Abstracting Influences for Efficient Multiagent Coordination Under Uncertainty

by

Stefan J. Witwicki

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2011

Doctoral Committee:

Professor Edmund H. Durfee, Chair
Professor Satinder Singh Baveja
Professor Michael P. Wellman
Assistant Professor Amy Ellen Mainville Cohn

To my late grandfather, Stephen F. Witwicki.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Ed Durfee. Not only has he provided me with numerous bits of advice, which have no doubt shaped me as a researcher, but he has also given me a tremendous amount of freedom to explore and to discover for myself the research that most excites me. It is the resulting enthusiasm and confidence that has propelled me to complete this dissertation. On that note, without Ed's thoughtful suggestions and critical feedback, my work surely would not have achieved the level of quality and depth that it has. I owe him a great debt of gratitude for all of the guidance, patience, and tireless assistance he has given me when I needed each the most.

My research has also benefited greatly from my interactions with my other doctoral committee members, namely Amy Cohn, Satinder Baveja, and Michael Wellman, each of whom have shared with me their own unique perspectives and insightful critiques. The feedback that I received during my proposal defense was instrumental in redirecting the focus of my dissertation in the two years leading up to my final defense. In particular, Michael Wellman's rigorous reviews of drafts of my dissertation at various stages of its development have been extremely useful.

I would be remiss if I did not extend thanks to the administrators and support staff of the CSE department. During my time as a grad student, they have helped me to navigate program policies and Ph.D requirements, to make sense of my finances, and to arrange conference travel, aside from performing countless other supportive acts behind the scenes. I am especially grateful for the help of Dawn Freysinger, Rita Rendell, Kelly Cormier, and Cindy Watts.

I'd also like to thank my colleagues. Fellow grad students and alumni, especially Jim, Erik, Dmitri, Jonathon, Lian, Jason S., Anna, Quang, Andrew R., Gargi, and Chris P., have made the lab an engaging and productive environment, have entertained, and also inspired me. Outside of Michigan, I've been fortunate to publish within an extremely supportive research community. Conversations and e-mail exchanges with Frans Oliehoek, Janusz Marecki, Hala Mostafa, Shlomo Zilberstein, Prashant Doshi, Milind Tambe, Chris Amato, and Victor Lesser, have provided me with invaluable

perspective, perpetual encouragement, and a strong motivation to continue to share my research and to collaborate with others.

My family is deserving of more appreciation than I am capable of conveying herein. Thank you so much for your unwavering support through all of the ups and downs of my Ph.D journey. I could not have made it this far without you.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

ABSTRACT

When planning optimal decisions for teams of agents acting in uncertain domains, conventional methods explicitly coordinate all joint policy decisions and, in doing so, are inherently susceptible to the curse of dimensionality, as state, action, and observation spaces grow exponentially with the number of agents. With the goal of extending the scalability of optimal team coordination, the research presented in this dissertation examines how agents can reduce the amount of information they need to coordinate. Intuitively, to the extent that agents are *weakly coupled*, they can avoid the complexity of coordinating all decisions; they need instead only coordinate abstractions of their policies that convey their essential *influences* on each other.

In formalizing this intuition, I consider several complementary aspects of weakly-coupled problem structure, including *agent scope size*, corresponding to the number of an agent's peers whose decisions influence the agent's decisions, and *degree of influence*, corresponding to the proportion of unique influences that peers can feasibly exert. To exploit this structure, I introduce a (transition-dependent decentralized POMDP) model that efficiently decomposes into local decision models with shared state features. This context yields a novel characterization of *influences* as transition probabilities (compactly encoded using a dynamic Bayesian network). Not only is this influence representation provably sufficient for optimal coordination, but it also allows me to frame the subproblems of (1) proposing influences, (2) evaluating influences, and (3) computing optimal policies around influences as mixed-integer linear programs.

The primary advantage of working in the influence space is that there are potentially significantly fewer feasible influences than there are policies. Blending prior work on decoupled joint policy search and constraint optimization, I develop influence-space search algorithms that, for problems with a low degree of influence, compute optimal solutions orders of magnitude faster than policy-space search. When agents' influences are constrained, influence-space search also outperforms other state-of-the-art optimal solution algorithms. Moreover, by exploiting both *degree of influence* and *agent scope size*, I demonstrate scalability, substantially beyond the reach of prior optimal methods, to teams of 50 weakly-coupled transition-dependent agents.

CHAPTER 1

Introduction

A fundamental characteristic of any intelligent system, natural or artificial, is its ability to make a rational decision when faced with a set of choices. As people, our daily lives are filled with decisions, each of which involves reasoning about the consequences of potential choices. Automating the decision-making process is the topic of an extremely active area of research. The motivation is that, by outfitting the automated decision-maker (or *agent*) with computational resources and developing efficient and effective reasoning algorithms for it to make its decisions, we can realize tremendously beneficial systems. For instance, decision-support software agents can help doctors and nurses in a hospital's intensive-care unit to make quick decisions about treatment options; Agents controlling nodes of a power grid can conserve energy by forecasting consumption and deciding where to route power and when; And unmanned autonomous vehicles can deliver relief supplies and search for survivors in the wake of a natural disaster. In each of these domains, there is *uncertainty* such that an agent cannot predict the consequences of its decisions deterministically, but can instead reason over a space of probabilistic outcomes. Further, each domain involves multiple interacting agents whose individual choices may affect each others' decision consequences. Hence, achieving the most desirable outcomes requires *coordination*.

Conceptually, one might view the multi-agent coordination problem as planning joint actions for the system of agents. That is, a centralized planner formulates a joint decision rule that dictates, for any given state of the overall system, a harmonious composition of individual agent actions. In this sense, the multi-agent problem is much like controlling a single agent's multiple arms. This centralized approach, taken by much of the literature on multi-agent sequential decision-making, and as reviewed in Section 2.3.1.2, implicitly achieves coordination since all agents' actions are planned together. However, there are limitations to solving the coordination problem in this manner. From a computational standpoint, the number of composite action choices

grows exponentially with the number of agents, leading to poor scalability of methods that plan every decision jointly. From a logistical standpoint, this method requires a centralization of all problem information during the planning process, a requirement that is not reasonable for systems maintaining a separation of information (either because the infrastructure does not support transmission of all information to a central entity, or because there are portions of information that need be kept private).

Building on prior work reviewed in Section 2.3.3, the research presented in this dissertation studies an alternative approach to multi-agent coordination that decentralizes the planning process. In particular, it applies the following insight. If some individual agent decisions do not affect other agents in the system, these decisions need not be jointly reasoned about. Instead of coordinating all decisions, the agents only need to coordinate the individual choices that affect one another. This is a standard method that people use to perform joint reasoning. For instance, when scheduling a meeting, instead of describing their individual activities in full, it is common for a group to communicate windows of availability. In doing so, they create a layer of abstraction that separates the *influence* each individual has on the group from the underlying joint decisions. The decisions around the meeting can then be planned individually, thereby avoiding much of the complexity of fully-centralized reasoning (and maintaining privacy of local decision information), yet still achieving harmonious joint behavior. The focus of this dissertation is on the computational aspects of influence abstraction: the development and evaluation of representations and algorithms for coordinating agents' abstract influences.

1.1 Multiagent Coordination Under Uncertainty

The label *multiagent coordination under uncertainty* could be used to describe a vast array of problems with different assumptions studied under disparate circumstances. This thesis focuses on one particular class of coordination problems with specific properties. I begin by describing a motivating domain (in Section 1.1.1) wherein problems from this class arise, and outlining the properties that define the class (in Section 1.1.2).

1.1.1 Motivating Example

Consider a team of robots sent to explore the surface of Mars to gather scientific data autonomously for a period of months with little or no human intervention. As in past NASA missions, this team would likely include rovers situated on the planet's

surface that move about and take various sensor measurements. The International Mars Exploration Workgroup is presently exploring the use of an array of additional robots and spacecraft equipped with other technologies to be deployed in future missions (Beaty et al., 2008). For instance, the rover team could benefit from the inclusion of orbiting satellites capable of collecting real-time imaging data for map-building and rover localization. Imagine also a data processing center that safely houses a database of scientific information (including data from past missions and newly-collected data) as well as specialized hardware and software for image processing, path planning, and forecasting of environmental conditions.

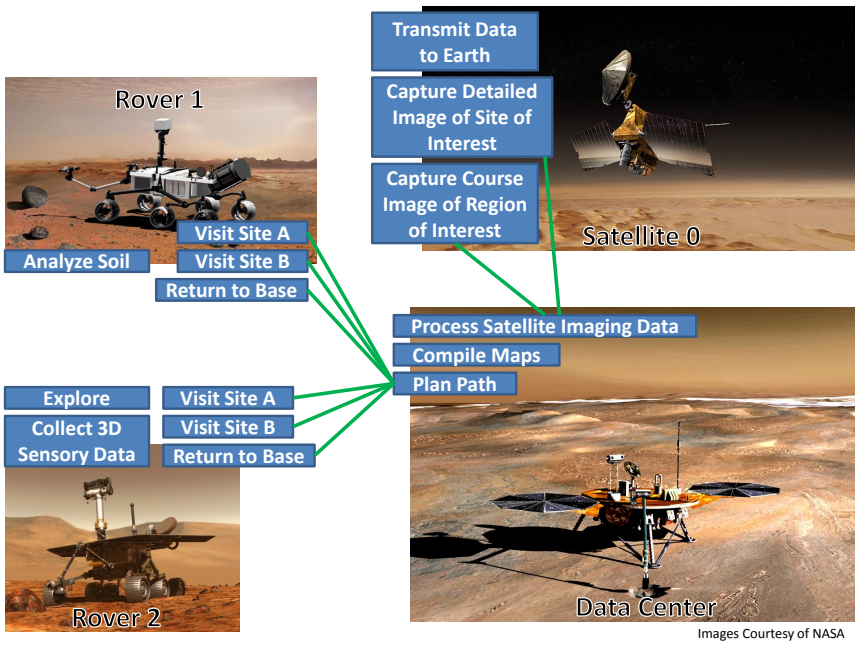


Figure 1.1: Planetary Exploration example domain.

