is said to be in *nats*. Other than a scaling factor, it is indifferent which units are actually used. Given the information content for individual outcomes, we can measure the overall information content of a variable, referred to as its *entropy* or *uncertainty*.

Definition 5.2 (Entropy)**.** For random variable X, with outcomes $x \in X$, each occurring with probability $P(x)$, the *entropy of* X is the average information content of any outcome:

$$H(X) \equiv \sum_{x \in X} P(x) \log \frac{1}{P(x)}, \tag{5.2}$$

using the convention that when $P(x) = 0$, we set $0 \times \log 1/0 = 0$.

Entropy has two basic properties, establishing its maximum and minimum values. Each is straightforward to establish, and no proofs are provided here.

- Entropy is maximized when X is uniformly distributed: $H(X) \leq \log |X|$ (where $|X|$ is the number of possible values $x \in X$), and $H(X) = \log |X|$ just when $\forall x \in X,\ P(x) = 1/|X|$.

- Entropy is bounded below by 0 ($H(X) \geq 0$), and is minimized when X has a sole possible outcome: $H(X) = 0$ just when $\exists x \in X,\ P(x) = 1$.

These properties provide some justification for thinking of entropy as a measure of uncertainty in random variables, since it is minimized when the outcome is deterministic, and grows larger as the distribution becomes more even-handed.

The definition of entropy is also easily extended to *joint distributions*; given the pair of variables X, Y, for instance, the entropy of their joint occurrences, $H(X, Y)$ is given by simply replacing $P(x)$ with $P(x, y)$ throughout formula (5.2). More interestingly, we can also define a *conditional* notion of entropy, based on the uncertainty of the conditional distribution over two variables.

Definition 5.3 (Conditional Entropy)**.** For random variables X and Y, the *conditional entropy of X given Y* is the average, over $y \in Y$, of the entropy of $P(X \mid y)$:

$$\begin{aligned} H(X \mid Y) &\equiv \sum_{y \in Y} P(y) \sum_{x \in X} P(x \mid y) \log \frac{1}{P(x \mid y)} \\ &= \sum_{x \in X,\, y \in Y} P(x, y) \log \frac{1}{P(x \mid y)}. \end{aligned} \tag{5.3}$$

We can now prove some basic properties of conditional entropy. The first establishes that conditioning always *reduces* entropy, so that the conditional entropy of X given Y is bounded above by the entropy of X alone. The second shows that this upper bound is reached precisely when X and Y are independent. Finally, we can demonstrate a useful *chain rule* for conditional entropy, relating it to other quantities.

Proposition 5.1. Conditioning never increases entropy: $H(X \mid Y) \leq H(X)$.

Proof. The result relies upon two facts, not proven here, but well-established:

1. *Jensen's Inequality* [65]: for a real convex function f and random variable x, $\mathcal{E}[f(x)] \geq f(\mathcal{E}[x])$, where $\mathcal{E}[z]$ is the expectation of z.

2. The function $\log 1/x = -\log x$ is convex.

With these facts in hand, we reason as follows:

$$H(X \mid Y) - H(X) = \sum_{x \in X,\, y \in Y} P(x, y) \log \frac{1}{P(x \mid y)} - \sum_{x \in X} P(x) \log \frac{1}{P(x)}$$

That is, by marginalization and the definition of conditional probability:

$$= \sum_{x \in X,\, y \in Y} P(x,y) \log \frac{P(y)}{P(x,y)} + \log P(x)$$
$$= \sum_{x \in X,\, y \in Y} P(x,y) \log \frac{P(x)P(y)}{P(x,y)}$$

And then, by Jensen's Inequality:

$$\leq \log \sum_{x \in X,\, y \in Y} P(x,y) \frac{P(x)P(y)}{P(x,y)}$$
$$= \log \sum_{x \in X,\, y \in Y} P(x)P(y) = \log 1 = 0.$$

Thus $H(X \,|\, Y) - H(X) \leq 0$ and so $H(X \,|\, Y) \leq H(X)$. □

Proposition 5.2. If variables X and Y are independent, then the conditional entropy of X given Y is simply the entropy of X: $H(X \,|\, Y) = H(X)$.

Proof. Suppose variables X and Y are independent of one another. Then, for any values $x \in X$ and $y \in Y$, $P(x,y) = P(x)P(y)$ and $P(x \,|\, y) = P(x)$, and we have:

$$\begin{aligned} H(X \,|\, Y) &= \sum_{x \in X,\, y \in Y} P(x,y) \log \frac{1}{P(x \,|\, y)} \\ &= \sum_{x \in X,\, y \in Y} P(x)P(y) \log \frac{1}{P(x)} \\ &= \sum_{y \in Y} P(y) \sum_{x \in X} P(x) \log \frac{1}{P(x)} \\ &= 1 \times \sum_{x \in X} P(x) \log \frac{1}{P(x)} = H(X). \end{aligned}$$

Thus, for independent variables X and Y, the uncertainty of X given Y is just the original uncertainty of X itself. □

Proposition 5.3. For any variables X and Y, the conditional entropy of X given Y is just their joint entropy, minus the entropy of Y; that is, conditional entropy satisfies the *chain rule*: $H(X \mid Y) = H(X, Y) - H(Y)$.

Proof. From the definition of conditional entropy of X given Y, we have:

$$\begin{aligned} H(X \mid Y) &= \sum_{x \in X,\, y \in Y} P(x, y) \log \frac{1}{P(x \mid y)} \\ &= \sum_{x \in X,\, y \in Y} P(x, y) \log \frac{P(y)}{P(x, y)} \\ &= \sum_{x \in X,\, y \in Y} P(x, y)(\log P(y) - logP(x, y)) \\ &= \sum_{x \in X,\, y \in Y} P(x, y) \log P(y) - \sum_{x \in X,\, y \in Y} P(x, y) \log P(x, y) \end{aligned}$$

Thus, by marginalization:

$$= \sum_{y \in Y} P(y) \log P(y) - \sum_{x \in X,\, y \in Y} P(x, y) \log P(x, y)$$

And so, since $\log 1/x = -\log x$:

$$\begin{aligned} &= \sum_{x \in X,\, y \in Y} P(x, y) \log \frac{1}{P(x, y)} - \sum_{y \in Y} P(y) \log \frac{1}{P(y)} \\ &= H(X, Y) - H(Y). \end{aligned}$$

Thus, the conditional entropy of X given Y is just the residual remaining when subtracting Y's entropy from the joint entropy of the two variables together. □

Finally, since we can measure how much the entropy of one variable is affected by the values of another, we can think about how much uncertainty about the first is *reduced*, given the second. This provides us with a basic means of measuring the amount of information that one variable carries about another:

Definition 5.4 (Mutual Information)**.** For two random variables X and Y, the *mutual information between X and Y* is given by the difference between the entropy of X and the conditional entropy:

$$I(X;Y) \equiv H(X) - H(X \,|\, Y), \tag{5.4}$$

Proposition 5.4. Mutual Information can be written equivalently as follows:

$$I(X;Y) = \sum_{x \in X,\, y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{5.5}$$

Proof. By definition, and the chain rule for conditional entropy (Proposition 5.3):

$$\begin{aligned} I(X;Y) \equiv H(X) - H(X \,|\, Y) &= H(X) - (H(X,Y) - H(Y)) \\ &= \sum_{x \in X} P(x) \log \frac{1}{P(x)} - \sum_{x \in X,\, y \in Y} P(x,y) \log \frac{1}{P(x,y)} + \sum_{y \in Y} P(y) \log \frac{1}{P(y)} \end{aligned}$$

By marginalization:

$$\begin{aligned} &= \sum_{x \in X,\, y \in Y} P(x,y) (\log \frac{1}{P(x)} + \log \frac{1}{P(y)} - \log \frac{1}{P(x,y)}) \\ &= \sum_{x \in X,\, y \in Y} P(x,y) \frac{P(x,y)}{P(x)P(y)}. \end{aligned}$$

Thus, the two forms given for $I(X;Y)$, equations (5.4) and (5.5), are equivalent. □

Mutual Information has three elementary and noteworthy properties:

- It is *symmetrical*: $I(X;Y) = I(Y;X)$. This follows straightforwardly from the form given in equation (5.5), and the elementary fact $P(x,y) = P(y,x)$.
- It is bounded below by 0: $I(X;Y) \geq 0$. This follows from the form given in equation (5.4), and the fact that $H(X \,|\, Y) \leq H(X)$ (Proposition 5.1).

- It is precisely 0 when variables X and Y are independent. This follows from equation (5.5): if X and Y are independent, then $P(x, y) = P(x)P(y)$ and so the log term is $\log 1 = 0$ at every point in the sum.

We can now use this idea of mutual information to deal with the dynamics of Dec-POMDPs, treating a given agent's actions, and various elements of the problem domain that can change in response, as if they were random variables. The mutual information between the actions and the various outcomes will then provide an easily calculated measure of how much influence the agent has on the system.

5.3.2 Agent Influence and Influence Gap

In a Dec-POMDP, an agent's actions can have effects on any or all of the three main components of the system: state-transitions, single-step rewards, and observations. We separate those parts of the problem, giving three corresponding *influence measures*, and then combine them into an overall quantification of an agent's interaction with the problem dynamics.

In the definitions that follow, we define our measures for each individual agent, relative to actions taken from its own action-set. For an n-agent Dec-POMDP, and one such agent α_i with action-set A_i, let $\overline{\alpha}_{-i}$ be the set of all agents *except* α_i, and let $\overline{A}_{-i}$ be the set of joint actions that can be taken by those other agents (so $\overline{A}_{-i} = \{\langle a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\rangle \mid a_m \in A_m\}$).

Definition 5.5 (State-influence)**.** For any agent α_i in a Dec-POMDP, $\mathcal{D}$, the *state-influence* of α_i, SI_i, is the mutual information between that agent's actions (taken as a random variable), and the outcome state variable.

$$\begin{aligned} SI_i \equiv I(S; A_i) &= \sum_{s \in S} \sum_{a_i \in A_i} p(s, a_i) \log \frac{p(s, a_i)}{p(s)\, p(a_i)} \\ &= \sum_{s \in S} \sum_{a_i \in A_i} p(s \mid a_i)\, p(a_i) \log \frac{p(s \mid a_i)}{p(s)} \end{aligned} \tag{5.6}$$

where we marginalize to get:

$$p(s \mid a_i) = \sum_{s' \in S} \sum_{\overline{a}_j \in \overline{A}_{-i}} P(s \mid s', a_i, \overline{a}_j)\, p(s')\, p(a_i)\, p(\overline{a}_j) \qquad [P(\cdot) \in \mathcal{D}]$$
$$p(s) = \sum_{a_i \in A_i} p(s \mid a_i)\, p(a_i)$$

and we treat starting states, s', and possible actions as occurring uniformly:

$$p(s') = \frac{1}{|S|} \qquad p(a_i) = \frac{1}{|A_i|} \qquad p(\overline{a}_j) = \frac{1}{\left|\overline{A}_{-i}\right|}.$$

Definition 5.6 (Reward-influence)**.** For any agent α_i in a Dec-POMDP, $\mathcal{D}$, the *reward-influence* of α_i, RI_i, is given by the mutual information between that agent's actions (taken as a random variable), and the outcome reward in a single time-step. (We will abuse notation somewhat and write $r \in R$ for the range of the reward-function, R, i.e., the various distinct reward-values possible in $\mathcal{D}$.)

$$\begin{aligned} RI_i \equiv I(R; A_i) &= \sum_{r \in R} \sum_{a_i \in A_i} p(r, a_i)\, \log \frac{p(r, a_i)}{p(r)\, p(a_i)} \\ &= \sum_{r \in R} \sum_{a_i \in A_i} p(r \mid a_i)\, p(a_i)\, \log \frac{p(r \mid a_i)}{p(r)} \end{aligned} \tag{5.7}$$

where we marginalize to get:

$$p(r \mid a_i) = \sum_{s \in S} \sum_{\overline{a}_j \in \overline{A}_{-i}} p(r \mid s, a_i, a_j)\, p(s)\, p(a_i)\, p(\overline{a}_j)$$
$$p(r) = \sum_{a_i \in A_i} p(r \mid a_i)\, p(a_i)$$

we use the definition:

$$p(r \mid s, a_i, a_j) = \begin{cases} 1 & \text{if } R(s, a_i, a_j) = r \text{ for } R(\cdot) \in \mathcal{D}; \\ 0 & \text{otherwise.} \end{cases}$$

and we treat states and actions as occurring uniformly:

$$p(s) = \frac{1}{|S|} \qquad p(a_i) = \frac{1}{|A_i|} \qquad p(\overline{a}_j) = \frac{1}{\left|\overline{A}_{-i}\right|}.$$

Definition 5.7 (Observation-influence)**.** For any agent α_i in a Dec-POMDP, $\mathcal{D}$, the *observation-influence* of α_i, OI_i, is given by the mutual information between that agent's actions (taken as a random variable), and the resulting joint observation, $\overline{o} \in \overline{\Omega} = \Omega_0 \times \cdots \times \Omega_n$:

$$\begin{aligned} OI_i \equiv I(\overline{\Omega}; A_i) &= \sum_{\overline{o} \in \overline{\Omega}} \sum_{a_i \in A_i} p(\overline{o}, a_i) \, \log \frac{p(\overline{o}, a_i)}{p(\overline{o})\, p(a_i)} \\ &= \sum_{\overline{o} \in \overline{\Omega}} \sum_{a_i \in A_i} p(\overline{o} \mid a_i) p(a_i) \log \frac{p(\overline{o} \mid a_i)}{p(\overline{o})} \end{aligned} \tag{5.8}$$

where we marginalize to get:

$$\begin{aligned} p(\overline{o} \mid a_i) &= \sum_{s' \in S} \sum_{\overline{a}_j \in \overline{A}_{-i}} O(\overline{o} \mid a_i, \overline{a}_j, s')\, p(s' \mid a_i, a_j)\, p(a_i)\, p(\overline{a}_j) && [O(\cdot)) \in \mathcal{D}] \\ p(s' \mid a_i, \overline{a}_j) &= \sum_{s \in S} P(s' \mid s, a_i, \overline{a}_j)\, p(s)\, p(a_i)\, p(\overline{a}_j) && [P(\cdot) \in \mathcal{D}] \\ p(\overline{o}) &= \sum_{a_i \in A_i} p(\overline{o} \mid a_i)\, p(a_i) \end{aligned}$$

and we treat prior states and actions as occurring uniformly:

$$p(s) = \frac{1}{|S|} \qquad p(a_i) = \frac{1}{|A_i|} \qquad p(\overline{a}_j) = \frac{1}{\left|\overline{A}_{-i}\right|}.$$

These three values thus give us a measure of how much information is shared between an agent's actions, on the one hand, and the values of the state-transition, reward, and observation functions that specify the Dec-POMDP, on the other. As we noted in the previous section, mutual information is always non-negative, and so $SI_i \geq 0$, $RI_i \geq 0$, and $OI_i \geq 0$, for any agent α_i. When in fact $SI_i = 0$, then for all intents and purposes agent α_i has no effect upon the state-transition table; given that agent's action in any system state, the outcome state is either purely random (if it is also the case that $S_j = 0$ for all α_j), or is influenced only by the other agents in $\overline{\alpha}_{-i}$. A similar point holds for RI_i and OI_i. (We make this more precise in the next section, and discuss upper bounds on the measures.)

We can thus look at the sum total of these three possible influences as measuring the combined influence that agent α_i can have on the system dynamics of a Dec-POMDP. The difference between these degrees of influence then measures how much more or less one agent dominates the dynamics of the system than another. The overall influence gap for a Dec-POMDP is then a function of individual agent gaps.

Definition 5.8 (Influence and Influence Gap)**.** For any agent, α_i, in a Dec-POMDP, $\mathcal{D}$, the *influence of* α_i, $\mathcal{I}(\alpha_i)$, is the sum of α_i's various influence measures:

$$\mathcal{I}(\alpha_i) = SI_i + RI_i + OI_i. \tag{5.9}$$

For any pair of agents, α_i and α_j, in a Dec-POMDP, $\mathcal{D}$, the *influence gap between* α_i *and* α_j, $IG(\alpha_i, \alpha_j)$, is the absolute difference between their influence:

$$IG(\alpha_i, \alpha_j) \equiv \left|\mathcal{I}(\alpha_i) - \mathcal{I}(\alpha_j)\right|. \tag{5.10}$$

The overall *influence gap for* $\mathcal{D}$ is the pairwise minimum of inter-agent gaps:

$$IG(\mathcal{D}) \equiv \min_{\alpha_i, \alpha_j \in \mathcal{D}} IG(\alpha_i, \alpha_j). \tag{5.11}$$

A word about the last part of the definition, where we give the gap for a problem in terms of the minimum of the pairwise gaps: as we shall show later on in our empirical study of two-agent problems, the difficulty of solving a problem tends to correlate inversely with the size of the gap between the two. That is, as the influence gap between the two agents shrinks to 0 (and both has equal, non-zero influence), problems tend to become much harder, while as it grows away from 0 (and one agent dominates the other), they tend to become easier to solve. Since multiagent problems generally grow in difficulty with the number of agents, the use of the minimum gap thus makes most sense. Even if large gaps exist between most agents, we can expect that any pair with a very small gap between them is usually still going to have a very hard sub-problem to solve when coordinating among themselves. (If the reader finds this line of reasoning unpersuasive, it should be noted that the definition can be seen as simply one of convenience.)