Together these components make up a system of agents with diverse capabilities that act and interact in a shared environment. The example pictured in Figure 1.1 contains four such agents that gather scientific data by performing various high-level activities. A satellite orbits the planet taking pictures and relaying information between Earth and Mars. On the surface sits a base station that houses the data center, whose activities include analyzing imaging data and compiling the data from multiple sources into detailed surface maps. Two rovers, that are situated at the base at the start of the mission, move about the surface and visit different sites of interest. Rover 1 is equipped with tools and sensors that it can use to dig into the ground and analyze the soil. Rover 2 is designed to travel more quickly and to position itself in

a series of locations so as to compile three-dimensional sensory data. Because of its speed, it also has the capability of rapidly exploring unknown areas.

As the agents complete their activities, they fulfill various science-gathering objectives. Thus, associated with each successfully completed activity is some value, and the overall productivity of the science-gathering mission may be quantified as the summation of completed activity values. As new information is collected, new objectives may present themselves. Imagine that with each new day comes a new mission and associated objectives, which may differ depending on environmental conditions and on analyses of past missions. In each such mission, the collective goal of the agents is to maximize their expected accumulated values.

The agents can each gather and process data on their own, but benefit from interacting with one another. Interactions occur through the pursuit of interdependent activities (denoted by the lines in Figure 1.1). For instance, Rover 1 can visit site A more efficiently if the data center agent first plans a path for it. By interacting in this manner, the combination of their coordinated activities allows them to achieve complex mission objectives like locating areas with unusual geographical features, navigating rovers to those areas, analyzing soil samples, and storing the results in a geological database. It is through such a composition of individual activities that the team of agents achieves the greatest collective value, thereby making the most of its time exploring the planet.

Successful coordination in this domain requires surmounting several difficult challenges. The agents' objectives are temporally constrained with strict deadlines. In the example from Figure 1.1, the satellite is constrained in when it can take pictures of sites and areas of interest because it is orbiting about the planet. The rovers' only source of power is the sun, so each is constrained to complete activities by a deadline related to the amount of energy it has stored and the time the sun sets. There are also behavioral constraints that dictate that each agent can perform only one activity at a time. For instance, the satellite imaging agent cannot simultaneously point its camera at two different locations. In order to optimally coordinate the team's behavior and avoid wasting resources, agents' activities should be carefully planned in advance.

Furthermore, there is uncertainty in the durations of various activities. For instance, depending on the path planned for Rover 1 and on the obstacles encountered along this path, it may take a variable amount of time to reach site A. In order to maximize productivity in expectation, the agents' plans should account for the uncertainty in their actions and interactions.

1.1.2 Core Problem Properties

The Mars exploration example, as with all problems considered in this dissertation, may be characterized using the properties outlined here and in Section 1.2. This description serves to clarify the context of this thesis and to preempt any broad misconceptions. I begin by stating the fundamental properties of the class of coordination problems addressed herein.

Property 1: ***Cooperative Multiagent System.*** The problem is to formulate intelligent behavior for a team of coexisting agents that share a common goal: to maximize the group's *joint value* (of which there is some well-defined measure). In the example from Figure 1.1, joint value is measured as the expectation of the sum of qualities accrued from the successful completions of activities. For each agent, there is no notion of personal gain, and consequently, issues of fairness, truth, and incentivization **do not** arise in the development of solution methods.

Property 2: ***Model-Based Planning.*** Agents' activities involve a significant investment of resources over time, so decisions about which activities to perform for what purposes, and when, should be carefully planned in advance so as not to waste time and resources. For this purpose, there exists a (generative) model, known to the team in advance, that the agents may use to make (probabilistic) predictions and plan activities that maximize expected outcome utilities.

Property 3: ***Sequential Decision Making Under Uncertainty.*** The agents interact with their environment and with one another by performing *actions* and receiving *observations*. The model describing their behavior is a Markov decision process (described more formally in Section 2.3.1) that associates an underlying *system state* with any situation that the agents may encounter. As the agents take actions, the model can be used to make (probabilistic) predictions about *transitions* of the state and resulting observations. The model also describes the value of activities by associating a *reward* with every state and combination of agents' actions. Uncertainty in an agent's activities (such as the analysis of soil by Rover 1 in Figure 1.1) translates to uncertainty in the system state and observation outcomes accounted for in the model as transition probabilities and observation probabilities. The model also accounts for constraints on when agents can successfully execute their activities as well as for interdependencies between activities. The problem of optimal coordination then becomes deciding how each agent should act given its observations, such that the sequence of joint decisions maximizes the agents' expected accumulation of rewards

over a finite time horizon. The formulation of joint decisions is referred to as a *joint policy.*

Property 4: *Decentralized Awareness.* While executing activities, the agents do not (necessarily) have complete views of the system state, nor the actions taken by other agents. Instead each is aware of, and bases its decisions on, only the subset of information conveyed by its *local* observations. For instance, in the planetary exploration domain, each rover agent observes only portions of information relevant to its navigation of the terrain (such as a measure of its velocity and its sensor readings), but it does not observe state information related to the satellite's camera position or the other rover's sensors. Any runtime communication is modeled through agents' transitions and observations. That is, an agent may transmit information to another agent by taking an action that causes a transition of the system state, and a resulting observation seen by the receiving agent.

1.2 Problem Statement

Together, the four properties described in Section 1.1.2 are closely aligned with those of the well-established (Seuken & Zilberstein, 2008) finite-horizon Decentralized Partially-Observable Markov decision process (Dec-POMDP), which I review in detail in Section 2.3.1. Dec-POMDPs are powerful theoretical models capable of representing a rich space of agent behaviors, interaction capabilities, and team objectives. However, with their expressiveness comes a general NEXP computational complexity (Bernstein et al., 2002). This result poses a substantial barrier in applying the Dec-POMDP model practically to solve problems of significant size.

To overcome the complexity barrier, researchers have sought tractable Dec-POMDP subclasses wherein agents are limited in their interactions. For instance, there has been significant effort in developing more efficient, scalable solution methods for transition-independent problems (Becker et al., 2004a; Kumar & Zilberstein, 2009; Marecki et al., 2008; Nair et al., 2005; Varakantham et al., 2007), where agents interact by jointly affecting the reward, but have *independent* effects on state transitions (as detailed more formally in Section 2.3.2.3). Intuitively, transition-independent agents cannot affect the outcomes of each others' actions. Transition-independent problems are believed to be fundamentally less complex (Allen, 2009; Goldman & Zilberstein, 2004) than general Dec-POMDPs. Although empirical results demonstrate scalability of quality-bounded[1] solutions to teams of more than a handful of transition-independent

[1]I use the term *quality-bounded* to refer to solutions whose values are guaranteed to be within

agents (Marecki et al., 2008), the drawback of these models is that they place a fairly stringent restriction on the way that agents may interact. The inability of the agents to alter the consequences of each others' actions means that many useful interactions simply cannot be represented. For instance, we would expect that the act of the data center agent from Figure 1.1 planning a path should reduce the duration of rover 1's "visit site" activity. But this constitutes a *transition-dependent* interaction and is outside of the scope of transition-independent models.

The research presented in this dissertation endeavors to concretely define a form of transition-dependent interaction structure that can be exploited, and to develop efficient[2], scalable solution algorithms capable of exploiting it. While others have taken steps in identifying transition-dependent structure, their models either (a) have not been shown to compute solutions with guaranteed bounds on quality (Varakantham et al., 2009), (b) have have not been shown to scale three beyond agents (Becker et al., 2004a; Oliehoek et al., 2008b), or (c) impose limitations on agents' individual (noninteracting) behavior by restricting the transition or observation function (Beynier & Mouaddib, 2005; Guestrin & Gordon, 2002; Marecki & Tambe, 2009, 2007). Alternatively, this dissertation focuses on identifying useful structure in agents' interactions without restricting agents' individual behavior. To this end, I now introduce (and formalize in Chapter 3) several additional problem properties.

Property 5: *Factored Decomposability.* The system model (described in Properties 2–3) conveys a complete description of the problem, containing information regarding all agents' action consequences, but the information is explicitly decomposed into subproblem descriptions, each conveying the dynamics of a single agent's behavior. In particular, the world state is factored into (overlapping) *local state* partitions, each composed of features relevant to an individual agent. In the example from Section 1.1.1, images that a satellite is taking of one side of the planet do not factor into the immediate decisions of a rover analyzing soil on the other side of the planet. As such, independent local transition functions dictate that each agent's actions may exert immediate effects on the values of features within its local state but not those outside of its local state. Observations are similarly factored such that an agent cannot observe immediate changes to features outside of its local state. Further, the reward function is composed of local reward functions that convey the immediate benefits of each agent's actions. Moreover, the overall value of a joint policy can be computed

some nontrivial, expressible bound of the optimal value.

[2]Given the daunting complexity of problems that I address, I use the term *efficient* here and throughout this dissertation to mean relatively efficient (in comparison to the computation required by other algorithms), but not necessarily polynomial.

efficiently by evaluating some well-defined function of agents' local policy values. For instance, in the domain from Figure 1.1, the agents' local values are accumulated from their individual task completions, which in turn sum across the agents to yield the joint value for the agent team. The factored state, transition, and reward structure results in a natural decomposition of the joint decision model into local decision models (though it is important to note that the local decision models are not necessarily independent of one another due to the potentially overlapping local state partitions).

Property 6: ***Nonconcurrently-controlled Nonlocal features.*** Agents affect outcomes of each others' activities in the following manner. With the overlap of agents' local states (described in Property 5), there are some features that are directly affected by one agent but that also appear in another agent's local state. For instance, in Figure 1.1, whether or not a rover visits a site depends upon whether or not the data center agent has planned a path for it. From the rover's perspective, we refer to "path-planned" as a *nonlocal feature* because its value is altered by the actions of another agent. In turn, the change in value will allow the rover to visit the site more quickly and reliably. Through changes to nonlocal features, one agent may affect the choices and consequences of another's subsequent (but not concurrent) actions. For instance, in Figure 1.1, the consequences of a rover's actions taken *after* the data center plans a path for it may be altered, but the actions taken by the rover *while* the data center agent is planning the path are unaffected. The non-concurrence of interaction effects may limit the space of representable interactions, but it vastly simplifies our decomposition of the joint planning problem (as elaborated in Section 4.2.3).

Property 7: ***Temporal Synchronization.*** One implication of the sequential interactions described by Property 6 is that successfully coordinated agents' decisions will anticipate interactions with other agents. For instance, if a rover knows that a path will be planned for it in the near future, it can spend the interim time engaging in a short activity instead of wasting time waiting or engaging in a longer but lower-quality activity than the planned path would allow. This anticipation and resultant coordination is made possible through the use of a synchronized clock signal. It is the shared awareness of the current time that allows agents to decide when to perform certain activities with assurance that they will be well-aligned with other agents' activities. As such, time is an integral feature of the system state.

The structure identified by these additional properties is significant in that it accommodates a broad spectrum of agent interaction. At one extreme of the spectrum, agents do not interact at all, translating to a factored model of system state

(as described by Property 5) composed of independent, non-overlapping local state factors: effectively, fully-independent POMDPs. In this degenerate case, there is no need for coordination because the optimal joint policy is simply the combination of independently-planned optimal local policies. Moreover, the decisions that each agent makes cannot influence the decisions of its peers. With the addition of nonlocal features (as described in Property 6), agents begin to influence each others' decisions. The *coupling* of the system (a metric developed formally in Section 3.5) describes the degree to which agents may influence each others' decisions. The expectation is that as the degree of agent coupling increases, a greater degree of coordination is required, and the coordination problem becomes harder to solve. It is on the weakly-coupled side of the spectrum that agents should be able to compute solutions more efficiently and scale their solution algorithms to larger problems (assuming they remain weakly coupled).

The main problem that I address in this dissertation is how to solve the class of cooperative, model-based, sequential, stochastic, decentralized, decomposable, structured, temporal coordination problems outlined by Properties 1–7 in such a way that will exploit interaction structure to solve weakly-coupled problems more efficiently than strongly-coupled problems. The objective is a practical computational methodology whose usefulness over existing approaches lies in its satisfaction of the following desiderata:

- **Exploitation of weak coupling for improved performance**
  The methodology should, in principle, compute optimal solutions to the entire spectrum of coordination problems defined by Properties 1–7. Since not every problem will be computationally tractable, the methodology should exploit structure in problems that are weakly-coupled[3], so as to require less computational overhead (measured by memory requirements and computation time) for weakly-coupled problems than for strongly-coupled problems. Moreover, gradual variations in agent coupling should lead to a gradual shift in the computational overhead required to formulate solutions.

- **Scalability in the number of agents**
  Much of the literature on multi-agent coordination under uncertainty restricts consideration of models or experiments to just two agents. By scaling to more agents, solution methods become more widely applicable (to domains with

[3]Note that *weakly-coupled* is not a binary classifier. Throughout this dissertation, whenever I qualify problems or agents as "weakly-coupled", I am referring to the fact that the relative degree of coupling falls towards the weak end of the coupling spectrum.

multiple interacting decision-makers) as well as more effective in domains where more agents means more diverse capabilities (such as in the planetary exploration domain described in Section 1.1.1). In particular, the methodology should produce optimal solutions to problems with dozens of transition-dependent agents (under the assumption that the agents are sufficiently weakly-coupled, though not transition-independent).[4]

- **Flexibility of Approximation**
  Given the complexity of coordination under uncertainty, there exist many problems for which it is impractical to compute optimal solutions. Thus, it is important that the methodology produce approximate solutions according to computational restrictions. Moreover, the methodology should be amenable to different degrees of approximation, thereby providing knobs that the practitioner can turn so as to strike a desired balance between computational overhead and solution quality.

1.3 Solution Approach

To provide satisfactory solutions that fulfill the desiderata in Section 1.2, I have developed a principled framework for influence-based policy abstraction. My framework formalizes the following simple intuition. Weakly-coupled agents, who have little influence on each others' decisions, can plan more efficiently by decomposing the joint policy computation problem into (partially) decoupled subproblems: formulation of individual agent policies and coordination of abstract influences. This is in stark contrast to fully-centralized planning, which is the paradigm adopted by much of the literature for solving Dec-POMDPs (reviewed in Section 2.3.1.2). Instead of a central entity reasoning about all agents' policy decisions together, my framework gives each agent its own planning perspective through which to compute its own local policy. However, the local policy formulation problems are not completely independent since the optimal policy of one agent can depend on the decisions made by other agents. Agents account for these dependencies by conveying, and coordinating over, only their essential influences on each other.

Figure 1.2 provides a diagrammatic overview of *influence-based policy abstraction*, wherein the blocks represent different components of the solution formulation process and the arrows and lines depict information flow between the components. I introduce

[4]For a demonstration of scalability of optimal solutions to problems with 50 agents, see Section 6.6.

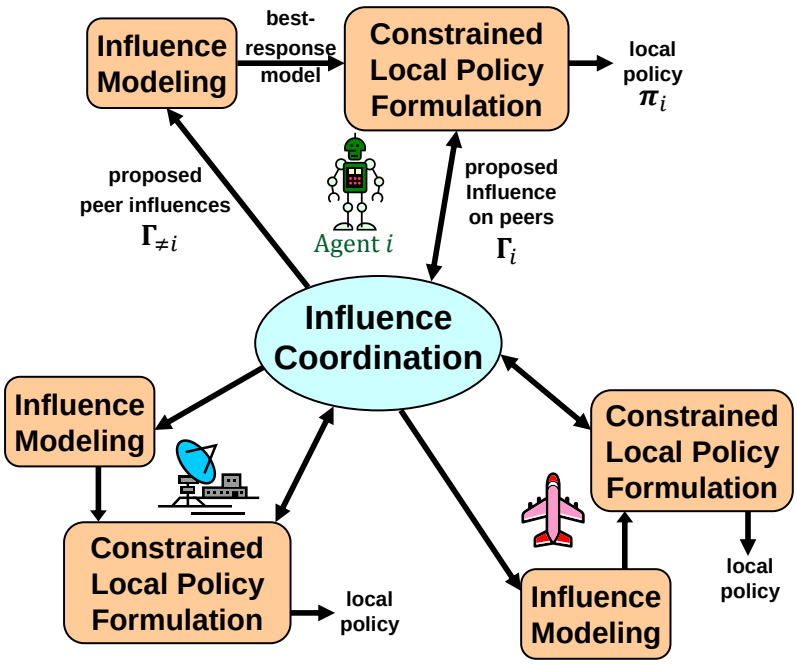


Figure 1.2: Components of Influence-Based Abstraction methodology.

each component below, describing how it fits into the overarching scheme as well as identifying the high-level research questions to which this dissertation is devoted to providing answers.

Influence Modeling. The fundamental question that ignited this body of work was: *How should agents model each other?* The system model (introduced in Properties 2–3) provides a representation of agents' joint behavior, but not in a manner that will allow an agent to efficiently infer how its decisions are impacted by those of its peers. This portion of my work develops efficient local models that account for peers' expected behaviors as they relate to the agent's own decisions.

Intrinsic to the modeling problem is the question of: *What does an agent need to know about peers' planned behavior in order to plan its own optimal local behavior in response?* Knowing all the policy decisions of all other agents would certainly suffice. However, weakly-coupled agents that interact only in the context of certain activities, and with only a subset of peers, need not model all decisions of all peers. There may be many peer decisions whose consequences do not affect an agent i. With this insight, I develop an abstraction of peer policies that I call an *influence*, referred to in Figure 1.2 as $\Gamma_{\neq i}$, which is a subset of peer policy information that conveys only the effects of peers' decisions as they relate to agent i's own decision problem. For instance, whereas

the complete policies of the team of agents in Figure 1.1 would include information about all of the activities each agent plans to pursue in every foreseeable situation, Rover 1 only cares about whether and when paths will be planned. Although influences could be represented in any number of ways, the model that I adopt in this dissertation takes the form of a probability distribution over interaction effects. For instance, the influence $\Gamma_{\text{DataCenter}}$ of the Data Center on Rover 1 would include the probability $Pr(\textit{Path-to-Site-A}|\textit{time})$ of the Data Center sending the Rover a planned path at various times over the course of the mission.