5.4 Properties of Influence

We now establish a few basic properties of agent influence. One thing to note, right off the bat, is that all our measures are straightforward to compute. Evidently, no complex mathematics is involved: all operations are either simple arithmetic, or involve taking the log. Further, for each agent, α_i, the number of operations required to calculate total influence $\mathcal{I}(\alpha_i)$ is

$$\mathcal{O}(|A_i| \times (|S| + |R| + \left|\overline{\Omega}\right|),$$

and so the calculation of total influence for all agents, and pairwise comparison between them, is clearly

$$\mathcal{O}([|\{A_i\}| \times (|S| + |R| + \left|\overline{\Omega}\right|)] + |\{\alpha_i\}|^2).$$

Not only is this polynomial in the size of the Dec-POMDP, $\mathcal{D}$, but it is worth noting that it will generally be somewhat less, since a full problem specification can be quite complex to detail completely, given the need to specify joint action transitions, rewards, joint observations, and so forth.

Beyond this basic fact, we can demonstrate some other properties of the influence measures. Bounds can be placed on their ranges, and we can demonstrate the necessity of taking all three distinct forms of agent influence—over state-transitions, rewards, and observations—into account when planning in a decentralized setting. Furthermore, we can show how agents without influence over parts of the system dynamics can be ignored in the planning process in certain cases, simplifying the resulting problem.

5.4.1 Bounding the Influence Measures

As we have discussed, each individual measure of influence is non-negative, by the very nature of mutual information. The upper bounds on the influence measures will vary from problem to problem, depending upon the number of distinct actions, states, reward-values, or shared observations:

Theorem 5.1. Each influence measure is bounded as follows:

1. $0 \leq SI_i \leq \log(|S| \times |A_i|)$.
2. $0 \leq RI_i \leq \log(|R| \times |A_i|)$ (where $|R|$ is, again, the number of possible reward-values in the range of reward-function, R).
3. $0 \leq OI_i \leq \log(\left|\overline{\Omega}\right| \times |A_i|)$.

Proof. For SI_i, we can reason as follows. The lower bound is given by the basic definition of mutual information, as already mentioned. Also by definition, and by the chain rule for conditional entropy (Proposition 5.3),

$$
\begin{aligned}
SI_i \equiv I(S; A_i) &= H(S) - H(S \mid A_i) \\
&= H(S) - (H(S, A_i) - H(A_i)) \\
&= H(S) + H(A_i) - H(S, A_i) \\
&\leq H(S) + H(A_i) \\
&\leq \log |S| + \log |A_i| = \log(|S| \times |A_i|),
\end{aligned}
$$

where the last two lines follow from the elementary bounds on entropy, namely that $0 \leq H(Z) \leq \log |Z|$, for any variable (or joint combination) Z. The same proof can be applied to establish the bounds on RI_i and OI_i. □

Just as we provided a simple bound on each individual influence measure, we can also put elementary limits on an agent's total influence, and on the overall influence gap for a problem.

Theorem 5.2. For any agent, α_i, and Dec-POMDP, $\mathcal{D}$:

1. $0 \leq \mathcal{I}(\alpha_i) \leq \log(|S| \times |R| \times \left|\overline{\Omega}\right| \times |A_i|^3)$.
2. $0 \leq IG(\mathcal{D}) \leq \max_{\alpha_i} \mathcal{I}(\alpha_i)$.

Proof. For the first part, we know from Theorem 5.1 that:

$$
\begin{aligned}
0 \leq \mathcal{I}(\alpha_i) &= (SI_i + RI_i + OI_i) \\
&\leq \log(|S| \times |A_i|) + \log(|R| \times |A_i|) + \log(\left|\overline{\Omega}\right| \times |A_i|) \\
&= \log(|S| \times |R| \times \left|\overline{\Omega}\right| \times |A_i|^3).
\end{aligned}
$$

The second part of the claim follows by definition of $IG(\mathcal{D})$, since the pairwise absolute difference between agent-influences is bounded above by the largest of the pair, and below by zero. □

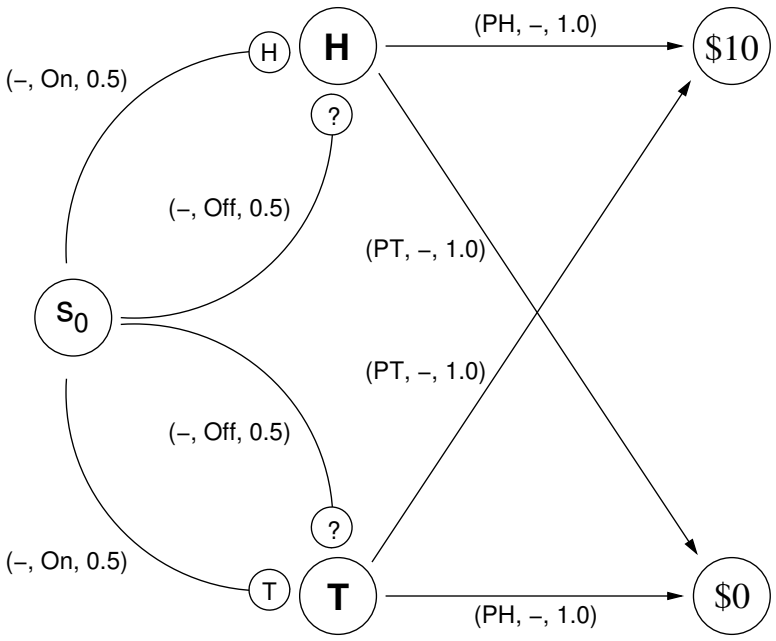


Figure 5.1: A sample problem domain where one agent only has observation-influence. States are given in circles, with the rewards for the final outcomes ($0 and $10), labeling those states. Arrows denote action transitions, and are labeled with pairs of actions and associated probabilities; a dash, '–', means that the action-transition is indifferent to any choice made by the agent in question. The small circles at the ends of the first set of arrows contain the associated observations for the state-action transition: either the state of the coin (H or T), or nothing at all ($?$).

5.4.2 Necessity of the Influence Measures

We have separated our conception of influence into three distinct measures, governing agent impact on state-transitions, single-step rewards, and observations. This is not merely for ease of definition. Rather, each element is defined separately due to the simple fact that an agent may have absolutely no influence on one or more parts of the system dynamics, and yet be key to the solution of a given Dec-POMDP via its effect on another. While it may seem at first as if an agent that, for instance, has no effect at all upon rewards received at any stage cannot be involved in any real way in the optimal policy, this is not the case. Via influence on joint state-transitions or on

the information other agents receive via observations, such an agent could still have all the effect in the world, making a difference to any final outcome.

As an illustration of this point, we present the problem given in Figure 5.1. The problem involves flipping a coin in a darkened room. There are two agents involved. Before the coin is flipped, in initial state, s_0, one agent chooses whether or not to turn on the lights in the room (actions *Off* and *On* in the diagram). The coin is then flipped, and lands Heads or Tails with equal probability, no matter which actions are chosen by either agent; however, if the lights have been turned on, both agents observe the coin's outcome, otherwise observing nothing at all. After the flip, whatever the observation, the other agent must pick heads (PH) or tails (PT). If it chooses correctly, the state transitions to a goal-state carrying a joint non-zero reward ($10), and if it chooses incorrectly, no reward is received.

It is easy to see that the agent choosing the light-switch position has no state-influence, and no reward-influence. In the first state-action transition, the state given by the coin-flip transitions randomly, and is completely independent of any action taken by either agent. In the second transition, to the two possible outcome states, the result is decided entirely by the state of the coin and the action of the other agent, who picks either Heads or Tails. Thus, all state-transitions are completely independent of the light-switching agent, and state-influence will be zero (0). Furthermore, the only reward to be gained is at the final stage in the process, and again, since this transition is independent of the actions of the light-switching agent, reward-influence will also be zero (0).

At the same time, however, the action-choice of that agent has positive influence on the observations. When that agent chooses to turn on the light, that action results in both agents observing the state of the coin with certainty; if the light is not turned on, then nothing is observed, also with certainty. Thus we have the possible joint observations $\overline{o} \in \overline{\Omega} = (\Omega_0 \times \Omega_1)$: $\langle H, H\rangle$, $\langle T, T\rangle$, and $\langle ?, ?\rangle$. And in accord with

Definition 5.7, we treat all states and agent as if they were uniformly distributed, so $p(s) = 0.2$ for any of the 5 states in the problem, and the agents two actions each are assumed to occur with probability $p(a) = 0.5$. We can then calculate the probability of moving to the Heads state, H, given the agent's choice of the On action (ignoring the other agent's actions, since the transition is indifferent, and they marginalize out):

$$p(H \mid On) = \sum_{s \in S} P(H \mid s,\ On) p(s) p(On),$$

which, since the leading transition-probability is 0 for all states but the intial one, is:

$$\begin{aligned} &= P(H \mid s_0,\ On) p(s_0) p(On) \\ &= 0.5 \times 0.2 \times 0.5 = 0.05. \end{aligned}$$

And similarly, we have the other important state-action probabilities, given by: $p(H \mid Off) = 0.05$, $p(T \mid On) = 0.05$, and $p(T \mid Off) = 0.05$.

Furthermore, we can calculate the probability of observation $\langle H, H \rangle$, conditional on the agent choosing the On action:

$$p(\langle H, H \rangle \mid On) = \sum_{s \in S} O(\langle H, H \rangle \mid On,\ s) p(s \mid On) p(On),$$

where the leading observation-probability will again be 0 for all states but *Heads*:

$$\begin{aligned} &= O(\langle H, H \rangle \mid On,\ H) p(H \mid On) p(On) \\ &= 1.0 \times 0.05.5 = 0.025. \end{aligned}$$

Additionally, we have the other values: $p(\langle H, H \rangle \mid Off) = 0$, $p(\langle T, T \rangle \mid On) = 0.025$, $p(\langle T, T \rangle \mid Off) = 0$, $p(\langle ?, ? \rangle \mid On) = 0$, and $p(\langle ?, ? \rangle \mid Off) = 0.025$.

Finally, the marginal probability of every observation-pair is easily shown to be identical ($p(\langle H, H\rangle) = p(\langle T, T\rangle) = p(\langle ?, ?\rangle) = 0.0125$), and we can calculate the observation-influence for the agent as follows:

$$OI \equiv I(\overline{\Omega}; A) = \sum_{\overline{o} \in \overline{\Omega}} \sum_{a \in A} p(\overline{o} \,|\, a) p(a) \log \frac{p(\overline{o} \,|\, a)}{p(\overline{o})},$$

which, given the above values, is expressed in bits as:

$$\begin{aligned} &= 4 \times (0.025 \times 0.5 \times \log_2 \frac{0.025}{0.0125}) \\ &= 4 \times (0.0125 \times \log_2 2) = 4 \times (0.0125 \times 1) = 0.05 \text{ bits.} \end{aligned}$$

Thus, the light-switching agent has non-zero observation-influence. (In fact, since its state- and reward-influence are both zero, this is its total influence, as well.) It is also easy to see that this agent is key to the expected value of any policy. For any policy where the agent turns the light on in initial state s_0, before the coin is flipped, the other agent will receive its observation of the state of the coin, and the optimal action-choice will guarantee the maximum reward, for an expectation of the full \$10. If the light is not turned on, then any choice by the second agent is indifferent, since the same observation ($\langle ?, ?\rangle$) is received with certainty, whatever the coin-flip outcome, and any such policy has expected value of only $(0.5 \times \$10) = \$5$. Although all its influence is limited to observations, the importance of the agent is clear.[1]

Similar examples can be drawn up for state- and reward-influence, so that some agent's overall effect is limited to just one aspect of the dynamics, but is still key to the outcome. For this reason, we have chosen to divide the influence-measures

[1]It is worth noting that the example also demonstrates the inverse effect: the agent who chooses Heads or Tails has *no observation-influence at all*, but is also key to the expected value of any policy, solely on the basis of its influence over state-transitions and single-step rewards.

into three parts. Doing so allows us to isolate the particular ways in which actions may actually affect the system, and it is only by looking at the total across all three dimensions of influence that we can be certain whether a given agent is important to solving the problem or not.

5.4.3 Insufficiency of the Influence Measures

In order to demonstrate the necessity of paying attention to each of our three influence measures, we have just shown how an agent can affect expected value with only a single form of positive influence on the system. It should now be noted that the converse is not true. That is, simply because an agent *does have* some positive influence on the system, this *does not* mean that it will have any effect on the expected value of policies.

This is easy to see. Turning again to the sample problem given in Figure 5.1, consider what would happen if we changed the rewards, so that the agents received \$10 in both final outcome states, but all else remained the same. In that case, the light-switching agent would still have precisely the same observation-influence on the system, but now would have absolutely no effect on expected value, since all policies whatsoever would have the same payoff in the end. Thus, when planning, such an agent could in theory be completely ignored: any course of action by the other agents will be indifferent to its behavior. (Note that it is not key to this point that all outcomes be the same, so that any policy whatsoever has the same value; it is just as easy to construct examples where an agent has positive influence, but zero impact on value, and yet all other agents are still faced with a challenging planning problem, with many distinctly different outcomes.)

In some sense, this sort of outcome is unavoidable, and we argue that it should not be thought of as a defect in the notion of influence. There will indeed be cases in which an agent will have influence on system dynamics, but will not play any

role in determining the expected value of a policy (or partial sub-policy). However, this is really no different than noting, after having solved a Dec-POMDP, that some possible actions play no role in deciding the outcome of any policy. While it may be easy to see that apparent influence is actually unimportant in the simple example just considered, this will not be readily apparent in other cases. Short of evaluating the expected value of every course of action—that is, short of solving the problem in full—we will not be able to eliminate the possibility that the actions of an agent that possesses some measure of influence are actually of importance.

5.4.4 Zero-Influence Agents in Multiagent Problems

Since it is possible for an agent to possess some influence, but be irrelevant to outcomes, positive influence cannot be used to identify agents that are of real importance when planning. On the other hand, our influence measures do allow us to rule out agents that are definitely *unimportant*.

Theorem 5.3. In a Dec-POMDP $\mathcal{D}$ with n agents, suppose some agent α_i has no influence at all: $\mathcal{I}(\alpha_i) = (SI_i + RI_i + OI_i) = 0$. In this case, the remaining agents can ignore α_i, since the value of any joint policy is indifferent to the particular policy π_i. That is, for any individual policies $\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n$ of the other agents, and some policy π_i of α_i, let $\pi^+(\pi_i)$ be the joint policy, $\langle \pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n \rangle$ (assuming all individual policies to be of some same length, k); we then have that:

$$(\forall \pi_i, \pi_i')\,(\forall s \in S)\, V^{\pi^+(\pi_i)}(s) = V^{\pi^+(\pi_i')}(s) \tag{5.12}$$

Proof. For convenience, we prove the claim for finite policies, although essentially the same technique could be used for the infinite-horizon case. Our proof will establish the converse claim: if some pair of policies for α_i lead to different values, then $\mathcal{I}(\alpha_i) \gneq 0$.

We begin by assuming that there exists policies π_i, π_i', and some state $s \in S$, such that policy-values differ: $V^{\pi^+(\pi_i)}(s) \neq V^{\pi^+(\pi_i')}(s)$. In that case, by the definition of

policy-value (Definition 2.5, page 27), there exists some tuple of observation-sequences $\langle \overline{o}_1, \ldots, \overline{o}_n \rangle$ and state $s' \in S$ such that:

$$P^{\pi^+(\pi_i)}(s, \overline{o}_1, \ldots, \overline{o}_n, s') \times R(s', \pi_1(\overline{o}_1), \ldots, \pi_{i-1}(\overline{o}_{i-1}), \pi_i(\overline{o}_i), \pi_{i+1}(\overline{o}_{i+1}), \ldots, \pi_n(\overline{o}_n))$$

has a different value than:

$$P^{\pi^+(\pi'_i)}(s, \overline{o}_1, \ldots, \overline{o}_n, s') \times R(s', \pi_1(\overline{o}_1), \ldots, \pi_{i-1}(\overline{o}_{i-1}), \pi'_i(\overline{o}_i), \pi_{i+1}(\overline{o}_{i+1}), \ldots, \pi_n(\overline{o}_n))$$

There are then two sub-cases to consider:

Case 1. The probabilities are different:

$$P^{\pi^+(\pi_i)}(s, \overline{o}_1, \ldots, \overline{o}_n, s') \neq P^{\pi^+(\pi'_i)}(s, \overline{o}_1, \ldots, \overline{o}_n, s'). \tag{5.13}$$

By Definition 2.4 (page 26), the probability of transitioning from s to s', given a collection of observation-histories of the form $\overline{o}_i o_i$ (that is, a sequence for agent α_i where the most recent observation is o_i), is defined recursively for any policy π:

$$P^{\pi}(s, \overline{o}_1 o_1, \ldots, \overline{o}_n o_n, s') = \sum_{s'' \in S} P^{\pi}(s, \overline{o}_1, \ldots, \overline{o}_n, s'') \times P(s'', \pi_1(\overline{o}_1), \ldots, \pi_n(\overline{o}_n), s') \times O(\pi_1(\overline{o}_1), \ldots, \pi_n(\overline{o}_n), s', o_1, \ldots, o_n).$$

The inequality in Equation (5.13) then means that at least one of these must hold:

Sub-case a. Observation sequences $\overline{o}_1, \ldots, \overline{o}_n$, and states $s, s' \in S$ exist such that:

$$P(s, \pi_1(\overline{o}_1), \ldots, \pi_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n), s') \neq P(s, \pi_1(\overline{o}_1), \ldots, \pi'_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n), s'),$$

which simply means that

$$p(s' \mid s, \pi_1(\overline{o}_1), \ldots, \pi_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n)) \neq p(s' \mid s, \pi_1(\overline{o}_1), \ldots, \pi'_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n)).$$

However, since $\pi_i(\overline{o}_i)$, $\pi'_i(\overline{o}_i) \in A_i$, we now have that the distribution over states is non-independent of actions for α_i: $p(S \mid A_i) \neq p(S)$. It is a basic fact about Mutual Information between two variables X and Y, however, that it is 0 if and only if X and Y are independent (see page 129). Thus, the state-influence (Definition 5.5) of α_i must be strictly positive, $SI_i \gneq 0$.

Sub-case b. State $s' \in S$, and observations $o_1, \ldots, o_n$ exist such that:

$$O(\pi_1(\overline{o}_1), \ldots, \pi_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n),\ s',\ o_1, \ldots,\ o_n), \neq$$
$$O(\pi_1(\overline{o}_1), \ldots, \pi'_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n),\ s',\ o_1, \ldots,\ o_n),$$

which means that

$$p(o_1, \ldots,\ o_n \mid \pi_1(\overline{o}_1), \ldots, \pi_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n),\ s'), \neq$$
$$p(o_1, \ldots,\ o_n \mid \pi_1(\overline{o}_1), \ldots, \pi'_i(\overline{o}_i), \ldots, \pi_n(\overline{o}_n),\ s').$$

Again, this implies that the distribution over joint observations is non-independent of the actions of α_i, $p(\overline{\Omega} \mid A_i) \neq p(\overline{\Omega})$, and the observation-influence (Definition 5.7) of α_i is strictly positive, $OI_i \gneq 0$.

Case 2. The one-step rewards are distinct:

$$R(s',\ \pi_1(\overline{o}_1), \ldots,\ \pi_{i-1}(\overline{o}_{i-1}),\ \pi_i(\overline{o}_i),\ \pi_{i+1}(\overline{o}_{i+1}), \ldots,\ \pi_n(\overline{o}_n)) \neq$$
$$R(s',\ \pi_1(\overline{o}_1), \ldots,\ \pi_{i-1}(\overline{o}_{i-1}),\ \pi'_i(\overline{o}_i),\ \pi_{i+1}(\overline{o}_{i+1}), \ldots,\ \pi_n(\overline{o}_n)).$$

In this case, we note that in the defining reward-influence (Definition 5.6), we defined the probability distribution over rewards using the convention:

$$p(r \mid s, a_1, , \ldots, a_n) = \begin{cases} 1 & \text{if } R(s, a_1, \ldots, a_n) = r \text{ for } R(\cdot) \in \mathcal{D}; \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the probability distribution over rewards is not independent of the actions of α_i, $P(R \mid A_i) \neq P(R)$, and so reward-influence is strictly positive, $RI_i \gneq 0$.

Therefore, since each possibility means that one of the influence quantities is strictly positive, we have that α_i has positive influence, $\mathcal{I}(\alpha_i) = (SI_i + RI_i + OI_i) \gneq 0$, as required. □

In effect, then, any n-agent Dec-POMDP for which some agent has no influence is in fact equivalent to an $(n-1)$-agent problem. Since any such agent has no effect on possible policy values, planning need not consider them. (As for those agents themselves, a random policy is automatically optimal.) While this is only the extreme case, it is hard to do better, at least when planning optimally. As we have discussed, it is always possible that an agent with some measured influence yet has no effect on policy values. However, it is very hard to identify such cases in general, short of evaluating all policies.

5.4.5 Influence in Factored Dec-POMDPs

In Section 3.1, we presented the model of a *factored* Dec-POMDP, where the state-space divides into non-overlapping local state-spaces for each agent: $S = S_0 \times S_1 \times \cdots \times S_n$, and for each α_i, the local space is $\hat{S}_i = S_0 \times S_i$. As it turns out, the current definitions of influence are somewhat inadequate to these sorts of problems. In particular, there will be cases in which the separate sub-problems for the agents are entirely disjoint, and yet the influence measure will still be positive.

A simple thought-experiment establishes the point. Assume that we have two wholly separate MDPs, $\mathcal{M}^1$ and $\mathcal{M}^2$, such that agent $\alpha_1 \in \mathcal{M}^1$ has some positive influence $\mathcal{I}(\alpha_1) \gneq 0$ in its problem, and similarly $\mathcal{I}(\alpha_2) \gneq 0$ in $\mathcal{M}^2$. Now form a factored Dec-MDP $\mathcal{D}$, by simply combining the two state-spaces, so that $S^1 \times S^2 = S \in \mathcal{D}$ (with $S_0 = \emptyset$), and let the system dynamics remain defined over each agent's local states. This will be a problem, obviously, that is fully independent, in terms of state-transitions, observations, and rewards. Thus, equally obviously, the agents can plan in their own state-space without paying any attention at all to the other agent, and yet, by definition, the influence measures will be identical (that is, strictly positive) in the new problem. Indeed, if we set up the problems so that the agent's originally had equal influence, $\mathcal{I}(\alpha_1) = \mathcal{I}(\alpha_2)$, the influence gap would be 0, something that we have argued is suggestive of the very hardest decentralized problems.

Thus, the presence of independence relations in a problem can mean that agents can in fact plan and work separately, even if they both have positive influence. To amend this, we extend the definitions of influence to factored state-spaces. In each case, the idea is once again to measure how much influence the agent has on the dynamics of the state-space, but to restrict the measurement to the local state-spaces of other agents. As an example, we define the *factored state-influence* of agent α_i as the mutual information between its actions and the state-space *apart from* S_i.

Definition 5.9 (Factored State-Influence)**.** For any agent α_i in a Dec-POMDP $\mathcal{D}$ with factored state-space $S = S_0 \times S_1 \times \cdots \times S_n$, let $\overline{S}_{-i}$ be all parts of the state-space that *do not* belong to that agent:

$$\overline{S}_{-i} = S_0 \times \cdots \times S_{i-1} \times S_{i+1} \times \cdots S_n.$$

The *factored state-influence* of α_i, FSI_i, is the mutual information between that agent's actions (taken as a random variable), and state variables in this new set.

$$FSI_i \equiv I(\overline{S}_{-i}; A_i) = \sum_{\overline{s}_j \in \overline{S}_{-i}} \sum_{a_i \in A_i} p(\overline{s}_j, a_i) \log \frac{p(\overline{s}_j, a_i)}{p(\overline{s}_j)\, p(a_i)}$$
$$= \sum_{\overline{s}_j \in \overline{S}_{-i}} \sum_{a_i \in A_i} p(\overline{s}_j \mid a_i)\, p(a_i) \log \frac{p(\overline{s}_j \mid a_i)}{p(\overline{s}_j)} \tag{5.14}$$

where we marginalize to get:

$$p(\overline{s}_j \mid a_i) = \sum_{s_i, s_i' \in S_i} \sum_{\overline{s}_j' \in \overline{S}_{-i}} \sum_{\overline{a}_j \in \overline{A}_{-i}} P(\overline{s}_j, s_i \mid \overline{s}_j', s_i', a_i, \overline{a}_j)\, p(\overline{s}_j', s_i')\, p(a_i)\, p(\overline{a}_j) \quad [P(\cdot) \in \mathcal{D}]$$
$$p(\overline{s}_j) = \sum_{a_i \in A_i} p(\overline{s}_j \mid a_i)\, p(a_i)$$

and we treat starting states, $(\overline{s}_j', s_i')$, and possible actions as occurring uniformly:

$$p(\overline{s}_j', s_i') = \frac{1}{|S|} \qquad p(a_i) = \frac{1}{|A_i|} \qquad p(\overline{a}_j) = \frac{1}{\left|\overline{A}_{-i}\right|}.$$

This is a straightforward generalization of the original (Definition 5.5); similarly we can define factored versions of reward- and observation-influence, FRI_i and FOI_i (we omit the repetition here). Given these, we can once again talk about agent influence in a factored problem.

Definition 5.10 (Factored Influence)**.** For any agent, α_i, in an n-agent Dec-POMDP $\mathcal{D}$ with factored state-space $S = S_0 \times S_1 \times \cdots \times S_n$, the *factored influence of* α_i, $\mathcal{FI}(\alpha_i)$, is the sum of its factored influence measures:

$$\mathcal{FI}(\alpha_i) = FSI_i + FRI_i + FOI_i. \tag{5.15}$$

Having done so, we note one property of these new measures, that is the factored analogue of Theorem 5.3.

Theorem 5.4. In a n-agent Dec-POMDP $\mathcal{D}$ with factored state-space $S = S_0 \times S_1 \times \cdots \times S_n$, agent α_i can be ignored by the remaining agents—in the sense that the value of any joint policy is indifferent to the particular policy π_i—if any of the following hold:

1. Agent α_i has no factored influence: $\mathcal{FI}(\alpha_i) = 0$.
2. Problem $\mathcal{D}$ is *reward-independent* and α_i has no factored influence on state-transitions or observations: $FSI_i = FOI_i = 0$.
3. Problem $\mathcal{D}$ is *transition-independent* and α_i has no factored influence on rewards or observations: $FRI_i = FOI_i = 0$.
4. Problem $\mathcal{D}$ is *observation-independent* and α_i has no factored influence on state-transitions or rewards: $FSI_i = FRI_i = 0$.
5. Problem $\mathcal{D}$ is *reward- and transition-independent* and α_i has no factored influence on observations: $FOI_i = 0$.
6. Problem $\mathcal{D}$ is *reward- and observation-independent* and α_i has no factored influence on state-transitions: $FSI_i = 0$.
7. Problem $\mathcal{D}$ is *transition- and observation-independent* and α_i has no factored influence on rewards: $FRI_i = 0$.
8. Problem $\mathcal{D}$ is *reward-, transition-, and observation-independent.*

Since the proof of each sub-claim is very similar to the proof of Theorem 5.3, we omit it here, and simply note that it is true. While the result is somewhat tedious to state, it is yet interesting, as it shows how partially independent problem dynamics can expand the ways in which a problem can be simplified.

5.5 Conclusions and Discussion

Mutual Information is a useful concept in many areas. Many machine learning techniques based on non-parametric methods, for instance, work by identifying places

where there is maximal or minimal mutual information between certain variables, as a means of discovering noteworthy connections and divining the structure of system dynamics. Our application of the idea to planning problems is new, and it leads to some interesting properties, particularly where we can identify agents that may be excluded from consideration when we are working out our courses of action. We would argue that, once understood, it makes some intuitive sense: clearly if knowing an agent's actions would tell us little or nothing about the system dynamics, there is less risk in ignoring those actions, and less to be gained by planning optimal responses to them. Furthermore, the idea of influence gap gives us a means of precisely quantifying a *degree of centralization* in some particular problem or organizational structure.

We have been able to bound this gap, and have established precise criteria for when agents may be safely ignored, in general Dec-POMDPs, and in their factored (perhaps partially independent) versions. At the same time, we have shown that there are cases of agents with no important effect on problem dynamics, even in the presence of apparently positive influence. The measure must therefore only be an approximate guide to the essential structure of agent interactions. However, as we shall show in the next chapter, there is substantial empirical evidence that the influence gap measure we have proposed has real bearing on the question of how hard particular Dec-POMDP instances will be to solve. This returns us to the conclusions drawn from our complexity results in Chapter 3. Since many Dec-POMDPs, even with highly restrictive models of interaction, remain highly complex in the worst-case analysis, we want to examine the *average-case* complexity, or at least its empirical counterpart, the expected difficulty of solving problems in practice. As we shall go on to show, the measures just provided give us tools to do just that.

CHAPTER 6

INFLUENCE GAP AND ALGORITHM PERFORMANCE

In the previous chapter, we have isolated some important theoretical properties of our influence measures. These would be of secondary interest, however, if they only revealed that we can safely ignore those agents that have no influence whatsoever. After all, such problems are only a degenerate case, and the fact does nothing to help in solving general Dec-POMDPs. Thus, we want to say something more substantive about the relationship between influence gaps and general problem difficulty: we are interested in how our measure bears on problems in which all agents have at least some real effect on system dynamics. In this chapter, we address the issue empirically, via two main sets of experiments.

First, we look at how influence gap bears on the performance of the optimal dynamic programming method (outlined in Section 2.3). As we shall see, performance of the algorithm correlates with influence gap, in that the average empirical difficulty of problem instances decreases as the gap grows. When the gap shrinks to zero, and agents have equal influence, problems are generally among the hardest possible instances, and the ability to solve them optimally is seriously constrained. Conversely, as the gap grows, and one agent's influence dominates (but all agents still have an affect), the problems tend to grow much simpler, and the optimal method is able to solve problems across time-horizons that are much greater than ever possible before.

Our second set of results concerns the memory-bounded approximate DP algorithms, which combine ground-up policy-tree generation with top-down heuristic pruning (detailed in Section 2.4). In these methods, memory is conserved by en-

forcing hard bounds on the number of policy trees retained at each backup step. As we shall see, the choice of this bounding value is key, as the time penalty for each additional tree is considerable. Previous work has simply fixed this value, more or less arbitrarily, across all problem instances. Our work here provides a more well-grounded method for choosing this value. Specifically, we find that problems with larger influence gaps have two nice features. Fist, the approximate methods generate much more valuable policies, as compared to small-gap problems. Second, effective approximation becomes easier as the gap grows, so that we can find policies that are apparently much closer to the best available, using fewer trees. These results suggest ways in which we can tune our approximate techniques intelligently, measuring influence gap and using that value to help us choose our bounds on the resources used, in order to effectively exploit trade-offs between solution quality and computation.

6.1 Influence Gap and Optimal Dynamic Programming

Our initial experiments deal with the exact dynamic programming (DP) algorithm for general Dec-POMDPs, devised by Hansen, et al. [59], and outlined in detail in Section 2.3. Working bottom-up, DP creates sets of n-step finite-horizon policy-trees by backing up existing $(n-1)$-step trees, then employs iterated pruning to reduce the overall set (and differentiating it from the elementary method of exhaustively exploring and evaluating all possible policies). In this pruning stage, a policy for one agent is retained if and only if it provides an optimal possible response to some policy of another agent. Unfortunately, this sort of pruning does not make the method any more practically effective: as we have detailed, existing results have shown that it is simply infeasible to generate policies of more than 4–5 finite action-steps, even in problems with extremely simple structures. This is not a particular fault of optimal DP, of course; as we have repeatedly stressed, these limitations are shared by all known optimal techniques, and reflect the basic difficulty of Dec-POMDPs.

At the same time, prior empirical tests of the various optimal algorithms have not been very systematic. Part of this is no doubt due to the observed difficulty of those test-cases already considered. Since each of the standard problem instances has turned out to be very hard to solve, and since the theoretical complexity of Dec-POMDPs is so high, researchers have seen little need to move beyond a handful of standard domains. There is a risk to this approach, however: lacking detailed knowledge of the average-case complexity of Dec-POMDP instances, it is harder to make detailed comparison of algorithms. In addition, if it turns out a new algorithm performs very well over some problem domains not yet considered by researchers, it is hard to evaluate the importance of the results. Finally, it would be worth investigating whether or not there is any legitimate hope for optimal algorithms. If it turns out that no problem instances whatsoever are ever amenable to exact solution, that would be sobering, of course; however, it would also be informative, particularly because no such thing has ever been convincingly demonstrated.

Our initial experiments address this issue. The results are interesting in two ways. First, we show that there are, indeed, many problem-instances for which optimal solution techniques work very well, something not well-documented in prior research on the topic. Second, we demonstrate a correlation between empirical problem difficulty and influence gap. In particular, as the gap widens among a class of problems of the same size, the optimal DP algorithm finds them easier to solve, and is able to effectively generate longer and longer finite horizon policies. This result provides our first confirmation that influence gap tracks an important property of Dec-POMDPs.