By abstracting away all the superfluous details of their policies, agents are left with succinct influences that summarize the effects of their policies on their peers. In turn, they propose these influences to their peers (as the labels of the incoming and outgoing arrows in Figure 1.2 indicate). Upon receiving the proposed influences $\Gamma_{\neq i}$ from its peers, agent i folds $\Gamma_{\neq i}$ into a local decision model (portrayed in Figure 1.2 as "influence modeling") which I refer to as a *best-response model*, from which it can compute, among other things, optimal local policies in response to peers' proposed influences.

Constrained Local Policy Formulation. Agent i can also use its best-response model to reason its own influence Γ_i on its peers (given peer influences $\Gamma_{\neq i}$). Influence Γ_i provides an abstraction that conveys expectations about the effects of agent i's policy on i's peers. Hence, using the abstraction involves translating back and fourth between the agent's policy and influence representations. For instance, the agent may propose an influence by starting with a completely-specified local policy and computing the influence that the policy exerts on its peers. More importantly, given a proposed influence, the agent must implement a local policy that delivers on the expectations conveyed by that influence. This evokes the question: *How can an agent enforce that its policy exerts a committed influence?*

Prior approaches encourage the exertion of various forms of influence through reward shaping (Mataric, 1997; Musliner et al., 2006; Varakantham et al., 2009), which injects artificial rewards (or penalties) into the agent's local decision model to encourage (or discourage) desired behavioral outcomes. Although this method could be employed to bias the agent to fulfill its committed influences, fulfillment is not guaranteed, nor is the optimality of the agent's local policy (as I prove in Chapter 5). These results have motivated me to develop a new method for influence enforcement that uses a fundamentally different strategy. Instead of biasing individual decisions to push the agent into situations where it will interact as desired, the idea is to constrain

the policy directly to enforce the committed influence.

When associating influences with policy constraints, issues of overconstrainedness arise. That is, a particular influence (or combination of influences) may not be feasible for the agent to exert by *any* local policy. Thus, it is vital that an agent be able to efficiently identify the feasible influences that it could exert. In addition to influence enforcement, the constrained policy formulation methodology that I develop in Chapter 5 also addresses the problems of checking feasibility and identifying feasible influences (without explicitly enumerating all individual policies).

Influence Coordination. The previous two components involve decentralized computation by the individual agents in the system, wherein each agent reasons about its own decisions and its own influences (on peers) as a function of potential peer influences. Using these components, the agents can avoid explicit *joint* reasoning about detailed policy decisions. Instead, they need only jointly consider their influences on one another. The "Influence Coordination" block at the center of the diagram in Figure 1.2 forms the intersection of the individual agents' decision-making problems, addressing the following question. *How can an agent team converge on a set of influences that yield an optimal joint policy.*

At a high level, I have recast the problem of policy-space search as one of *influence-space search.* The motivation is that although a weakly-coupled agent may have a very large number of policies, depending on the relative portion of its policy decisions which do not affect its peers, the agent will have a proportionally smaller number of unique influences that it can exert on its peers. In contrast to prior distributed planning approaches that work directly with policies (Marecki et al., 2008; Nair et al., 2003; Varakantham et al., 2009), approaching the problem in this manner offers its own interesting challenges. For instance, the influence space (as defined more formally in Chapter 4) is a continuous space of probability vectors. Further, since one agent's influence value changes the feasible influences of another agent, the order that the team reasons about different influences can have significant effects on the completeness and efficiency of the search algorithm.

1.4 Contributions

The primary contribution of this work is the design and evaluation of a principled methodology for multiagent coordination under uncertainty, with a focus on efficiency of solution generation and scalability to problems with many weakly-coupled agents

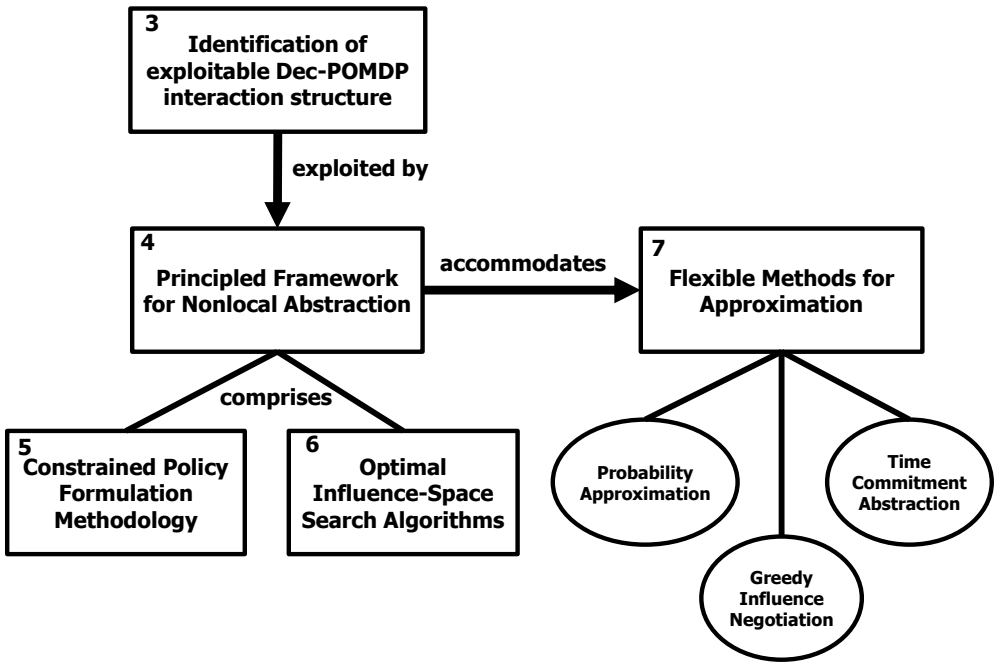


Figure 1.3: Overview of dissertation contributions, indexed by chapter.

interacting in a structured manner. Out of this effort comes definitive evidence in support of the hypothesis that coordinating using abstractions of structured interactions can afford agents significant reductions in computational complexity, thereby enabling solutions to problems that were previously thought to be intractable (Allen, 2009; Bernstein et al., 2002). An overview of the components of this thesis, indexed by chapter, is shown in Figure 1.3. The primary contributions of each component are as follows.

- *Identification of Interaction Structure Amenable to Tractable Solutions*
  Given the computational complexity (Bernstein et al., 2002) of the general Dec-POMDP where agents' interactions are unrestricted, successful Dec-POMDP applications beyond small, two-agent toy problems require exploitation of structure amenable to tractable computation of optimal or near-optimal solutions. Akin to prior work that identifies transition-independent structure through which agents affect each others' rewards (Becker et al., 2004b; Nair et al., 2005), the work presented in this dissertation identifies structure in the way that agents affect each others' transitions. The formalization of this structure (which I present in Chapter 3 as a TD-POMDP) is novel in its combination of (a) explicitly distinguishing each *nonlocal* state feature (through which an agent is influenced by a peer) from the *local* state features (that the agent controls), thereby facilitating

a natural decoupling of the joint model into local models, (b) enabling a systematic analysis and abstraction (in Chapter 4) of sequential transition-dependent interagent influences, and (c) not imposing overly-restrictive constraints on agents' local behavior nor their ability to interact. Moreover, results indicate that for weakly-coupled agents, exploiting the structure I have identified can yield exponential speedups over solution methods for general flavors of transition-dependent Dec-POMDPs, not to mention scalability to teams of many more agents. These traits make my structural model a useful candidate for researchers to extend and for practitioners to adopt.

- *Principled Framework for Nonlocal Abstraction*
  This dissertation develops a general, principled framework for abstracting agents' transition-dependent influences from their policies. The practical contribution of the framework is a novel influence model that compactly incorporates nonlocal information (abstracted from peer agents' committed policies) into a local POMDP. The conceptual contribution is the idea that, by formally characterizing this nonlocal information, an influence space emerges that is often more efficient to search than the policy space, yet is still amenable to optimal solutions. Furthermore, in developing and evaluating influence-based solution algorithms, this dissertation sheds light on the impact of nonlocal abstraction, particularly as it relates to the degree of agent coupling and the efficiency of solution computation. Knowledge of the circumstances under which influence-based policy abstraction provides the greatest computational gains can inform researchers seeking to apply such techniques.

- *Constrained Policy Formulation Methodology*
  This work extends the research of others (D'Epenoux, 1963; Dolgov & Durfee, 2005; Kallenberg, 1983) in applying linear optimization to sequential decision making. The insight is that, since the probabilistic effects that influences encode are intrinsically represented in the MDP dual linear program, agents can compute policies that directly account for the influences that they exert on their peers. In Chapter 5, I develop several flavors of (mixed-integer) linear programs for constraining agents' policies and exploring the space of possible influences. In contrast with prior approaches geared towards enforcing interacting behavior, this novel methodology enables an agent to (a) determine whether a desired influence is feasible, if so (b) compute the optimal local policy that is constrained to exert the influence, and (c) completely avoid any tuning of parameters associated

with influence enforcement. More generally, the contribution to the agent-based optimization community is an arsenal of constrained policy formulation techniques that may be adapted and extended to solve other decision-making problems involving behavioral constraints.