6.1.1 Experimental Design

Because the optimal DP method has intractability issues with even small problems, our testbed was restricted to a collection of simple Dec-POMDPs. Figure 6.1 shows a simple sketch of one such instance, the *Noisy Broadcast Channel Problem*;

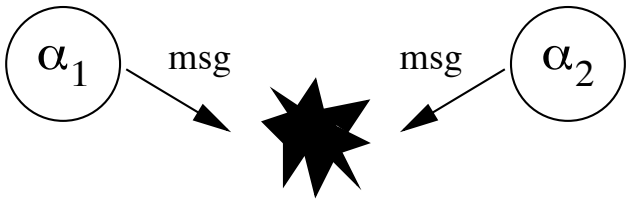


Figure 6.1: A sample of a small, 2-agent Dec-POMDP, the *Noisy Broadcast Channel Problem*. In it, agents send messages along a common channel, receiving a positive reward as long as there is no collision. Messages arrive to the agents' broadcast queues at random intervals, and no communication is possible in the attempt to coordinate channel usage at run-time.

first introduced by Ooi and Wornell [89], the broadcast domain has been converted into Dec-POMDP form and used as a standard basis for testing performance of Dec-POMDP solvers, and is one of the problems that has proved so challenging, even in its simplicity. In that problem, two agents share control of a single broadcast channel, down which they must send packets. The packets to be sent arrive randomly (with a known distribution) to each agents queue, and at each time-step an agent chooses to send a packet or not; if only one agent sends a packet, the pair receives a positive reward, but if both agents send at once, there is a collision, the packets are lost, and no reward is received. The goal, then, is to maximize sent packets over the time-horizon of the problem, minimizing collisions. While the problem is elementary to describe, generation of optimal deterministic policies is very challenging, and dynamic programming, along with other known optimal methods, ranging from heuristic search to sophisticated techniques based on mixed-integer linear programming, fail to solve the problem beyond 4–5 time-steps. (Aras et al. [5] give an overview of the performance results for the full range of these techniques.)

We move beyond the single broadcast domain, however, to consider not only that particular problem but rather a full range of domains that are comparable in size and complexity. So, in order to measure how influence gap correlated with the

Problem	Agents	States	Actions	Observations
Broadcast	2	4	2	2

Table 6.1: Parameters for the set of problems used to test optimal DP.

performance of the DP algorithm, we generated a testbed of 900 distinct two-agent domains, each of which had the same number of parameters as the broadcast problem (given in Table 6.1). While of the same size as the original, the state-transition and reward dynamics of these Dec-POMDPs differ in systematic fashion, allowing us to test over a wide range of possible influence gaps. We thus separated the 900 problems into three main groups (of 300 instances each) based on their transition matrices:

Random non-independent instances: The matrix of transition probabilities for joint actions was generated at random (respecting the need for it to comprise a proper probability distribution).

Independent instances: Transition probabilities were generated randomly, under the constraint that they could be factored into independent components for each of the two agents.

Single-agent instances: Transition probabilities were generated randomly, under the constraint that only the first agent, α_1, had any influence over the outcomes (i.e., state-influence $SI_2 = 0$).

As well, within each class of 300 instances there were three subclasses, each comprised of 100 instances and divided based on the structure of their reward matrices:

Random non-equal instances: The reward generated for any joint action in any state was selected at random, uniformly from the set $\{1, 0\}$.

Fixed equal instances: The reward matrix of the literature's original broadcast problem was utilized; it is noted that in these cases, each agent has an equal influence upon the reward ($RI_1 = RI_2 = 0.0338$).

Fixed single-agent instances: The reward matrix of our preliminary single-agent experiments was utilized; only the first agent had any influence over the outcomes (i.e., reward-influence $RI_1 = 0.2158$ while $RI_2 = 0$).

In each case, we fixed observations (which simply amounted to whether a packet had arrived to the queue or not), so that neither agent had any influence over it, at any time. Thus, the observation-influence of each agent did not factor into the gap between them, and we discounted it; influence gap is then measured in terms of the absolute difference between the sum $(SI_i + RI_i)$, for each agent, α_i. This choice merely simplified our models somewhat, and had no meaningful effect on the results.

Given our testbed cases, we calculated the following measures for each problem (all values are given in *nats*, using the natural log in our calculations for convenience):

State-influence: SI_i for each agent α_i; this value fell in the range $[0, 0.504]$.

Reward-influence: RI_i for each agent α_i; this value fell in the range $[0, 0.3804]$.

Influence gap: The difference in influence; this value fell in the range $[0, 0.394]$.

For each problem instance, the DP algorithm was run as usual. Since the algorithm operates deterministically, providing an identical output each time, there was no need to run it more than once per instance. We were not interested in absolute timing data for the algorithm, which in any case could vary considerably, from a few seconds on easy problems to upwards of an hour on the hardest instances.

Instead, performance was measured by allowing DP to run until the number of policy-trees generated at the backup stage for any one agent exceeded 1200, at which point further progress was infeasible, given time and memory constraints.[1] Further-

[1]Our test machine had a dual 2.5 GHz PowerPC G5 processor and 2GB RAM. Increasing these resources might have allowed a somewhat larger cap on the number of trees, but would not have affected our results in any qualitative manner. All prior testing of optimal DP, on all available platforms, indicates that we were operating very close to the algorithm's feasible limits. In addition, these results still establish the correlation we sought, since they allowed us to separate problem instances in a meaningful way.

more, if a problem-instance could be solved to a time-limit of 100 iterations (a 100-step policy tree) without exceeding the 1200-tree bound, the program was terminated. Such performance indicates a very easy problem, and one for which continued exploration is no longer interesting, since it is effectively possible to iterate indefinitely. (While such problems were no longer interesting to explore, their very *existence* was interesting, since no such long-range optimal solutions had ever been generated before in the literature.) This main independent variable, the number of iterations before the limit on trees was reached, thus ranged between 3 (all problems were solved to at least these many instances) and 100, our cap value. For the DP algorithm, then, this provides a straightforward empirical measure of problem difficulty.

6.1.2 Empirical Results

Upon analysis of our results, we found that the operation of the DP algorithm fell into three main classes, and we separated our problem-instances accordingly. (Table 6.2 outlines the various classes of our variables, giving the full details.)

Hard: Problems for which only 3–4 backup steps were possible before the 1200-tree limit was reached, and further progress became effectively impossible, given computational constraints. It is worth noting that 4 steps is the prior best general limit for optimal dynamic programming, suggesting that in fact prior research has concentrated mainly on hard instances.

Medium: Problems for which more than 4 backup steps are possible, but do not allow indefinite policy generation. Such problems were concentrated heavily in the range of $[5, 9]$ iterations, with 93% of the Medium instances falling there, while nearly all the remainder fall in the range of $[10, 19]$ iterations, with just two outliers solved to 32 policy-steps. While the lower range is not much greater than that for the hard problems, it in fact reflects a two-fold increase in solution size from all prior optimal results.

Variable	Source	Type	Unit	Classes	Range	Prop. (%)
Influence Gap	Problem	Dependent	nats	Zero	0	0.282
				Small	$(0, 0.05)$	0.27
				Medium	$[0.05, 0.2)$	0.22
				Large	$[0.2, 0.394]$	0.228
Steps	Algorithm	Independent	Iterations	Hard	3–4	0.407
				Medium	5–32	0.216
				Easy	100+	0.377

Table 6.2: Classes of the two main variables measured in the experiments with the optimal DP algorithm. The table includes their ranges, and their proportion among the overall set of problem instances.

Easy: Problems for which a full 100 steps are possible, and policy generation appears to continue indefinitely.

Additionally, the influence gap measure fell into four natural classes, each of which comprised relatively equal proportions of the space of all problem instances:

Zero: Agents exercised the same amount of influence, and so the difference is 0.

Small: A difference in the range $(0, 0.05)$.

Medium: A difference in the range $[0.05, 0.2)$.

Large: A difference in the remaining range, up to the maximum observed, $[0.2, 0.394]$.

Our results measure how many of the problem instances falling into each of the influence gap classes also fall into the various classes of difficulty. Our results indicate that problems with large gaps between the influences of the two agents will tend to be proportionately easier, whereas problems where the difference between agent influences falls to 0 are proportionately harder. Before presenting the results, however, we exclude a certain subclass of problems that otherwise threaten to skew the numbers unfairly in our favor.

In certain problems, the overall effect of one agent α_i is completely negated, and we have the sum of influences $SI_i + RI_i = 0$. In such cases, as we proved in Section 5.4.4,

the resulting Dec-POMDP is in fact equivalent to a single-agent problem; the other agent can plan without taking α_i into account, and α_i itself can effectively enact any randomly chosen policy at all, without affecting resulting value. Practically, this has the effect that all such instances are easily solved by the DP algorithm; indeed, we included such problems as a way to check that the method worked properly, and could indeed solve an effectively single-agent domain.

Reducing the influence of one agent to 0 has another effect, however; specifically, it tends to make the influence gap measure *larger overall*, since the gap is then just equal to the influence of the other agent. Thus, if such cases were not excluded from our results, the effect of influence gap on proportionate difficulty would seem even more strongly to indicate the correlation we claim between a large gap and an easy problem. Such single-agent problems are a special, degenerate case, and do not reflect algorithm behavior for truly multiagent decentralized systems; furthermore, the ease of solving them has everything to do with the fact that they are single-agent, rather than the gap between agent influences. Our final numbers therefore reflect this exclusion. We remove 145 problem instances from our data; all 100 instances that combine single-agent transitions with single-agent rewards, and 45 others for which the various random processes happened to produce an effectively single-agent problem. Our final results are thus based on 755 of our original 900 instances, and Table 6.2 outlines each of the categories into which we divided our main variables, along with the proportion of total cases into which the remainder fall.

Figure 6.2 compares the four categories of influence gap, showing the proportion of each set of problems falling into each difficulty class. Evidently, there is a trend towards more easier problems as the gap between agent influences increases. When the influence gap is 0, the majority (69%) of the problems are of the hardest type, solvable only to 3–4 iterations; the remainder are split almost evenly between medium problems and easy ones, with 15% falling into the latter class. As the gap increases,

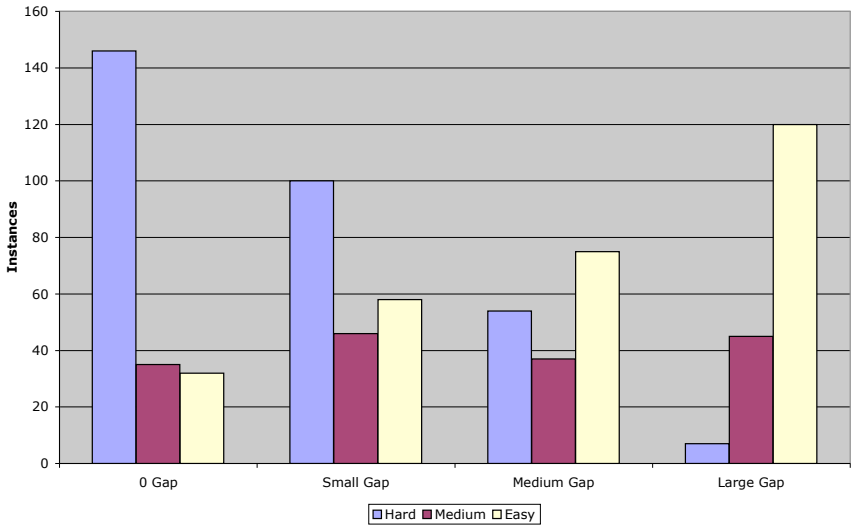


Figure 6.2: Instances falling into each range of difficulty (*Hard, Medium, Easy*), for each range of influence gap (with single-agent 0-instances omitted).

these proportions reverse themselves dramatically. When the influence gap falls into the large range, most of the problems (70%) fall into the easy difficulty class, whereas only a few (4%) remain very hard. Table 6.3 gives the full numbers.

These results confirm that there is a correlation between influence gap and problem difficulty for general Dec-POMDPs. Furthermore, our results also help account for other phenomena, observed during experimentation, that are otherwise somewhat surprising. In our original plans for this research, we had provisionally hypothesized that the special subclass of *transition-independent* Dec-POMDPs (see Section 3.1), to which specialized algorithms had previously been applied, would turn out to be easier to solve using DP than would the more general instances (especially since the former are of a lower complexity class). Contrary to this hypothesis, however, it was found that that a higher proportion of the problems featuring independent transition matrices proved difficult to solve.

This fact does not pose any challenge to the theoretical complexity results, of course, nor does it suggest a deficiency in the DP algorithm. After all, even if

Influence Gap	Total Cases	Hardness	Range (steps)	Cases	Prop. (%)
Zero [0]	213	Hard	3–4	146	0.69
		Medium	5–32	35	0.16
		Easy	100+	32	0.15
Small $(0, 0.05)$	204	Hard	3–4	100	0.49
		Medium	5–32	46	0.23
		Easy	100+	58	0.28
Medium $[0.05, 0.2)$	166	Hard	3–4	54	0.33
		Medium	5–32	37	0.22
		Easy	100+	75	0.45
Large $[0.2, 0.394]$	172	Hard	3–4	7	0.04
		Medium	5–32	45	0.26
		Easy	100+	120	0.70

Table 6.3: The correlation between the different classes of influence gap and the different levels of difficulty, for the optimal DP algorithm experiments.

transition-independent Dec-POMDPs are worst-case NP-complete, rather than being NEXP-complete, we should still expect exponential penalties in computing their solutions, and so the optimal algorithm cannot be expected to provide a general-purpose efficient method for solving them. In addition, the algorithms that solve these sorts of problems are specially designed to exploit the independent dynamics, like the Coverage Set Algorithm [12] or the bilinear programming method of Petrik and Zilberstein [99] (see Section 4.2). Thus, we cannot necessarily expect better performance from a general method, which does nothing to utilize independence. However, this still leaves the question of why DP seems to find the independent problems *harder* on average; we might have expected that, at the very least, it would be indifferent to the type of transitions featured in a given Dec-POMDP.

We can now offer an explanation for this result. If influence gap and difficulty are indeed correlated, as we have suggested, then we can predict that the larger proportion of hard problems should mean that a higher percentage of the independent problems have small influence gaps. Indeed, this is just what we found, as shown in Figure 6.3, which compares problems based on type of transition-matrix (excluding

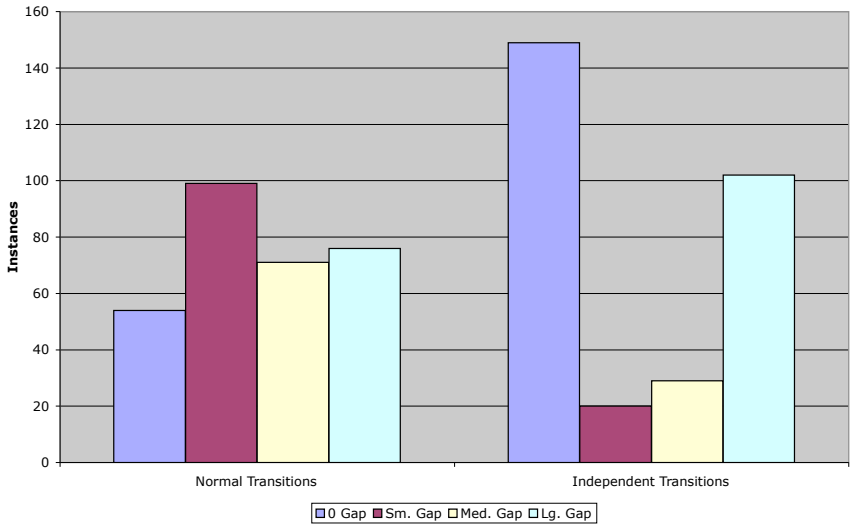


Figure 6.3: Instances falling into each range of influence gap (*0, Small, Medium, Large*), for the two types of transition structures: general, randomly completed non-independent matrices and independent matrices, respectively.

the single-agent transition problems, so we compare just 600 instances here). As we can see, more of the independent problems (50%) feature a 0-level influence gap than do the non-independent problems (18%). More of the independent problems (34%) also exhibit large gaps than do the non-independent cases (25%), but that difference is much less pronounced.[2] While transition-independent problems do fall into a lower worst-case complexity class, a large proportion of them still fall into the hardest empirical category. The fact that this proportion accords with prediction is further evidence for the correlation between influence gap and difficulty.

[2]The preponderance of 0-gap problems has to do, it seems, with the extra constraints that independence places on the form of the state-transition matrix. For these problems, this leads to more symmetrical problems, i.e., those for which each agent has the same influence, and so the gap between agents is zero (0).

6.1.3 Conclusions and Discussion

Our results demonstrate a connection between the information-theoretic influence gap measure and the ease with which a Dec-POMDP can be solved using dynamic programming. As the gap between agent influences widens, more problems become easier to solve, although some still remain of the hardest difficulty. As the gap tightens, so that agents exert the same (non-zero) amount of influence, more problems become difficult, although some are still easily solved. The fact that the observed correlation is not absolute, at either end of the scale, and that some problems do not exhibit the difficulty level generally associated with their influence gap, suggests some further factor in the system dynamics that interacts with algorithm performance.

Still, the correlation is clearly demonstrated, and we thus argue that the influence gap measure is of real interest. As we have described, prior research on exact algorithms for Dec-POMDPs has generated largely negative results, as even simple problems are found to be difficult to solve optimally. The usual response has been to turn to approximate methods, for which exact performance guarantees are not readily available, or to special algorithms only applicable to particular restricted subcases. We provide another approach. First, we have shown that there exist many problems for which optimal methods are in fact practical. Second, we provide a general-purpose heuristic measure to aid in finding those problems. Influence gap can be easily and automatically calculated in advance, for any Dec-POMDP whatsoever, and does not rely upon any special domain structure. Furthermore, since a large gap correlates with problems that tend to be easier to solve, it can be used as a means of sorting out instances to which optimal methods are more likely to be applicable.

Lastly, we note that there is a somewhat philosophical moral to these results. Some research in multiagent systems has suggested that increasing the centralization of control can lead to simplified planning [41, 135]. This intuitive idea, often observed in practice, is given support by our results. In a precise sense, an increasing influence gap

reflects a concentration of control more and more in the hands of one agent, and indeed this tends to lead to easier problems. We argue that influence gap captures a real feature of multiagent problems, namely the relative centralization of control. When the gap is zero, and a pair of agents have equal non-zero influence, the problem is in some real sense maximally decentralized, since neither of the pair has any more control than the other. As the gap grows, this trend is reversed, and one agent becomes more and more central to the system dynamics, reaching maximal centralization when that agent has all the influence, and the other has none at all.

6.2 Influence Gap and Approximate Dynamic Programming

While it is interesting to note the correlation between widening influence gap and ease of exact solution, the prior results are somewhat limited, due to the very nature of the optimal DP algorithm. As we have repeatedly stressed, optimal methods for solving Dec-POMDPs are faced with significant computational challenges, and even for large-gap problems, only very small instances were able to be solved at all. We are thus also interested in the applicability of approximate methods to Dec-POMDPs. Furthermore, we want to extend the empirical analysis to include other algorithms, in order to determine whether the observed correlation is a function of the DP algorithm alone, or holds up under application of other techniques as well.

Here, we look at the performance of two related approximate dynamic programming algorithms (described in Section 2.4), Improved Memory-Bounded Dynamic Programming (IMBDP) and a variant of same that uses Observation Compression techniques (MBDP-OC). As we shall show, there is an even more pronounced correlation between the efficacy of these methods—in terms of the ability to generate high-value solution policies—and the influence gap measure. Furthermore, we will show how the performance of the algorithms can be tuned in important ways, using knowledge supplied by our influence gap measure. This provides a means of achiev-

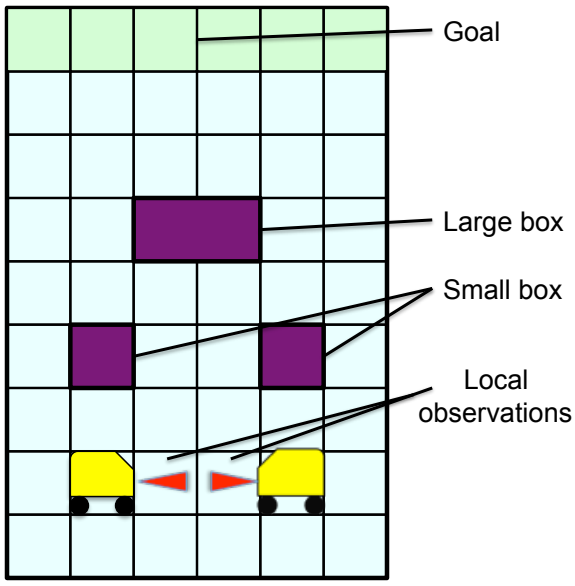


Figure 6.4: The 2-agent box-pushing test domain. Each agent observes the environment immediately in front of itself. The goal is to push boxes into the goal region; the larger boxes carry more reward, but require both agents to push them.

ing greater efficiency in the use of the approximation techniques, making intelligent choices about how to trade off solution quality and the time and resources required for computation. Using the influence gap, that is, we can evaluate a problem ahead of running our method, then set parameters of the approximate DP technique in a way that provides good quality guarantees while minimizing run-time. This work is then a first step in using a generally applicable measure of agent interaction in the design and implementation of approximation techniques.

6.2.1 The Box-Pushing Test Domain

In our experiments, we tested algorithm performance on variations of the *repeated box-pushing problem domain*. This sort of domain is relatively familiar in AI [73]; it is easily formulated as a Dec-POMDP and has been used as a test-bed for empirical testing and comparison of the IMBDP and MBDP-OC methods. Figure 6.4 gives a graphical overview of the domain, which has the following features:

Grid: The environment is specified in terms of a set of grid locations. Variations of the problem can be formed by changing the size of this grid. One special grid location (marked in black in the figure) is designated the *goal*.

Boxes: There are a number of boxes located at various points around the grid. Boxes may be of two distinct sizes, *small* and *large*; the former take up one grid square, the latter take up two of them.

Agents: A pair of agents, each representing an autonomous robot. Agents can move about the grid, although two agents may not occupy the same grid location at once. When adjacent to a box, an agent can choose to push it, by attempting to move into a grid location containing the box.

States: The global state consists of the current location of each agent, as well as the size and locations of each box, along with the current orientation of each agent.

Local Observations: Each agent observes only the grid square immediately in front of it, based on its orientation.

Action Dynamics: The domain is stochastic: whenever an agent chooses to rotate, or to move to an adjacent square, it succeeds or fails with some set probability; movements into squares occupied by other agents, or in the direction of immediately adjacent walls always fail. Boxes shift position in the direction of agent movement, if an agent successfully moves into a square containing that box, with one important exception: *large boxes require two agents to move*. That is, in order to shift a large box, the agents must work together, moving in unison into the corresponding grid locations.

Shared Reward: The agents collect a joint reward whenever a box is pushed into the goal location, dependent upon its size; these amounts can vary from instance to instance, but it is usually the case that successfully pushing the large box

Problem	Agents	States	Actions	Observations
Broadcast, etc.	2	4	2	2
Box-Pushing	2	100	4	5

Table 6.4: Number of different parameters for the simple two-agent problems, used in testing the optimal DP algorithm, compared to those for the box-pushing variants, used in testing the approximate methods.

to the goal leads to a significantly greater reward than does pushing the small box. There is generally a penalty for bumping into walls or other agents.

Time: Whenever one of the boxes is pushed to the goal location, the domain re-sets to the initial state, with agents and boxes returned to random locations. There is an overall time-limit, in terms of a number of action-steps (successful or not).

Even for relatively small grids, these box-pushing problems can be significantly more complex than those domains studied in the preceding work on optimal methods. In our tests, we employed a series of different reasonably-sized problem-instances, with parameters given as in Table 6.4, where we compare them to those used in the tests of optimal DP.

We explain the particulars of the domain in more detail below, when we discuss our experimental design. For now, we simply note that these numbers represent a significant increase in the difficulty of the problem instances. As we have previously explained, the complexity of DP backups depends exponentially on the numbers of possible observations and actions, and doubling these numbers means that these problems are effectively beyond reasonable solution, particularly given the order of magnitude increase in the number of system states (from 4 to 100).

6.2.2 Motivation for the Experiments

The key feature in the memory-bounded DP algorithms is the parameter governing the *maximum trees* retained for each agent at each backup iteration. By enforcing

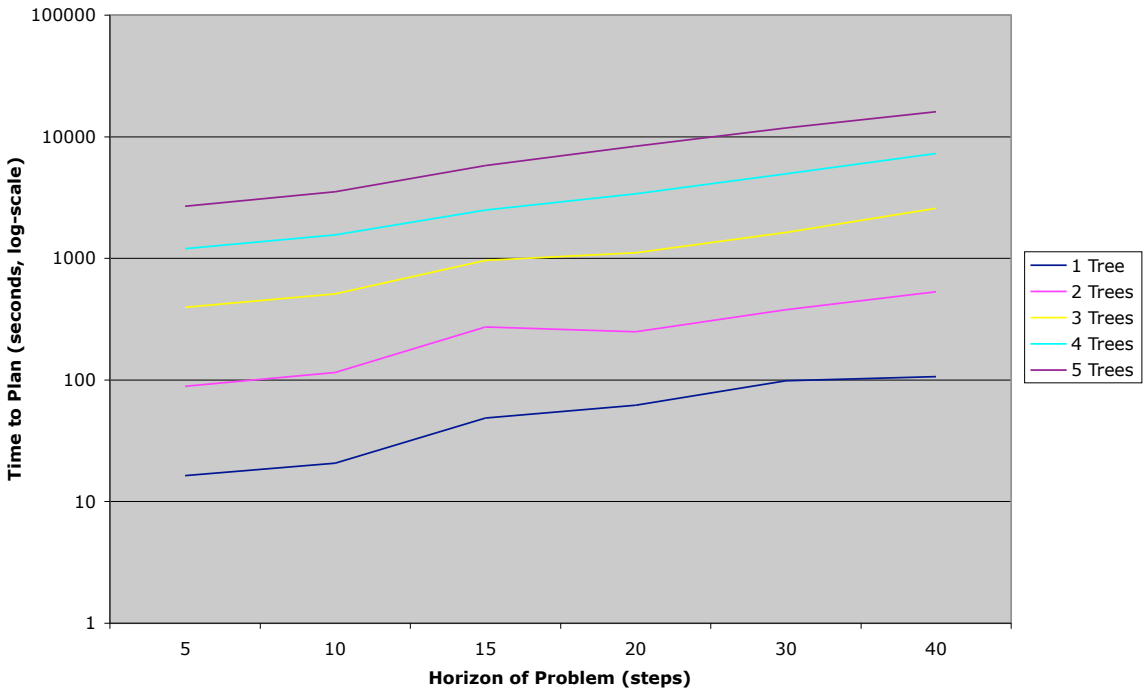


Figure 6.5: Time to calculate a policy, for an increasing number of time-steps. As we increase the maximum number of trees per agent retained at each backup iteration, planning time also increases. The data here is for the MBDP-OC algorithm, on a box-pushing problem with minimal centralization.

a hard cap on the number of trees kept to be backed up at the next step, the algorithm avoids the full combinatorial explosion that tends to arise with regular dynamic programming, and as a result can generate policies with much longer horizons. The choice of how many trees to retain is key to algorithm performance. On the one hand, the larger the cap value, the better the possible approximation.[3] On the other hand, any increase in the cap value carries with it a severe penalty, in terms of the time taken to perform computation.

Figure 6.5 illustrates this point. Shown there is the performance of MBDP-OC, run on a box-pushing problem with minimal centralization (as explained in the next

[3]In the limit, when the cap is allowed to grow arbitrarily, the approximate method simply reverts to the normal DP algorithm, since all non-dominated trees will be retained as before.

section). This is representative of the entire range of our results, and plots using the other algorithm and other degrees of centralization are essentially identical. As we can see, the time to plan increases exponentially with the time-horizon (note that the y-axis is log-scale). In addition, each increase in the maximum-trees parameter is marked across the board by what is nearly an order-of-magnitude increase in time required. When the algorithm only retains the single tree considered best at each backup step, planning for a 40-step policy takes approximately 100 seconds; by the time we move to using a maximum of five (5) trees per backup, time taken has run to over 16,000 seconds (about four and a half hours). Clearly, there is strong motivation to minimize the number of trees used, so that planning can proceed effectively.[4]

One difficulty, however, is that it is not yet known how best to set the parameters governing algorithm performance. In the existing literature, the approach has generally been to set the number of trees to a "reasonable number" (usually no more than 3), and compare algorithm performance in terms of value and time, given this essentially arbitrary cap. Here, we set out to address this issue, comparing performance of various settings over a wide range of distinct box-pushing problem instances, while also tracking how that performance varies as the influence gap between agents shrinks or grows. As we shall see, the results are interesting in a number of ways. In brief, we discover that influence gap is very strongly correlated with the overall quality of outcome policies in the two-agent box-pushing domain. As one agent's influence grows to dominate the other, the output policies become much more successful, leading to higher and higher levels of expected joint reward. Furthermore, we discover some useful properties of the maximum-trees parameter. First, we can show that it is possible to achieve much of our expected value with a relatively small number of

[4]There is also a parameter governing the maximum number of observations used in generating trees; we did not vary this in these experiments. Our results are clear enough as they are, and we will look at this parameter in future.

trees. Second, we can demonstrate that approximation in fact gets somewhat easier as influence gap increases, so that agents are able to set the maximum-trees parameter lower, while still being assured of a better overall outcome.

6.2.3 Experimental Design

We performed a wide range of experiments as follows. First, we set up a 100-state box-pushing domain, with total number of parameters as given in Table 6.4. The initial set-up for the problem was identical to that previously used in existing work for testing the two algorithms [23], and involved:

Grid: a 12-location domain.

Agents and observations: 2 agents, with 4 possible directional orientations, each with 5 total distinct observations, based on what it sees in the square immediately in front of it (nothing, a wall, the other agent, or either type of box).

Actions: Each agent has 4 possible actions, choosing to turn left or right, move ahead one square, or stay put for one step; each individual action in the base problem succeeds with base probability of 0.9 for each agent, *independent* of the actions of the other agent.

Boxes and Goals: There were 3 boxes, 2 small ones and 1 large. Small boxes could be pushed by a single agent, while the large box required both moving in concert. There was a set of 4 distinct"goal" locations in the grid, and if any box were pushed to one of those location, the problem re-set: boxes were returned to their initial locations, and agents were placed in a random configuration of location and orientation.

Reward: If a small box was pushed to a goal, reward $r = 10$ was received; for the large box, $r = 100$; if agents attempt to move forward and fail because they bump into a wall, or attempt to push the large box alone, there is a penalty

$r = -5$; finally, for attempting to move forward but failing due to bumping into the other agent the penalty was $r = -6$.

In this initial set-up, agents are entirely symmetrical, so far as the influence over the system dynamics. Effects on reward, observations, and state-action transitions are entirely identical across the pair of agents, and so the influence gap is zero (0). We thus needed to vary the domain in order to affect the gap. In order to do so, we systematically transformed the ways in which the state-transition and reward matrices were drawn up, allowing agents to have varying influence, and increasing centralization steadily as we did so. As the problem became increasingly centralized, information about action-outcomes and reward for both agents became increasingly concentrated in the hands of the first agent, α_1, although we ensured that α_2 retained at least as much influence as before.

While the actual specifics of the matrices that lead to these results are uninformative, we can show the actual influence values generated, to track increasing centralization.[5] We tested nine (9) distinct settings of the influence gaps for our problem, and in presenting our data later, we will refer to them using the values $\langle 10,\ 20,\ 30, \ldots, 90 \rangle$. These are used mainly for convenience, but were not chosen entirely arbitrarily: in generating test-cases, we controlled the revision of our matrices using a "centralization parameter" that fell in the range $[0, 100]$. At level 0, the parameter does nothing, and the problem would be identical to the original (with the zero influence gap); as the parameter goes up, α_1 comes to dominate the problem more and more, until we get to a maximum value at 100, where α_1 would effectively have full control, and α_2's

[5]We think this is another useful feature of our influence gap measure: it can identify increasing centralization even where it is not intuitively obvious. That is, a cursory inspection of the transition and reward matrices in our set of problems would not immediately suggest that some are more centralized than others. All of them feature the same sets of actions and rewards, and the numbers do not necessarily "look" any more centralized for one problem or another. Influence gap, on the other hand, allows us to calculate this sort of effect across problem instances, providing a transparent indication of differences.

Level	α_1 Inf.	α_2 Inf.	Gap	Diff.
10	0.1545	0.0270	**0.1275**	*n/a*
20	0.1726	0.0271	**0.1455**	0.0179
30	0.1934	0.0273	**0.1661**	0.0206
40	0.2162	0.0274	**0.1888**	0.0227
50	0.2408	0.0276	**0.2132**	0.0244
60	0.2667	0.0278	**0.2390**	0.0258
70	0.2939	0.0279	**0.2660**	0.0270
80	0.3221	0.0281	**0.2940**	0.0280
90	0.3511	0.0283	**0.3229**	0.0289

Table 6.5: The various "centralization levels" for the box-pushing problems, with the associated influence measures for each agent, and the gap between them. The right-most column shows the differences between the current gap and the previous level (i.e., the difference between the gap for that level and the one below it), showing that the gap grows more rapidly as the levels go up. All values are in nats.

influence would in fact revert to 0. Since we were not interested in the latter case, where the problem is in fact just a single-agent MDP, and wanted to move beyond the original problem, we tested the effect of our centralization parameter at each 10 units in between. Table 6.5 gives the actual values for each agent's influence, and the associated gap, for these centralization levels.