- *Optimal Influence-Space Search Algorithms*
  In Chapter 6, I develop and evaluate efficient algorithms that employ the abstraction models of influence from Chapter 4 and constrained policy formulation techniques from Chapter 5. The novelty of my algorithms is their use of influence-based policy abstraction to compute *optimal* solutions for a general class of transition-dependent problems. In addition to a depth-first search algorithm, I extend and apply a method from constraint optimization to exploit graphical structure in influences, which allows scaling of optimal solution computation to teams of more than a handful of weakly-coupled agents. These algorithms, by themselves, constitute a meaningful contribution to the Dec-POMDP community because they demonstrably advance the state of the art in efficiency and agent scalability for classes of commonly-studied transition-dependent problems. Additionally, this dissertation contributes an empirical evaluation of the benefits and limitations of optimal influence-space search that may serve as a guide for researchers and developers so that they may make informed decisions about the suitability of influence-space search to the problems that they address.

- *Flexible Influence Approximation*
  Lastly, this dissertation outlines several extensions of the influence-based abstraction methodology for coping with larger problems whose optimal solutions are intractable to compute. In Chapter 7, I develop three different techniques that agents may employ to trade solution quality for computational efficiency. The first technique approximates the space of influence probabilities, ignoring influences whose settings are close to those already considered. The second technique approximates the structure of agents' influence encodings, thereby applying an extra layer of abstraction to reduce the number of parameters with which influences are conveyed. The third technique searches the space of influences greedily rather than exhaustively for significantly faster convergence on approximate solutions. Although less systematic than some of my earlier analyses, my empirical evaluations of these techniques contribute evidence of the effectiveness of these variations of my methodology at reducing computation while still achieving near-optimal solutions.

CHAPTER 2

Background

In order to gauge the scope and magnitude of this dissertation's contributions, we must first consider the prior work that addresses related problems. Research in the field of automated decision making has grown far too vast to recount in its entirety. Instead I review the most closely related paradigms, problem formalisms, and solution approaches. This chapter provides the reader with a brief survey of background work to better understand the foundation on which my models and methodologies are built. It motivates the remainder of this dissertation by noting the inadequacies of these prior approaches in relation to the desiderata that I outlined in Section 1.2.

2.1 Overview

I have divided the background material into *single-agent* and *multiagent* decision making research, reviewed in Sections 2.2 and 2.3 respectively. In Section 2.2, I begin by describing the Markov Decision Process (MDP) and its partially-observable extension (the POMDP). In addition to forming the basis for the *multiagent* models described in Section 2.3, single-agent (PO)MDPs are employed by my solution approach for the *local* portions of agents' planning (referred to in Figure 1.2 as "constrained local policy formulation"). As such, I also give an overview of MDP and POMDP planning algorithms, digging into the details of those that I extend in later chapters, and review the computational complexities of optimal MDP and POMDP planning. I also give a brief survey of foundational single-agent work in *decomposition* and *abstraction* and its relationship to my methodology.

In Section 2.3, I review the Decentralized POMDP (Dec-POMDP), a general extension of the POMDP for teams of cooperative agents. After presenting the Dec-POMDP formalism, the computational complexity of optimal Dec-POMDP planning, and an overview of general-purpose planning algorithms, I focus the remainder of this

chapter on work that exploits particular problem structure to improve efficiency and scalability. In this vein, I give an overview of structural restrictions that researchers have imposed, each of which has yielded planning algorithms with significant computational advantages over general-purpose algorithms, but whose scopes are limited to problems in specialized Dec-POMDP subclasses. In particular, I characterize those algorithms that exploit weakly-coupled[1] problem structure, wherein agents' limited interactions engender an efficient decoupling of the centralized joint policy formulation into largely-decentralized local planning problems.

My review of Dec-POMDP algorithms and subclasses in Section 2.3 exposes a division of research thrusts in the field of Dec-POMDP planning, which I elaborate in Section 2.4. On one side, there is work that remains general, assuming no particular structure, so as to develop the most broadly-applicable Dec-POMDP theory and algorithms. On the other side, there are those approaches that intentionally avoid generality in favor of exploitability, each restricting consideration to a subclass that exhibits particular structure amenable to efficient and scalable algorithms. Few approaches are both generally applicable and efficient and scalable (subject to the degree to which exploitable structure is present). In particular, there are no quality-bounded algorithms for transition-dependent problems that have achieved scalability beyond three agents.

2.2 Single-Agent Sequential Decision Making

Let us begin by considering a single agent inhabiting an environment, which we call the *world*. Over the course of the agent's lifetime, it encounters situations, which we call *states*, wherein it must decide what *action* to take. The outcome of each action an agent takes is a *transition* into a new state (according to the dynamics of the world), where the agent faces another decision. With each transition is associated an intrinsic value, called a *reward*, that depends on the agent's state and action. As it makes decision after decision, the agent's objective is to maximize some function of its rewards received. Such is the basic premise of *sequential decision making*, which encompasses a wide variety of problems in Artificial Intelligence (Littman, 1996).

In this dissertation, I adopt the Markov Decision Process (Bellman, 1957), along with its later-described extensions, as a general model for sequential decision making. MDP planning can be thought of as a generalization of *classical planning* (Fikes & Nilsson, 1971) (where the problem is finding a sequence of actions with deterministic

[1] I develop a more concrete definition of weak coupling later Section 3.5.

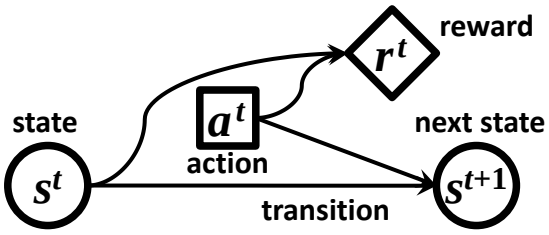


Figure 2.1: MDP *state*, *action*, *transition*, and *reward* dynamics.

transitions that lead to a goal state), adding transition uncertainty as well as reward. It can also be thought of as a generalization of (discrete-time) *scheduling* (Pinedo, 2008) (where the problem is timing an agent's decisions). In the subsections that follow, I review the MDP formalism, an extension for partially-observable worlds, algorithms, complexity results, and advanced solution strategies rooted in the single-agent MDP literature that I use in my multiagent methodology.

2.2.1 Markov Decision Processes

I begin with a very brief introduction to MDPs, citing just a few results from other authors' more detailed treatments (Bellman, 1957; Kallenberg, 1983; Papadimitriou & Tsitsiklis, 1987; Puterman, 1994; Sutton & Barto, 1998). A **single-agent MDP** may be described by a 4-tuple $\langle S, A, P, R \rangle$ whose contents are as follows:

- S is a finite set of world states, called the *state space*, over which there is a probability distribution $\boldsymbol{\alpha}$ that specifies the probability that the agent will start in any given state $s^0 \in S$.
- A is a finite set of actions, called the *action space*, such that for each state s, a specified subset $A_s \subseteq A$ of actions are available for the agent to perform.
- The *transition function* $P : S \times A \times S \mapsto [0, 1]$ specifies the probability, denoted $P(s^{t+1}|s^t, a^t)$, of the agent transitioning into state s^{t+1} given that it takes action $a^t \in A$ in state $s^t \in S$.
- The *reward function*[2] $R : S \times A \mapsto \mathbb{R}$ defines a local reward, denoted $r^t = R(s^t, a^t)$, ascribed to the action a^t taken in state s^t.

[2]In some other work, the reward function r^t depends upon the previous state s^{t-1}, action a^{t-1}, and resulting state s^t. This is simply a different convention than the one I present here, and both are equally expressive (one not any more or less general than the other).

In the above treatment, superscripts t and $t+1$ denote any two successive decision steps, which are depicted graphically in Figure 2.1 as a two-stage Dynamic Bayesian Network (DBN) (Guestrin et al., 2003; Koller & Friedman, 2009), showing the dependencies among the components of the MDP model. We will assume that all of the functional components of the model are stationary (returning the same value regardless of the particular value of t). As depicted in the DBN, the agent's next state s^{t+1} depends only upon its latest state s^t and latest action a^t. The actions the agent took or states that the agent encountered prior to arriving in state s^t cannot affect s^{t+1}. This conditional independence of the future from the past conditioned on the present is called the *Markov property*, and is what makes the MDP *Markovian*.

Example 2.1. Consider that a rover, shown in Figure 2.2, has three activities that it can pursue as it explores a newly-visited portion of the Martian surface. It can construct a map of the area using a lower-level mapping algorithm which, due to the unknown complexity of the terrain, it estimates will take 1, 2, or 3 hours with equal probability. It can also excavate, which involves drilling into the surface for approximately 1 hour and collecting various samples of rock and dirt. However, with a small probability (0.1), the excavation will be unsuccessful due to equipment malfunction or if the surface is too rocky. If successful, the rover can bring the excavated samples back to base for analysis, which will take approximately 2 hours.

Borrowing from the TÆMS modeling language (Decker, 1996), Figure 2.2 describes these activities as **tasks** with probability distributions ("Pr") over **duration** ("D"), and also over **quality** ("Q"), which measures the relative value of completing each activity. For instance, the *map area* task is valued twice as highly as the *excavate* task but may take the rover longer to complete. As described, the durations represent hours of execution. Additionally, each task has a **window** of feasibility. For instance, the rover can successfully map the area for the next 4 hours, during which time the sunlight is ideal, and after which time the rover will automatically stop mapping whether or not the map is complete. The possible **failure** of a task is indicated with outcomes quality 0. The rover's completion of the *excavate* task with a positive quality (indicating that the rover has collected samples) **enables** the rover to perform its *analyze samples* task.