Having created the distinct problems, with differing influence gaps, we performed a wide-ranging series of tests, varying the following quantities for each of the two algorithms, IMBDP and MBDP-OC:

Influence gap: 9 levels, 10–90, as just described.

Maximum trees: 5 settings, from 1–5 trees per agent at each backup iteration.

Time Horizon: 6 levels, ranging from 5–40 action-steps for finite policy generation.

We tested all possible combination of these parameters, testing 270 total settings per algorithm. Further, due to the fact that we used the *random policy heuristic* for each solution method (detailed in Section 2.4.1), performance could vary somewhat on different runs over the same problem instance. Although the variation was not

extremely pronounced, we compensated for it, and sought to eliminate artifacts in the timing data, by performing 20 runs of each of the variable combinations (for 5400 total experimental runs). We then measured two independent variables, averaged over the 20 runs in each case:

Solution Time: total time in seconds to provide a solution.

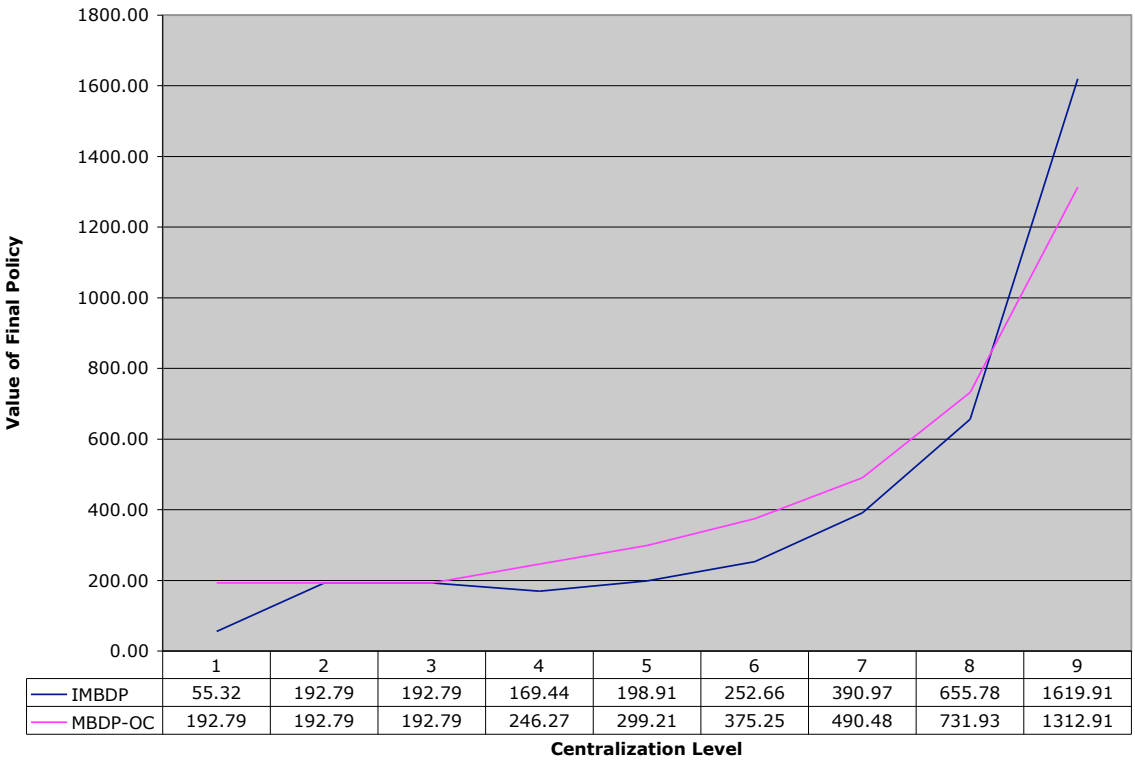
Solution Value: the expected value of the policy generated.

Experiments were run on the University of Massachusetts Amherst Computer Science Swarm Cluster.[6] Each set of combined parameters for testing was run on a single compute node of the Swarm; each node consisted of an 8-core Xeon 5355 2.66 GHz processor, running with 16GB of RAM and 250GB of hard drive space.

6.2.4 Empirical Results

Precise timing data was not central to our concerns here, but we note a few things about the results. First, as already discussed—and as illustrated in Figure 6.5—planning time increases dramatically across the time horizon, and goes up an order of magnitude with each increase in the maximum number of trees. This is as expected, given what we know about the cost of doing backups and pruning; indeed, we collected timing data to confirm and illustrate the points made in the prior discussion. Also expected, but worth noting, is that time to solution is not significantly affected by the influence gap. That is, given a particular fixed cap on the number of trees used by the algorithms, and a given number of policy-steps to compute, each method runs in essentially the same amount of time on all problems of a given size, no matter the influence gap in question. This is expected, since the algorithms retain the same number of trees (given by the cap size), no matter how hard or easy the problem may

[6]See the Swarm page at `http://www.cs.umass.edu/~swarm/index.php?n=Main.HomePage` for details. Many thanks go to Andre Gauthier for assistance with use of the system.



	1	2	3	4	5	6	7	8	9
IMBDP	55.32	192.79	192.79	169.44	198.91	252.66	390.97	655.78	1619.91
MBDP-OC	192.79	192.79	192.79	246.27	299.21	375.25	490.48	731.93	1312.91

Figure 6.6: The maximum value of policies, relative to the increasing influence gap. Value shown is for the longest-horizon (40-step) policy-trees, using the maximum number (5) of trees for the two algorithms.

be. Further, top-down heuristic generation is the same throughout, and different gaps make no real difference to the process of individual policy-tree evaluation.

Thus, the only real way to have a significant impact upon the run-time of the approximate DP algorithms is to limit the number of trees used. As we shall now argue, however, this is exactly where the influence gap measure can be useful. Our results show that an increasing gap has two consequences. First, we see a gain in absolute value of the policies generated, with large-gap problems and increased centralization leading to dramatic increases in overall value. Second, we see that the more centralized problems are in a real sense easier to approximate, since a larger proportion of the overall value can be gained using fewer trees overall.

The first of these outcomes is demonstrated in Figure 6.6. This plot gives the total value (in r units) accumulated for the longest policies generated, the 40-step horizon trees, using the maximum number (5) of trees in each algorithm, relative to the increasing information gap across our problem-instances. On the one hand, we are not interested in comparing the two algorithms over the particular instances.[7] On the other hand, it is notable that both algorithms display very similar behavior: almost without exception, value of resulting policies trends upwards with increasing influence gap, beginning to climb quickly at gaps above level 60, and trending sharply upwards from there in a seemingly exponential form (the sole exception to the trend being a slight down-turn in value for IMBDP between levels 30 and 40). Thus, there is a great difference in value between the least and most centralized problem-instances.

This is very interesting, particularly because it is borne out for both algorithms, and accords with what we have observed in the optimal DP case. As influence gap increases, our methods are able to find much better policies, and the problems provide much greater potential rewards, using the same resources in each case. Furthermore, our results also reveal interesting features of the ways in which value was accumulated, relative to influence gap. By comparing how much of the final value of the longest policies could be gained by using different caps on the number of maximum trees, we can see how much each increase in this value contributed to the best possible solution.

These results are shown in Figure 6.7, where we compare the percentages of total value possible that can be attributed to each possible number of trees. That is, for any value of $n \in [1, 5]$, we take the total value gained for a 40-step policy generated using n trees, minus the value of the policy generated using $(n-1)$ trees, and express it as a percentage of the total possible value (using 5 trees). This then isolates the percentage

[7]It is interesting to note, however, that MBDP-OC dominates IMBDP over all problems from centralization levels 10–70, but is then surpassed at the levels higher than that. We do not have any suggestions as to why this may be so, and further testing would be required to see if it is a genuine and persistent phenomenon. Further work on the subject will take up the question.

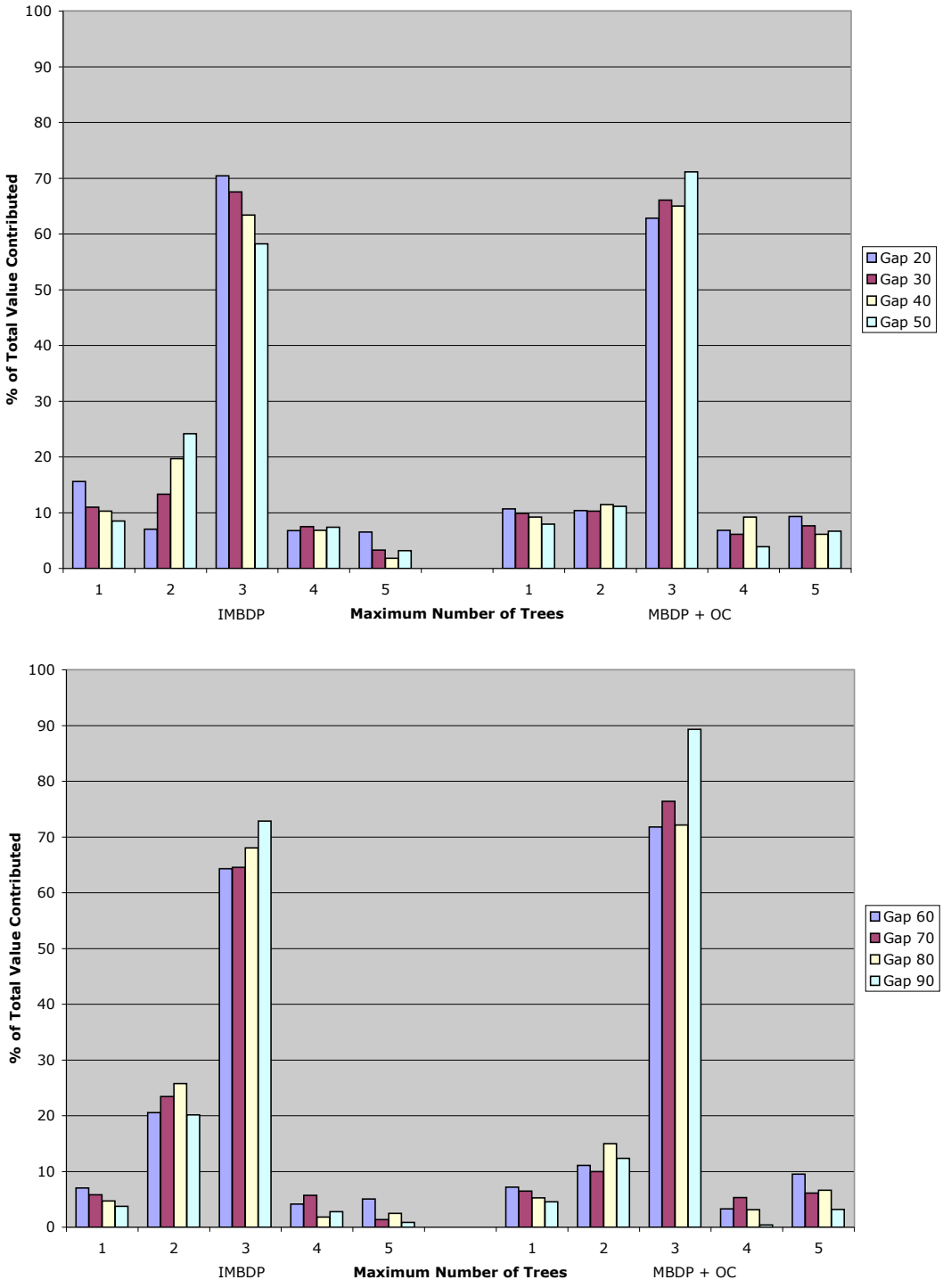


Figure 6.7: The percentage of the maximum value accumulated (i.e., using 5 maximum trees per backup) due solely to using a particular number of trees. Data is given for lower and higher sets of influence gap ranges, for both algorithms.

of overall possible value that can be attributed to using n as our cap for backups. The figure shows these values for various levels of influence gap, from the lowest values and least-centralized problems (at top), to the highest and most-centralized problems (at bottom), for each of the two algorithms.

Two things stand out about these results. First, they are remarkably consistent in that, in all cases, the greatest majority of overall value can be gained using just three (3) trees per backup step. After that, the marginal value to be gained drops off sharply. This means that the great majority of the total value to be had from the various instances of the box-pushing domain is in fact gained from using just three trees. While numbers differ from level to level, all problem instances allow at least 80% of the best-observed value to be gained using only 3 trees, even given the great range in absolute total value to be had. Thus, the substantial time-penalties suffered when moving up to 4 or 5 trees only brings with it rather modest increases in value.

The second interesting feature comes when we look a little closer at the differences between levels, however. Comparing the top and bottom plots in the figure, we can see that as influence gap goes up, performance is increasingly *front-loaded*: at the higher gaps, more and more of the overall value is accumulated in using fewer trees, while for the lower gaps, there is still some substantial gain to be had by moving on to the greater number of trees. This is clearly illustrated when we plot the performance profiles of our algorithms, as shown in Figure 6.8, where we normalize absolute values of the best policies over the different levels of influence-gap, and show how this value accumulates as a percentage, given the increasing number of trees used. (For clarity in presentation, we have isolated the odd-numbered levels of influence-gap.)

As we can see, both algorithms exhibit the same trend: as the gap grows, the peak found at the level of 3 trees grows sharper, and things flatten out much more. (There is a slight bit of noise in the plot for MBDP-OC, where gap-level 50 and 70 are virtually identical.) The effect is much more pronounced for IMBDP: for that

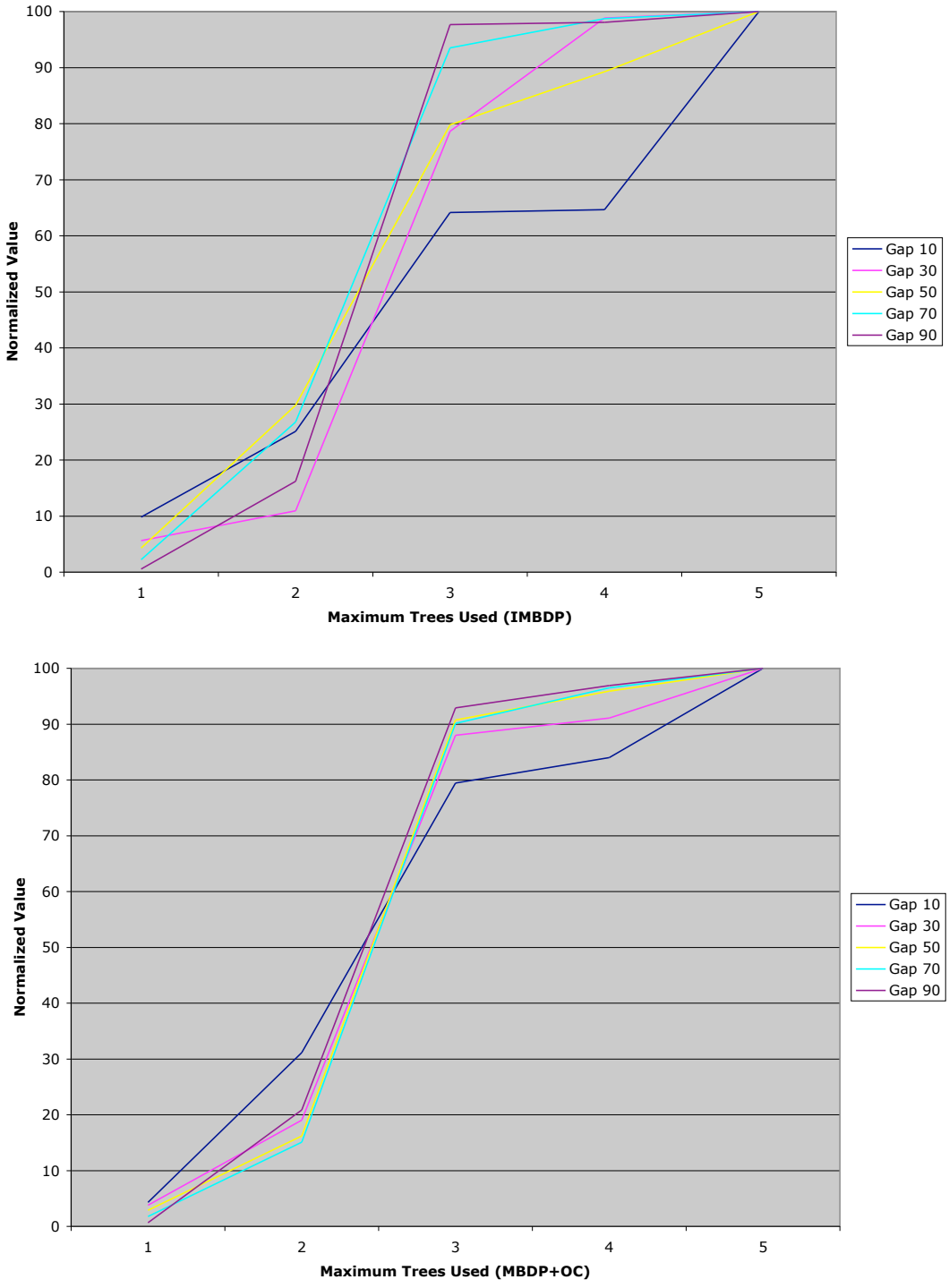


Figure 6.8: The performance profiles for both algorithms. Values of best policies are normalized, and we compare the amount of total value accumulated, using different numbers of trees. Profiles shown for odd-numbered influence-gap levels only.

algorithm, only about 65% of total value is accumulated using 3 trees at influence-gap level 10, whereas 98% of the value is gained using the same number for level 90, with a relatively smooth progression between them. For MBDP-OC, the effect is somewhat less drastic, but the value still ranges from 79–96%. At the level of 4 trees, there are still gains of at least 15% of total value to be made at the lowest level, while the higher levels have leveled off considerably (admittedly, this is only really evident at the least-centralized level 10). Thus, as the gap grows between agents, there is a real sense in which our ability to approximate efficiently increases.