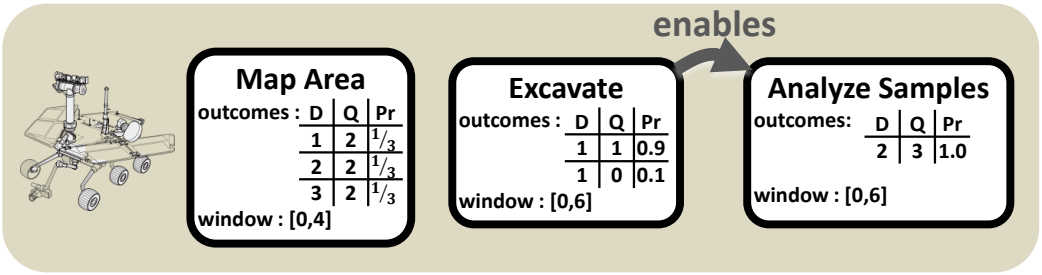


Figure 2.2: A simple example of a planning problem faced by a Mars rover.

Once the rover starts one of its tasks, it cannot stop until the task finishes (either completing with positive quality or failing), and the rover cannot restart a task which has previously failed. The rover's objective is to maximize its accumulation of qualities of the tasks it completes over the next 6 hours.

We can represent the rover's decision problem using the MDP shown in Figure 2.3, whose states model the time and task status information. At the start, the agent has not started any of the three tasks and the time is 0. Each hour, the rover acts by either beginning one of its tasks, continuing a task, or idling. Figure 2.3 shows two steps of actions and transitions. In each state, the available actions correspond to those that are allowed given task statuses and window constraints. For instance, the rover cannot analyze samples until reaching a state in which the "Excavate" task has completed. Figure 2.1 also shows the *terminal states*, whose outgoing transitions are assigned positive *rewards* equal to the sum of all completed task qualities.

2.2.1.1 Values and Policies

In addition to the reward function $R()$, which specifies the immediate reward assigned to a particular state s^t and action a^t, we can also consider the long-term value, or *expected utility*, of taking a^t in s^t. Expected utility in an MDP is typically defined, with function $U^*(s^t, a^t)$, as the maximal expected discounted reward, written recursively as:

$$U^*(s^t, a^t) = R(s^t, a^t) + \gamma \cdot \sum_{s^{t+1} \in S} \left[P(s^{t+1}|s^t, a^t) \cdot \max_{a^{t+1} \in A} U^*(s^{t+1}, a^{t+1}) \right] \tag{2.1}$$

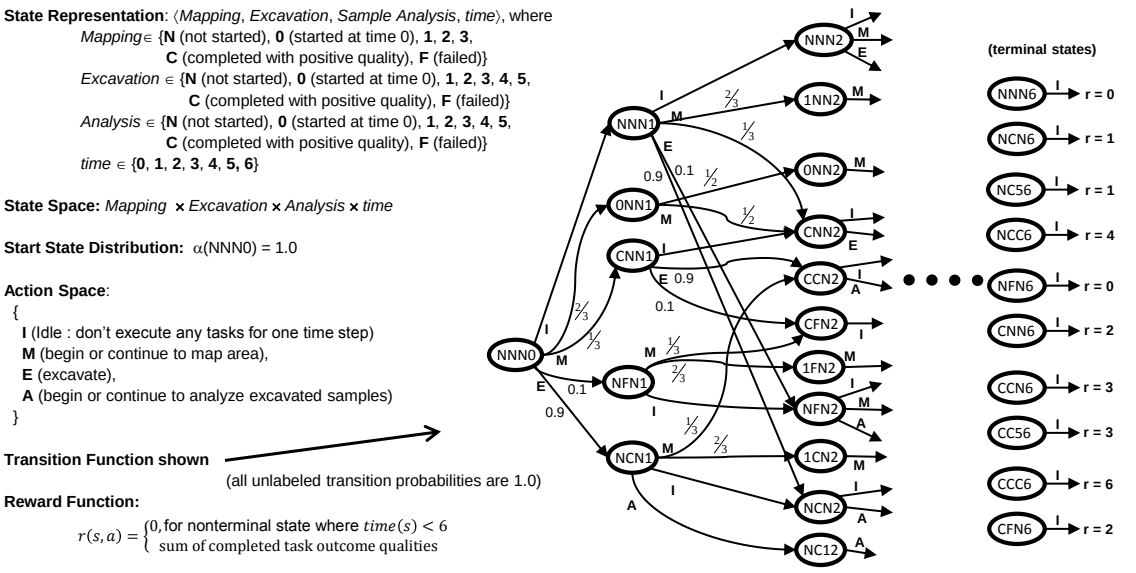


Figure 2.3: The MDP for the rover in Example 2.1.

where the first term represents immediate reward, $\gamma \in [0, 1]$ denotes the discount factor, and the summation computes the expected future reward given the best action is chosen in every subsequent state. Equation 2.1, which is commonly referred to as the *Bellman equation* (Russell et al., 1996), presents the most general notion of expected utility, where future rewards are discounted over a potentially infinite-length sequence of decisions. In this dissertation, I focus on *finite horizon* problems, wherein the agent's objective is to maximize is accumulation of rewards within finite mission deadlines. In this case, rewards are *undiscounted* ($\gamma = 1$).

An agent's behavior is prescribed by an **MDP policy** π, which encodes how the agent should behave in each world state. In general, an MDP agent may use a *randomized* policy, which maps each state to a probability distributions over randomly-selected actions ($\pi : S \times A \mapsto [0, 1]$). However, in this dissertation, I assume that each agent adopts a *deterministic policy* $\pi : S \mapsto A$, selecting a single unique action, denoted $a^t = \pi(s^t)$, for each state. For any MDP problem, there exists at least one deterministic policy that is just as good as any stochastic policy (subject to the value function $V()$ I define below), so there is no loss in solution quality associated with restricting attention to deterministic policies (Puterman, 1994).

When following a (deterministic) policy π, an agent's expected utility of entering

a given state s^t is defined (using the notation from Equation 2.1) as:

$$U_\pi(s^t) = R(s^t, \pi(s^t)) + \gamma \cdot \sum_{s^{t+1} \in S} \left[P(s^{t+1}|s^t, \pi(s^t)) U_\pi(s^t) \right] \tag{2.2}$$

Overall, the **value** $V(\pi)$ of a policy π is the expected utility of following π from the probabilistic distribution over initial states $\boldsymbol{\alpha}$ specified in the MDP description:

$$V(\pi) = \sum_{s^0 \in S} \left[\alpha(s^0) U_\pi(s^0) \right] \tag{2.3}$$

where $\alpha(s^0)$ is the probability of starting in state s^0.

MDP planning is the problem of, given a complete description of the MDP, finding the **optimal policy** π^*, whose value is greatest:

$$\pi^* = \arg\max_{\pi \in \Pi} V(\pi) \tag{2.4}$$

In Equation 2.4, Π denotes the agent's *policy space*. The optimal policy π^* is also referred to as the *solution* to the MDP. Similarly, *solving an MDP* refers to the process of computing π^*.

2.2.1.2 Solution Algorithms

Aside from simple enumeration of the policy space (as implied by the arg max in Equation 2.4), a variety of more efficient solution methods are commonly used. For instance, *policy iteration* and *value iteration* apply variations of the Bellman equation (Eq. 2.1) to iteratively converge on an optimal policy and an accurate optimal value function, respectively (Russell et al., 1996). In this dissertation, I make use of a Linear Programming (LP) approach (D'Epenoux, 1963; Kallenberg, 1983), which frames an MDP planning problem as the following linear optimization problem:

$$\begin{aligned} &\max_{\boldsymbol{x}} \sum_{s \in S} \sum_{a \in A} x(s,a) R(s,a) \\ &\quad \left| \begin{array}{l} \forall s^{t+1} \in S, \sum_{a^{t+1} \in A} x(s^{t+1}, a^{t+1}) - \gamma \sum_{s^t \in S} \sum_{a^t \in A} x(s^t, a^t) P\left(s^{t+1}|s^t, a^t\right) = \alpha(s^{t+1}) \\ \forall s \in S, \forall a \in A, x(s,a) \geq 0 \end{array} \right. \end{aligned} \tag{2.5}$$

where the vector $\boldsymbol{x}$ of variables $\{x(s,a), \forall s \in S, \forall a \in A\}$, often called the *occupation measures*, denotes the total expected discounted number of times action a is performed

in state s. Upon solving this LP, we can straightforwardly compute the optimal policy π^* from the computed optimal occupation measures:

$$\pi^*(s,a) = \frac{x^*(s,a)}{\sum_{a' \in A} x(s,a')} \tag{2.6}$$

If the LP algorithm *simplex* is used to solve the LP in Equation 2.5, it is guaranteed to return a solution $\boldsymbol{x}^*$ corresponding to *deterministic* policy as long as the components of $\boldsymbol{\alpha}$ are nonzero (Dolgov & Durfee, 2006). For MDPs wherein $\boldsymbol{\alpha}$ contains zeros, and for use of the MDP solution methodology with other LP algorithms, additional constraints can be introduced into Equation 2.5 to guarantee that a deterministic policy is returned (as I develop in Chapter 5).