6.2.5 Conclusions and Discussion

These results supplement those discovered for the optimal DP algorithm, and extend them in significant and interesting ways. On the one hand, we see that a widening influence gap continues to correlate with improved algorithm performance. As the gap widens, our agents are able to generate policies that go much further, and gain much more value, using no more (and often substantially less) in the way of computational resources.

This result was by no means guaranteed: while anecdotal evidence had previously suggested that influence gap had a similar effect on the performance of other algorithms beyond exact DP, it was possible that the effects were algorithm-specific, or somehow limited to the small parameter sets used in prior testing. While it is true that the approximate methods under consideration use DP backups, too, their reliance on top-down search and various forms of heuristic policy completion mean that they operate in some fundamentally different ways. Furthermore, the box-pushing problems we have used go far beyond the sorts of dynamics seen in the prior experiments, ranging over far larger state-sets.

In addition, our latest empirical results suggest some exciting new implications for the influence measure. Not only do the solutions found for wider-gap problems

yield significantly greater absolute value, the algorithms converge much more quickly towards the maximums observed. Referring back to the relationship between time to plan and the maximum number of trees used, as discussed in connection with Figure 6.5, we can see how significant this approximation effect actually is. For a 40-step policy, the time to plan using 3 maximum trees is 35% of that required when using 4 trees, and 16% of the time required to use a full 5 trees per iteration. These effects grow ever greater as the time-horizons are extended, and beyond a certain point, planning with the largest number of trees is simply infeasible. In the case of small-gap problems, however, the time-penalty may be thought worthwhile, since only 65–80% of the total possible value can be expected if we use only 3 trees, and we may feel that using 4 or 5 is advisable. For the large-gap problems, better guarantees are available: not only is absolute value much higher, but upwards of 95% of the total observed can be gained using just the 3 trees.

In some sense, of course, these results come by benefit of hindsight. That is, we ran our algorithms using various parameter-sets, and then analyzed the results to reveal these trends, after already having used the full number of trees on all instances. While this is true, we argue that the usefulness of our influence-gap measure lies in the fact that it can now be applied to *new instances* of a problem-domain, in advance of running our algorithms. For new instances of the box-pushing domain, we can calculate the influence-gap ahead of time, then choose our approximation parameters accordingly, with some real expectation about the quality—absolute and relative—of our results.

For new classes of problems, of course, some work has to be done in advance. Since the range of the influence-gap is problem-specific, there are no fixed levels that

correspond to "high" and "low" gaps across the board.[8] Given a new set of instances, we thus will need to calculate the influence-gaps over a range of representative instances, and perhaps do some off-line testing to establish our performance profiles. Nonetheless, once these sorts of results are known for a new class of problems, influence gap provides a straightforward way of classifying given instances encountered in practice. The results we have seen so far lead us to expect that we can apply these measures in more and more domains, with similar outcomes.

[8]That is, while we know that the various influences for each agent are positive, they are unbounded above. Thus, as the dynamics of systems grow more complex, and more distinct values of the associated quantities are available, the influence and associated gap will grow.

CHAPTER 7

CONCLUSION

7.1 Summary of Contributions

Our work here has focussed on the ways in which agents may interact with one another and influence the dynamics of a decentralized, cooperative control problem. We have extended existing knowledge of these problems in a number of new directions, and we have answered some lingering questions about some existing models of agent interaction.

We began by outlining the general formal model on which all this work is based, the decentralized partially observable Markov decision process, or Dec-POMDP, and touched on its jointly fully observable Dec-MDP variant. We looked also at some well-understood algorithms for solving them, based on either optimal or approximate dynamic programming. As they are NEXP-complete, these problems are extremely hard to solve in general. We gave an overview of the proof of this fact, and looked at a known result about the reduced complexity of problem instances that can be factored into separate sub-problems for each agent. In particular, when the local state-transitions and observations of individual agents are independent, and the only connection is via the shared reward function, the problems are then NP-complete.

Our first major results established the limitations of this reduction. We showed first that when the situation is reversed, and only rewards are independent, the problem remains NEXP-complete, as in the general case. NEXP-completeness was also demonstrated from two interesting models taken from the literature, using event-based dependencies between actions on the one hand, and state-based action selec-

tion on the other. Each of these models seeks to isolate simpler types of decentralized problems by restricting the form of interaction, without making the agents wholly independent; however, our results show that this is not possible in general. In both cases, these fact were previously unknown, and they significantly strengthen the known lower bounds, establishing them firmly for the first time. We were also able to identify precisely when problems with event-based dependencies do in fact drop from NEXP-hard to NP-hard; as we showed, this holds exactly when the number of dependencies is no more than a logarithmic factor of the overall state-space size.

We then established some new properties of the Dec-MDP model with shared reward, but otherwise independent dynamics. As we showed, there is a precise connections between the number of events necessary and sufficient to specify the joint reward function and the essential dimensionality of the problem, where the latter is understood in terms of the size of matrices used to solve such Dec-MDPs via mathematical programming techniques. In addition, we demonstrated that a process of dimensionality reduction, using basic matrix algebra techniques, allows such problems to be reformulated in such a way that the number of events is provably minimized.

While the hardness results can be viewed pessimistically, we stress that worst-case complexity is not always correlated with the actual difficulty of solving most problem instances. With that in mind, we sought to examine how the average-case empirical hardness of problems related to their interaction structure. We note that there are many restricted models of agent interaction. Rather than proposing yet another new variant, we chose to take a more general approach. We defined an information-theoretic measure of agent influence, in terms of how much knowing an agent's actions would tell us about the underlying state dynamics (i.e., the transition-outcomes, observations, and rewards). A given problem's *centralization* can then be understood in terms of the gap between these influences for different agents: as this gap grows, one agent's influence comes to dominate more and more. These influence

measures apply to all Dec-POMDPs, and we were able to bound their possible values. In addition, we showed that they are precise evaluators of the importance of certain agents in the planning process in exactly one way: if an agent has no influence, according to our way of reckoning it, then that agent can be discounted in all planning. We also extended the general influence measures to the case of Dec-POMDPs with locally factored state-spaces, and established analogous exclusion results when various possible independence relations hold (or not).

Our empirical work moved this analysis beyond the limit cases where agents are provably without any influence at all. In that regard, our results are two-fold. First, for the optimal dynamic programming solution algorithm, we showed that there is a correlation between an increasing influence-gap and ease of solution. As problems grow more centralized, and one agent dominates, we are more often able to solve them optimally while using a reasonable amount of computational resources, generating optimal finite-policy solutions for much longer time-horizons than ever before possible. This suggests that we can use the influence measure as a means of identifying problem instances for which optimal methods can actually be effective.

Furthermore, where an exact solution is still not possible, our results show that influence gap can still predict the performance of approximate algorithms. Our second set of empirical results establish that a widening gap correlates with better performance for two memory-bounded inexact dynamic programming variants. As the gap grows, increased centralization leads to an increase in the ability to find high-valued policies. In addition, there is evidence that the algorithms can work in a more efficient fashion: since we can calculate the influence gap in advance, we can set control parameters in such a way that we can expect to achieve better approximations while minimizing time and space required for computation.

Overall, our work here solidifies our understanding of important issues in multi-agent decision making. Given how hard the problem, even in apparently simplified

forms, has proven to be, new ways to identify simplifications and important structures are needed. Our general measures provide some foundations for further work on this necessary next step.

7.2 Future Work

Many open and interesting questions remain, or are newly suggested by the results we have established here. We end by describing some of these, making suggestions where we can for the directions in which future research might progress.

7.2.1 Ignoring Low-Influence Agents

We have shown that when an agent has no influence, as we define it, in a general or factored Dec-POMDP, planning can proceed without needing to take that agent into account. A related issue is then what happens if we plan while ignoring agents that *do have* some influence over the system's dynamics. Clearly, in the worst case, this sort of approach would be disastrous. However, if applied intelligently, it might have some real potential; that is, if we only chose to ignore agents with *very little or no* influence, the resulting policies might still work quite well. It would be interesting to consider this question both analytically and empirically. On the one hand, we would like to consider the problem of bounding the potential negative effect on policy value that comes with ignoring low-influence agents. Ideally, this could be given in terms of the measure itself. That is, we should like to be able to express the difference between the value of an optimal, fully cooperative joint policy $\pi^\star$, and the value of some policy π_{-i} that was arrived at while ignoring agent α_i, as a function of influence $\mathcal{I}(\alpha_i)$. While this might be difficult to determine, an empirical profile of the strategy would be an interesting next step, investigating the actual declines in policy-value encountered by following it, and charting how they relate to the computational advantages, in terms of time and space required for planning, that result.

7.2.2 Influence and Problem Decomposition

Another possible approximation strategy would involve *changing system dynamics* so that agent influences decouple. That is, rather than ignoring agents with little or no influence, and then planning for the original domain as if they did not exist, we would look for ways to change the structure so that the new version allowed for better solution. For example, we might consider new organizational structures for hierarchical domains, evaluating them based on the relative sizes of the influence gaps in each one. By settling on high-gap organizations—that is, those that are relatively centralized—we might then expect to find more easily possible plans within those structures, according to the empirical results we have already seen. Alternatively, agent's could seek to eliminate from consideration any actions that might cause them to influence one another. Again, there will obviously be cases where such a strategy will not work, since there are problems that can only be solved at all if agents actually work together. However, it is certainly possible that agents can often find ways of eliminating interactions where influence levels are high, perhaps by agreeing ahead of time not to take certain possible actions in certain circumstances, thus eliminating those events from the possible system dynamics. Given the choice of several events to avoid, they might choose those where the influence measures were the greatest (e.g. , where the mutual information between actions and some state-variable were highest). Again, we should like to be able to bound the effect such choices might have on outcome policy value, and to investigate the effects empirically.

7.2.3 Limited Interactions and Reduced Complexity

While we have shown that some models of apparently restricted interaction are still highly complex in general, we also demonstrated that in at least one case we can bound complexity more strictly. When the structure of dependencies between agents is suitably small, we showed, worst-case hardness falls by an exponential factor (from

NEXP to NP). It would be worth trying to find similar cases in other known models, since the general complexity results seem to apply quite broadly. Further, we would like to examine the practical consequences of such reductions in essential hardness. As our proofs have it, the number of dependencies must be no more than logarithmic in the overall state space. This is a somewhat abstract characterization of an interesting and potentially important sub-class of problems. We would like to see if there are common-sense and real-world examples of such structures. One example that comes to mind are those domains in which the time at which events occur is an essential feature of the state-space, but most if not all actions are indifferent as to the time at which they occur. In such cases, it might be possible to specify dependencies in terms of a small collection of events, since all states that are identical apart from time variable will be treated the same as far as actions go. With a long time-horizon, and so a large resulting state-space, these savings can be considerable, and the logarithmic property can well hold. This is worthy of more study.

7.2.4 Learning with Interactions and Approximation

Another interesting question concerns using the structure of reduced interaction models as tools in learning useful policies for decentralized problems. Thomas [127] and Guo [54] have both shown how to use special forms of interactions in a reinforcement learning framework. In each case, agents are able to learn action policies that produce far more value in far less time than is possible using a general learning technique which ignores the nature of the interactions. Such work can be extended to other interaction models.

In addition, it would be interesting to combine learning approaches with the sorts of heuristic problem-reduction methods just discussed. When faced with a large and complex problem, real-world intelligent agents often simplify their planning by identifying certain key points at which their actions and those of others interact.

Planning for these key elements, to reduce conflict or enhance cooperation, generally precedes (or even supersedes) the fine details when it comes to determining individual courses of action. In addition, many incidental interactions between agent plans are ignored, often because their apparent possible effect on the outcome is negligible. One might then consider ways in which agents may learn to do this sort of identification and hierarchization process effectively, picking out interactions which seem to require more or less planning. Another possibility is that learning can be used to converge upon improved measures of general interaction, replacing a quantity like influence gap with other, better predictors of problem difficulty.

BIBLIOGRAPHY

[1] Allen, Martin, Goldman, Claudia V., and Zilberstein, Shlomo. Learning to communicate in decentralized systems. In *Proceedings of the Workshop on Multiagent Learning, Twentieth National Conference on Artificial Intelligence (AAAI-05)* (Pittsburgh, PA, 2005), pp. 1–8. AAAI Tech. Report WS-05-09.

[2] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Optimal fixed-size controllers for decentralized POMDPs. In *Proceedings of the Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains* (Hakodate, Japan, 2006). Held in conjunction with the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

[3] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence* (Vancouver, British Columbia, Canada, 2007).

[4] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (Hyderabad, India, 2007), pp. 2418–2424.

[5] Aras, Raghav, Dutech, Alain, and Charpillet, François. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Providence, RI, 2007), pp. 18–25.

[6] Aström, Karl J. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications 10* (1965), 174–205.

[7] Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. Transition-independent decentralized Markov decision processes. In *Proc. 2nd Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems* (2003), pp. 41–48.

[8] Becker, Raphen. *Exploiting Structure in Decentralized Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2006.

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

[10] Becker, Raphen, Lesser, Victor, and Zilberstein, Shlomo. Analyzing Myopic Approaches for Multi-Agent Communication. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (Compiegne, France, 2005), pp. 550–557.

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

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

[13] Bernstein, D. S., Hansen, E. A., and Zilberstein, S. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* (Edinburgh, Scotland, 2005), pp. 1287–1292.

[14] Bernstein, D. S., Hansen, E. A., Zilberstein, S., and Amato, C. Dynamic programming for decentralized POMDPs. In *Proc. AAAI Spring Symposium on Bridging the Multi-Agent and Multi-Robot Research Gap* (March 2004).

[15] Bernstein, Daniel S. *Complexity Analysis and Optimal Algorithms for Decentralized Decision Making*. PhD thesis, University of Massachusetts, Amherst, 2005.

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

[17] Bernstein, Daniel S., and Zilberstein, Shlomo. Reinforcement learning for weakly-coupled MDPs and an application to planetary rover control. In *Proceedings of the Sixth European Conference on Planning* (Toledo, Spain, 2001).

[18] Bernstein, Daniel S., Zilberstein, Shlomo, and Immerman, Neil. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (Stanford, California, 2000), pp. 32–37.

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

[20] Boutilier, Craig, Dean, Thomas, and Hanks, Steve. Decision-theoretic planning: Structual assumptions and computational leverage. *Journal of Artificial Intelligence Research 1* (1999), 1–93.

[21] Boutilier, Craig, Dearden, Richard, and Goldszmidt, Moises. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (Montreal, Canada, 1995), pp. 1104–1111.

[22] Boutilier, Craig, and Poole, David. Computing optimal policies for partially observable Markov decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (Portlan, Oregon, 1996), pp. 1168–1175.

[23] Carlin, Alan, and Zilberstein, Shlomo. Value-based observation compression for DEC-POMDPs. In *Proceedings of the Seventh International Conference on Autonomous Systems and Multiagent Systems* (Estoril, Portugal, 2008), pp. 501–508.

[24] Carver, Norman, and Lesser, Victor. Nearly Monotonic Problems: A Key to Effective FA/C Distributed Sensor Interpretation? *Proceedings of Thirteenth National Conference on Artificial Intelligence* (1996), 88–95.

[25] Carver, Norman, and Lesser, Victor. Domain Monotonicity and the Performance of Local Solution Strategies for CDPS-based Distributed Sensor Interpretation and Distributed Diagnosis. *Autonomous Agents and Multi-Agent Systems 6* (2003), 35–76.

[26] Casper, J., Micire, M., and Murphy, R. Issues in intelligent robots for search and rescue. In *SPIE Ground Vehicle Technology II* (2000), vol. 4024, pp. 292–302.

[27] Cassandra, Anthony R. Optimal policies for partially observable markov decision processes. Tech. Rep. CS-94-14, Brown University Department of Computer Science, Providence, RI, 1994.

[28] Cassandra, Anthony R. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Department of Computer Science, Providence RI, 1998.

[29] Cassandra, Anthony R., Littman, Michael L., and Zhang, Nevin L. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence* (Providence, RI, 1997), pp. 54–61.