The value $V(\pi^*)$ of the optimal policy found by solving the LP from Equation 2.5 is simply the value of the objective function, which is equal to the dot product of the occupation measures $\boldsymbol{x}^*$ and the vector of rewards specified by the MDP reward function $R()$:

$$V(\pi^*) = \sum_{s \in S} \sum_{a \in A} x^*(s,a) R(s,a) \tag{2.7}$$

2.2.2 Partially-Observable MDPs

The MDP model reviewed above assumes the agent is able to sense its true state of the world upon taking an action. For instance, the rover from Example 2.1 knows whether or not its excavation has succeeded or failed. The partially-observably Markov Decision Process (POMDP) relaxes this assumption, instead dictating that the agents receive *observations* that are perhaps distinct from the true world state.

Example 2.2. Consider that the rover from Example 2.1 models its excavation activity using a lower-level decision process, in order to decide how many holes to drill and what samples to keep. In particular, the rover is looking for white-colored rocks (which indicate a desirable chemical composition). There may *or* may not be any white rocks in the ground at the dig site. Let us use a boolean feature, **white-rocks-exist**, to represent the existence of white rocks at the dig site. In the rover's excavation, *white-rocks-exist* is an important, but *partially-observable*

state feature. That is, the rover cannot see white rocks buried below the surface until after digging them out. Moreover, if the rover drills a hole and does not see any white rocks, that does not mean that *white-rocks-exist=false*, since there may be white rocks buried just a few inches away. Instead, the rover receives an *observation*, *no-white-rocks-in-hole*, that is correlated with the state *white-rocks-exist=false*, but not necessarily equal to the true state value.

Partial observability makes the rover's decision of whether or not to drill a second hole, and then a third hole, and then a fourth hole, nontrivial. For instance, the rover may be better off spending its time collecting other-colored samples from the first hole than digging additional holes, depending on the relative values of the various rock samples.

Formally, a single-agent finite-horizon **POMDP** is may be described by tuple $\langle S, A, P, R, \Omega, O, T \rangle$ whose contents are as follows:

- S is the *state space*, A is the *action space*, $P()$ is the *transition function*, and $R()$ is the *reward function*, exactly as they were defined in the fully-observable MDP (Sec 2.2.1).
- Ω is a finite set of *observations*, such that the agent receives an observation $o \in \Omega$ with every transition that it makes.
- $O : A \times S \times \Omega \mapsto \mathbb{R}$ is the *observation function*, specifying the the probability, denoted $O\left(o^{t+1} | a^t, s^{t+1}\right)$, that the agent receives observation $o^{t+1} \in \Omega$ after taking action $a^t \in A$ and arriving in state $s^{t+1} \in S$.
- $T \in \mathbb{N}$ is the finite *time horizon* of execution, specifying that the agent faces decisions at (discrete) *time steps* $\langle 0, 1, \ldots, T \rangle$.

Figure 2.4 shows a DBN depicting the graphical relationships among the POMDP variables, indicating that observations depend solely upon current state and latest action. Notice that the POMDP specification above has also extended the MDP specification with a time horizon T component (in addition to the observation-related components). Without T, we would have an *infinite-horizon* decision problem. Although the infinite horizon POMDP is well defined, researchers have found little use for it due to its *undecidability* (as I describe more formally in Section 2.2.3). Similarly, in this dissertation, I restrict consideration to *finite-horizon* problems.

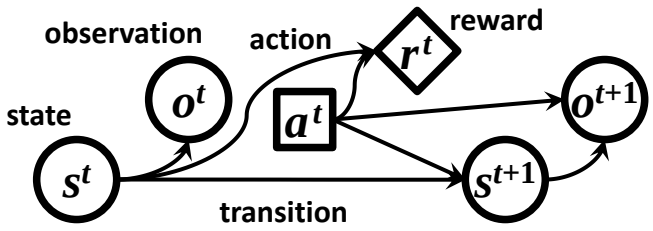


Figure 2.4: POMDP *state*, *action*, *transition*, *observation*, and *reward* dynamics.

2.2.2.1 Histories of Observations

Although the POMDP state transition dynamics are *Markovian*, as clearly depicted in Figure 2.4, a POMDP agent does not necessarily know its true state at any given decision step, and so it must rely on present *and past* observations to make the most informed decisions. It can no longer safely forget the past (as it could in the case of an MDP). If it did it would be throwing away potentially useful information about the present (with which to disambiguate the present state).

Example 2.2 (continued). Assume that the rover drills one hole and observes no white rocks, then drills another, again observing no white rocks, then a third, still observing no white rocks. Should the rover dig again? If it bases its decision on only the latest observation, forgetting all of its past failures to find white rocks, it is likely to keep on trying and keep on failing. However, by considering all observed evidence, $\langle$*no-white-rocks-in-hole*, *no-white-rocks-in-hole*, *no-white-rocks-in-hole*$\rangle$, it can make a better informed decision about whether or not to try again or to pursue a different-colored rock.

The information that a POMDP agent collects about the world from time steps 0 to t is captured by its *history of observations* $\vec{o}^t = \langle o_1, ..., o_t \rangle \in (\Omega)^t$. Since, in general, an agent's optimal decisions may depend on past observations, a (deterministic) **POMDP policy** $\pi : (\Omega)^T \mapsto A$ is a mapping of complete observation history[3] to action, prescribing an action $a^t = \pi(\vec{o}^t)$ for every possible sequence of observations $\vec{o}^t$. The difficulty of such a representation is that it grows with every additional decision step, making a POMDP agent's policy space exponential in the time horizon T.

[3]In the case that an agent is following a randomized policy, it must also base its decisions on its history of actions. However, in this dissertation, I assume that the agent's policy is deterministic, implying that the agent can always recover its action history from its observation history.

2.2.2.2 Belief State

Researchers have developed a useful strategy for combating the exponentially-growing policy representation, which I briefly review now. It turns out that an agent *can* forget past observations as long as it maintains a *belief state* that encodes sufficient information about past observations and actions to make the best possible predictions about future transitions. In a completely-observable single-agent MDP, the current world state constitutes a sufficient belief state. In a POMDP, the probability distribution over all possible current world states is sufficient (Smallwood & Sondik, 1973). Thus, the POMDP belief state is a vector $\boldsymbol{b}$, containing a component for each world state:

$$b^t\left(s^t\right) = Pr\left(s^t | \vec{a}_j^{t-1}, \vec{o}_j^t\right), \forall s^t \in S \tag{2.8}$$

At the start of execution, before an agent takes a single action or receives a single observation, the initial belief state is equal to the probability distribution over initial world states $\boldsymbol{b}^0 = \boldsymbol{\alpha}$ (where $\boldsymbol{\alpha}$ is the probability distribution over start states specified in the POMDP description from Section 2.2.1). As the agent takes actions and receives observations, it updates each component of its belief state using the following *belief-state estimator*:

$$b^{t+1}\left(s^{t+1}\right) = \frac{Pr(o^{t+1}|a^t, s^{t+1}) \sum_{s^t} Pr\left(s^{t+1}|s^t, a^t\right) b^t\left(s_j^t\right)}{\text{a normalizing factor}} \tag{2.9}$$

where $b^t\left(s_j^t\right)$ is a component from the latest belief state, as derived by Smallwood & Sondik (1973). Although the POMDP belief state representation $\boldsymbol{b}$ is still larger than the MDP state in that it requires a probability value for every world state, it takes only constant space to maintain as the agent makes more and more decisions. In contrast, the history of observation grows with each new decision.

Example 2.2 (continued). Smallwood & Sondik's theory dictates that the rover can forget past observations as long as it maintains a value of $b^t = Pr(\textit{white-rocks-exist}^t = \textit{true})$. Let $O(\textit{no-white-rocks-in-hole} \mid \textit{white-rocks-exist} = \textit{true}) = 0.5$, indicating that if there exist white rocks at the dig site, the rover is just as likely not to find them in a given hole than it is to find them. Let the initial state distribution (i.e., the prior probability of white rocks existing) $\alpha = 0.5$. Table 2.1 below shows the world state, actions, and observations of the previously-described execution trace, along with the rover's estimated belief state.

step t	true state s^t	observation o^t	belief state b^t	action a^t
0	*white-rocks-exist=false*	—	$Pr(\textit{white-rocks-exist} = \textit{true}) = 0.5$	drill-hole
1	*white-rocks-exist=false*	*no-white-rocks*	$Pr(\textit{white-rocks-exist} = \textit{true}) = \frac{1}{3}$	drill-hole
2	*white-rocks-exist=false*	*no-white-rocks*	$Pr(\textit{white-rocks-exist} = \textit{true}) = 0.2$	drill-hole
3	*white-rocks-exist=false*	*no-white-rocks*	$Pr(\textit{white-rocks-exist} = \textit{true}) = \frac{1}{9}$	drill-hole

Table 2.1: A sample execution trace for the rover in Example 2.2

Another benefit of the POMDP belief state is that its dynamics are Markovian. Moreover, the space of all reachable POMDP belief states and their transition dynamics (which are simply derived from Equation 2.9) together define a belief-state MDP, whose solution is equivalent to that of the POMDP. In effect, the belief state representation reduces the POMDP to a complicated but normal MDP. As such, a common approach for solving a POMDP is to work entirely in the belief-state space, thereby solving the equivalent belief-state MDP (Cassandra et al., 1996; Kaelbling et al., 1998; Littman et al., 1995a). In a later chapter (Section 4.2), I will derive a more complicated belief state representation that incorporates information about other agents in a multiagent system, but that is based upon the same principles of sufficiency and MDP reducibility.