[30] Chalkiadakis, Georgios, and Boutilier, Craig. Coordination in multiagent reinforcement learning: A Bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems* (Melbourne, Australia, 2003), pp. 709–716.

[31] Chalkiadakis, Georgios, and Boutilier, Craig. Bayesian reinforcement learning for coalition formation under uncertainty. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (New York, New York, 2004), pp. 1090–1097.

[32] Cheng, Hsien-Te. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, BC, Canada, 1988.

[33] Claus, Caroline, and Boutilier, Craig. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceddings of the Fifteenth National Conference on Artificial Intelligence* (Madison, Wisconsin, 1998), pp. 746–752.

[34] Cogill, Randy, Rotkowitz, Michael, van Roy, Benjamin, and Lall, Sanjay. An approximate dynamic programming approach to decentralized control of stochastic systems. In *Lecture Notes in Control and Information Sciences*, vol. 329. Springer, 2006, pp. 243–256.

[35] Cottle, Richard W., Pang, Jong-Shi, and Stone, Richard E. *The Linear Complementarity Problem*. Academic Press, 1992.

[36] Dean, Thomas, Kaelbling, Leslie Pack, Kirman, Jak, and Nicholson, Ann. Planning under time constraints in stochastic domains. *Artificial Intelligence 76* (1995), 35–74.

[37] Decker, Keith S., and Lesser, Victor R. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management 2* (1993), 215–234.

[38] Feng, Zhengzhu, and Hansen, Eric A. Symbolic heuristic search for factored Markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)* (Edmonton, Alberta, Canada, 2002), pp. 455–460.

[39] Feng, Zhengzhu, and Zilberstein, Shlomo. Region-based incremental pruning for POMDPs. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence* (Banff, Alberta, Canada, 2004), pp. 146–153.

[40] Feng, Zhengzhu, and Zilberstein, Shlomo. Effcient maximization in solving POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (Pittsburgh, Pennsylvania, 2005), pp. 975–980.

[41] Georgeff, M. A theory of action for multi-agent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence* (Austin, Texas, August 1984), pp. 121–125.

[42] Gmytrasiewicz, Piotr, and Doshi, Prashant. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research 24* (2005), 49–79.

[43] Goldman, Claudia V., Allen, Martin, and Zilberstein, Shlomo. Decentralized language learning through acting. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (New York City, NY, July 2004), pp. 1006–1013.

[44] Goldman, Claudia V., Allen, Martin, and Zilberstein, Shlomo. Learning to communicate in a decentralized environment. Tech. Rep. UM-CS-2006-16, University of Massachusetts Amherst, Department of Computer Science, 2006.

[45] Goldman, Claudia V., Allen, Martin, and Zilberstein, Shlomo. Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems 15*, 1 (2007), 47–90. DOI URL: `http://dx.doi.org/10.1007/s10458-006-0008-9`.

[46] Goldman, Claudia V., and Zilberstein, Shlomo. Mechanism design for communication in cooperative systems. In *Proceedings of the Fifth Workshop on Decision Theoretic and Game Theoretic Agents* (Melbourne, Australia, July 2003), pp. 33–40.

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

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

[49] Goldman, Claudia V., and Zilberstein, Shlomo. Goal-oriented Dec-MDPs with direct communication. Technical Report UM-CS-2004-44, University of Massachusetts, Amherst, Department of Computer Science, 2004.

[50] Guestrin, Carlos, and Gordon, Geoffrey. Distributed planning in hierarchical factored MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence* (Edmonton, Alberta, Canada, 2002), pp. 197–206.

[51] Guestrin, Carlos, Koller, Daphne, and Parr, Ronald. Multiagent planning with factored MDPs. In *Proceeding of Advances in Neural Information Processing Systems* (Vancouver, British Columbia, Canada, 2001), pp. 1523–1530.

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

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

[54] Guo, AnYuan. *Planning and Learning for Weakly-Coupled Distributed Agents*. PhD thesis, University of Massachusetts Amherst, 2006.

[55] Guo, AnYuan, and Lesser, Victor. Planning for Weakly-Coupled Partially Observable Stochastic Games. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Edinburgh, Scotland, July 2005), pp. 1715–1716.

[56] Guo, AnYuan, and Victor, Lesser. Stochastic Planning for Weakly-Coupled Distributed Agents. In *Proceedings of the Fifth Joint Conference on Autonomous Agents and Multiagent Systems* (Hakodate, Japan, 2006), pp. 326–328.

[57] Hansen, Eric, and Feng, Zhengzhu. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling* (Breckinridge, Colorado, 2000), pp. 130–139.

[58] Hansen, Eric A. *Finite-Memory Control of Partially Observable Systems*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst, 1998.

[59] Hansen, Eric A., Bernstein, Daniel S., and Zilberstein, Shlomo. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (San Jose, California, 2004), pp. 709–715.

[60] Hansen, Eric A., and Zhou, Rong. Synthesis of hierarchical finite-state controllers for POMDPs. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS-03)* (Trento, Italy, June 2003).

[61] Hauskrecht, Milos. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research 13* (2000), 33–94.

[62] Horling, Brian, Mailler, Roger, Shen, Jiaying, Vincent, Regis, and Lesser, Victor. Using autonomy, organizational design and negotiation in a distributed sensor network. In *Distributed Sensor Networks: A multiagent perspective*, Victor Lesser, Charles Ortiz, and Milind Tambe, Eds. Kluwer Academic Publishers, 2003, pp. 139–183.

[63] Horst, Reiner, and Tuy, Hoang. *Global Optimization: Deterministic Approaches*. Springer, 2003.

[64] III, James W. Lark. *Applications of Best-First Heuristic Search to Finite-Horizon Partially Observed Markov Decision Processes*. PhD thesis, University of Virginia, Charlottesville, Virginia, 1990.

[65] Jensen, Johan L. W. V. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica 30*, 1 (1906), 175–193.

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

[67] Kallenberg, L. C. M. *Linear Programming and Finite Markovian Control Problems*. Mathematical Centre Tracts, Amsterdam, 1983.

[68] Kapetanakis, Spiros, and Kudenko, Daniel. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (Edmonton, Alberta, Canada, 2002), pp. 326–331.

[69] Kearns, Michael, and Mansour, Yishay. Efficient Nash computation in large population games with bounded influence. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence* (Alberta, Canada, 2002), pp. 259–266.

[70] Kim, Y., Nair, R., Varkantham, P, Tambe, M., and Yokoo, M. Exploiting locality of interaction in networked distributed POMDPs. In *Proceedings of the AAAI Spring Symposium on Distributed Planning and Scheduling* (2006).

[71] Koller, Daphne, and Megiddo, Nimrod. The complexity of zero-sum games in extensive form. *Games and Economic Behavior 4*, 4 (1992), 528–552.

[72] Koller, Daphne, and Megiddo, Nimrod. Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory 25*, 1 (1996), 73–92.

[73] Kube, C. Ronald, and Zhang, Hong. Task modelling in collective robotics. *Autonomous Robots 4* (1997), 53–72.

[74] Lesser, V., Decker, K., T.Wagner, Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., NagendraPrasad, M., Raja, A., Vincent, R., Xuan, P., and Zhang, X.Q. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems 9*, 1 (2004), 87–143.

[75] Lewis, Harry R. Complexity of solvable cases of the decision problem for predicate calculus. In *Proceedings of the Nineteenth Symposium on the Foundations of Computer Science* (Ann Arbor, Michigan, 1978), pp. 35–47.

[76] Littman, Michael L., Cassandra, Anthony R., and Kaelbling, Leslie Pack. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning* (Tahoe City, California, 1995), pp. 362–370.

[77] Littman, Michael L., Dean, Thomas L., and Kaelbling, Leslie Pack. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence* (Montreal, Québec, Canada, 1995), pp. 394–402.

[78] Lovejoy, William S. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research 28* (1991), 47–66.

[79] Lusena, Christopher, Mundhenk, Martin, and Goldsmith, Judy. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research 14* (2001), 83–103.

[80] MacKay, David J. C. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, England, 2003.

[81] Madani, Omid, Hanks, Steve, and Condon, Anne. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence 147* (2003), 5–34.

[82] Mangasarian, O. L. The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization 12* (1995), 1–7.

[83] Meuleau, Nicolas, Hauskrecht, Milos, Kim, Kee-Eung, Peshkin, Leonid, Kaelbling, Leslie, Dean, Thomas, and Boutilier, Craig. Solving very large weakly coupled markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (1998), pp. 165–172.

[84] Murphy, Kevin. A survey of POMDP solution techniques. Tech. rep., University of California at Berkeley, Berkeley, California, 2000.

[85] Nair, R., Tambe, M., Roth, M., and Yokoo, M. Communication for improving policy computation in distributed POMDPs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems* (New York City, NY, 2004), pp. 1098–1105.

[86] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., and Marsella, S. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (Acapulco, Mexico, 2003), pp. 705–711.

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

[88] Oliehoek, Frans A., Spaan, Matthijs T. J., Whiteson, Shimon, and Vlassis, Nikos. Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems* (2008), pp. 517–524.

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

[90] Papadimitriou, Christos H. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

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

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

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

[94] Parker, L. Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous Robots 12*, 3 (2002), 231–255.

[95] Parr, Ronald. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (Madison, WI, 1998), pp. 422–430.

[96] Parr, Ronald, and Russell, Stuart. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (Montréal, Québec, Canada, 1995), pp. 1088–1094.

[97] Paruchuri, Praveen, Tambe, Milind, Ordóñez, Fernando, and Kraus, Sarit. Security in multiagent systems by policy randomization. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems* (Hakodate, Japan, 2006), pp. 273–280.

[98] Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (2000), pp. 489–496.

[99] Petrik, Marek, and Zilberstein, Shlomo. Anytime coordination using separable bilinear programs. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence* (Vancouver, Canada, 2007), pp. 750–755.

[100] Pineau, Joelle, Gordon, Geoffrey, and Thrun, Sebastian. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27 (2006), 335–380.

[101] Platzman, Loren K. A feasible computational approach to infinite-horizon partially-observed Markov decision processes. Tech. rep., Georgia Institute of Technology, 1980.

[102] Poupart, Pascal, and Boutilier, Craig. VDCBPI: an approximate scalable algorithm for large scale POMDPs. In *Proceedings of Advances in Neural Information Processing Systems* (Vancouver, British Columbia, Canada, 2004), pp. 1081–1088.

[103] Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley & Sons, Inc., New York, 1994.

[104] Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley & Sons, Inc., New York, 2005.

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

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

[107] Rabinovich, Zinovi, Goldman, Claudia V., and Rosenschein, Jeffrey S. The complexity of multiagent systems: The price of silence. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems* (Melbourne, Australia, 2003), pp. 1102–1103.

[108] Radner, Roy. Team decision problems. *Annals of Mathematical Statistics 33* (1962), 857–881.

[109] Ratitch, Bohdana, and Precup, Doina. Characterizing Markov decision processes. In *Proceedings of the Thirteenth European Conference on Machine Learning* (Helsinki, Finland, 2002), pp. 391–404.

[110] Ross, Stéphane, Pineau, Jolle, Paquet, Sébastian, and Chaib-draa, Brahim. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research 32* (2008), 663–704.

[111] Russell, Stuart, and Norvig, Peter. *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2003.

[112] Schneider, Jeff, Wong, Weng-Keen, Moore, Andrew, and Riedmiller, Martin. Distributed value functions. In *Proceedings of the Sixteenth International Conference on Machine Learning* (1999), pp. 371–378.

[113] Seuken, Sven, and Zilberstein, Shlomo. Formal models and algorithms for decentralized control of multiple agents. Tech. Rep. UM-CS-2005-68, University of Massachusetts, Amherst, Department of Computer Science, 2005.

[114] Seuken, Sven, and Zilberstein, Shlomo. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence* (Vancouver, British Columbia, Canada, 2007).

[115] Seuken, Sven, and Zilberstein, Shlomo. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (Hyderabad, India, 2007), pp. 2009–2015.

[116] Seuken, Sven, and Zilberstein, Shlomo. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems 17*, 2 (2008), 190–250.

[117] Shannon, Claude E. A mathematical theory of communication. *Bell Systems Technical Journal 27* (July and October 1948), 379–423, 623–656.

[118] Shannon, Claude E., and Weaver, Warren. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[119] Shen, Jiaying, Becker, Raphen, and Lesser, Victor. Agent Interaction in Distributed MDPs and its Implications on Complexity. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems* (2006), pp. 529–536.

[120] Smallwood, Richard D., and Sondik, Edward J. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research 21*, 5 (1973), 1071–1088.

[121] Sondik, Edward J. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research 26* (1978), 282–304.

[122] Spaan, Matthijs T. J., and Melo, Francisco S. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems* (2008), pp. 525–532.

[123] Sutton, Richard S., and Barto, Andrew G. *Reinforcement Learning An Introduction*. MIT Press, Cambridge,MA, 2000.

[124] Szer, Daniel, and Charpillet, François. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning* (Porto, Portugal, 2005), pp. 389–399.

[125] Szer, Daniel, and Charpillet, François. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (Boston, Massachusetts, 2006), pp. 1233–1238.

[126] Szer, Daniel, Charpillet, François, and Zilberstein, Shlomo. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence* (Edinburgh, Scotland, 2005), pp. 576–583.

[127] Thomas, Vincent. *Proposition d'un formalisme pour la construction automatique d'interactions dans les systèmes multi-agents réactifs*. PhD thesis, Université Henri Poincaré Nancy 1, 2005.

[128] Thomas, Vincent, Bourjot, Christine, and Chevrier, Vincent. Interac-Dec-MDP: Towards the use of interactions in Dec-MDP. In *Proc. Third International Conference on Autonomous Agents and Multi-Agent Systems* (New York, NY, 2004), pp. 1450–1451.

[129] Vidal, R., Shakernia, O., Kim, J., Shim, D., and Sastry, S. Probabilistic pursuit-evasion games. *IEEE Transactions on Robotics and Automation 18*, 5 (2002), 662–669.

[130] Wagner, Thomas. *Toward Quantified, Organizationally Centered, Decision Making and Coordination*. PhD thesis, University of Massachusetts, Amherst, 1999.

[131] Wagner, Thomas, Raja, Anita, and Lesser, Victor. Modeling Uncertainty and its Implications to Sophisticated Control in TAEMS Agents. *Autonomous Agents and Multi-Agent Systems 13*, 3 (2006), 235–292.

[132] Wagner, Tom, Guralnik, Valerie, and Phelps, John. TAEMS agents: Enabling dynamic distributed supply chain management. *Journal of Electronic Commerce Research and Applications 2* (2003), 114–132.

[133] Wang, Xiao Feng, and Sandholm, Tuomas. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Proceedings of Advances in Neural Information Processing Systems* (Whistler, British Columbia, Canada, 2002), pp. 1010–1016.

[134] Wolpert, David H., Wheeler, Kevin R., and Tumer, Kagan. General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents '99)* (Seattle, Washington, 1999), pp. 77–83.

[135] Xuan, Ping, and Lesser, Victor. Multi-Agent Policies: From Centralized Ones to Decentralized Ones. *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems* (2002), 1098–1105.

[136] Xuan, Ping, Lesser, Victor, and Zilberstein, Shlomo. Communication in Multi-agent Markov Decision Processes. *University of Massachusetts Computer Science Technical Report 2000-01* (2000).

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

[138] Zhang, Nevin L., and Lee, Stephen S. Planning with partially observable Markov decision processes: Advances in exact solution methods. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (Madison, Wisconsin, 1998), pp. 523–553.

[139] Zhang, Xiaoqin, Lesser, V., and Wagner, Tom. Integrative Negotiation Among Agents Situated in Organizations. *IEEE Transactions on Systems, Man, and Cybernetics, Part C 36*, 1 (2006), 19–30.

[140] Zhang, Xiaoqin, and Lesser, Victor. Solving Negotiation Chains in Semi Cooperative Multi-Agent Systems. Tech. Rep. UMASSD-CIS-TR-2005007, University of Massachusetts, Amherst, Department of Computer Science, 2005.

[141] Zhang, Xiaoqin, Lesser, Victor, and Abdallah, Sherief. Efficient Management of Multi-Linked Negotiation Based on a Formalized Model. *Autonomous Agents and Multi-Agent Systems 10*, 2 (2005), 165–205.

[142] Zhang, XiaoQin, Lesser, Victor, and Podorozhny, Rodion. Multi-Dimensional, MultiStep Negotiation for Task Allocation in a Cooperative System. *Autonomous Agents and MultiAgent Systems 10*, 1 (2005), 5–40.

[143] Zilberstein, Shlomo, Washington, Richard, Bernstein, Daniel S., and Mouaddib, Abdel-Illah. Decision-theoretic control of planetary rovers. In *Plan-Based Control of Robotic Agents*, Michael Beetz, Ed., vol. 2466 of *Lecture Notes in AI*. Springer, 2002, pp. 270–289.