2.2.3 Complexity of Single-Agent Planning

I now briefly review some complexity results and their implications on the computation required by MDP and POMDP solution algorithms, foregoing the foundational background of complexity theory such as *Turing machines* and *complexity classes* (but I refer the reader to a textbook on complexity theory (e.g., Papadimitriou, 1994) for a deeper understanding of the results in this section).

Completely-observable single agent MDPs have been proven to be *polynomial*, implying that there exist algorithms for which, in the worst case, the time and space taken to compute an optimal MDP policy is a polynomial function of the size of the MDP problem description (Littman et al., 1995b; Papadimitriou & Tsitsiklis, 1987). Formally, the size of the problem description is the amount of space required to store the complete model specification (i.e., the tuple $\langle S, A, P, R\rangle$ in the case of the MDP, and $\langle S, A, P, R, \Omega, O, T\rangle$ in the case of the POMDP). If the action space is significantly smaller than the state space ($\|A\| \ll \|S\|$), then the MDP can be solved in time and space polynomial in $\|S\|$. In particular, the LP methodology that I reviewed in Section 2.2.1.2 admits polynomial-time solutions.[4]

[4]Despite this result, polynomial time LP algorithms are rarely used to solve MDPs. Rather, the worst-case-exponential *simplex* algorithm has been shown empirically to yield better average case

Not surprisingly, theoretical results suggest that, in general, POMDPs are harder to solve than MDPs. Papadimitriou & Tsitsiklis (1987) have proven the finite-horizon POMDP to be in a higher complexity class, *PSPACE*, for which there are believed to be at best *exponential* time (and polynomial space) solution algorithms (Allen, 2009; Papadimitriou, 1994). Consequently, under the assumption that $\{\|A\| \ll \|S\|$, $\|\Omega\| \ll \|S\|$, and $T \ll \|S\|\}$, it is believed that the worst-case computation time of computing an optimal (finite-horizon) POMDP solution is *exponential*, denoted $EXP(\|S\|)$, in the size of the state space $\|S\|$. Lusena et al. (2001) have proven an even stronger result, indicating that the time required to compute ϵ-approximately optimal (finite-horizon) POMDP policies (whose values are within ϵ of the optimal value) is also *exponential*.

In the case of infinite-horizon POMDPs (wherein the time horizon T is unbounded), the problem of determining whether or not a given policy is optimal is *undecidable* (Madani et al., 1999). The implication is that no general technique exists for computing an optimal policy to an infinite-horizon POMDP.

2.2.4 Decomposition and Abstraction

Although this thesis is concerned with coordination in systems of multiple agents, some of its central themes are routed in single-agent research. In particular, *decomposition* and *abstraction* techniques have proven to be effective for improving the efficiency of single-agent planning and reasoning. I now briefly review these concepts and provide citations to pioneering work in decomposing and abstracting single-agent sequential decision making.

Decomposition breaks one large problem into smaller, more manageable problems. For example, Singh & Cohn (1998) study MDP models composed of concurrent subprocesses with interdependent actions that can be solved in parallel and merged to construct optimal global solutions. Meuleau et al. (1998) develop a method for decomposing very large MDPs into independent subprocesses coupled by resource constraints. Both of these works compute solutions efficiently by exploiting *factored* structure. That is, they isolate subsets of actions and portions of the world state that may be treated independently of one another. There has since been a lot of work in developing efficient solution algorithms for factored MDPs (Boutilier et al., 1999; Guestrin et al., 2003; Kearns & Koller, 1999; Poupart et al., 2002). In the multiagent methodology that I present in this dissertation, I too take advantage of

performance, as detailed by Littman et al. (1995b).

factored structure so as to decouple each agent's *local* decision model from the joint decision model.

Researchers have also reduced single-agent MDP complexity by solving smaller (often approximate) models with knowledge or action representations abstracted from the original models. For example, Dean (along with others) explores reduction of large state and action spaces through heuristic prioritization (Boutilier et al., 1997; Dean & Lin, 1995) and aggregation (Dean & Givan, 1997; Dean et al., 1998; Dearden & Boutilier, 1997). There is also a large body of literature on hierarchical representations that treat individual actions and states as abstractions of sequences of primitive actions and state transition (Barto & Mahadevan, 2003; Jonsson & Barto, 2005; Osentoski & Mahadevan, 2007; Sutton et al., 1999). I take a similar approach, abstracting expected (nonnonlocal) transition sequences of an agent's peers as local transitions in its local model.

2.3 Multiagent Coordination

Propelled by the momentum gained and results achieved in single-agent sequential decision making, researchers have developed a variety of multiagent extensions to the MDP and POMDP models. Some extensions, such as the Partially-Observable Stochastic Game (POSG) studied by Hansen et al. (2004), and Gmytrasiewicz & Doshi's Interactive POMDP (I-POMDP), represent agents as having their own objectives and intentions, making these models appropriate for systems of self-interested (non-cooperative) agents. In this dissertation, I restrict consideration to teams of cooperative agents who share a common objective. As I cite in Section 2.3.1.1, the problem of computing optimal behavior for cooperative agents under transition and observation uncertainty is extremely challenging in and of itself.

Here I review the Decentralized POMDP (Dec-POMDP)[5] as studied by Bernstein et al. (2000), which has emerged as the most popular and the most general POMDP extension for cooperative agents. Before delving into the details of the Dec-POMDP, I now give brief mention of some other general models and their relationships to the Dec-POMDP. Another name for the Dec-POMDP is the Partially Observable Identical Payoff Stochastic Game (POIPSG), which was introduced by Peshkin et al. (2000) in the same year as the Dec-POMDP's inception. Additionally, the Multiagent Team Decision Problem (MTDP) (Pynadath & Tambe, 2002) has been proven to

[5]Here and throughout this dissertation, I maintain the convention of abbreviating "Decentralized–" with "Dec-" (using a lowercase e and c). Note that I am referring to the same models that appear in some other work abbreviated as "DEC–" (e.g. "DEC-POMDP").

be equivalent to the Dec-POMDP (Seuken & Zilberstein, 2008), and differs only in its representation of policies[6]. Researchers have also developed extensions to the Dec-POMDP and the MTDP models, called the Dec-POMDP-Com (Goldman & Zilberstein, 2003) and MTDP-Com (Pynadath & Tambe, 2002), that represent communication among agents distinctly from agents' actions and observations. Both of these extensions have been shown to be no more general (in representational power) than the Dec-POMDP (Seuken & Zilberstein, 2008).

After reviewing the general Dec-POMDP formalism, general Dec-POMDP algorithms, and Dec-POMDP complexity, I describe a variety of other more restrictive models that may be considered as *Dec-POMDP subclasses*. These include, among many others, the Multi-agent Markov Decision Process (MMDP) as described by Boutilier (1996), and the Dec-MDP as described by Bernstein et al. (2002), both of which impose restrictions on agents' observations. I go on (in Sections 2.3.2–2.3.3) to characterize these models by their restrictions as well as the problem structure that their respective solution algorithms are designed to exploit. For other characterizations of Dec-POMDP models, subclasses, and algorithms, I refer the reader to the treatments of Seuken & Zilberstein (2008), Allen (2009), and Oliehoek (2010).

2.3.1 Decentralized POMDPs

The qualifying prefix "Dec-" in Dec-POMDP refers to a decentralization both of control and of observation of an underlying POMDP. Instead of one agent taking actions and receiving observations, we now have a team of agents, each of which independently takes its own action and receives its own observation at every time step. Figure 2.5 shows a DBN, whose graphical structure illustrates the conditional independencies among Dec-POMDP variables. The formal details of the Dec-POMDP model are as follows.

Definition 2.3. A **Dec-POMDP** is specified by tuple $\langle \mathcal{N}, S, A, P, R, \Omega, O, T \rangle$, where

- $\mathcal{N}$ is a team of n agents,
- S is the *state space*, a finite set containing all world states, with distinguished *initial state*[7] s^0,

[6]The MTDP encodes policies as mappings from belief state to action rather than observation history to action.

[7]The conventional definition of the Dec-POMDP (Bernstein et al., 2002), for simplicity of exposition, specifies a unique start state instead of a distribution over start states ($\boldsymbol{\alpha}$). Note, however, that this does not restrict the representational power of the model.

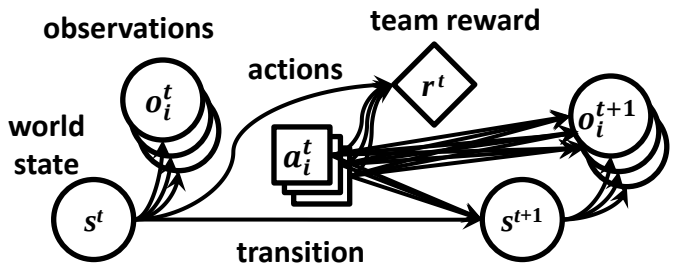


Figure 2.5: DBN describing relationships among Dec-POMDP variables.

- $A = A_1 \times \ldots \times A_i \times \ldots \times A_n$ is the *joint action space*, wherein component A_i refers to the finite set of *local actions* available to agent i,
- $P : S \times A \mapsto [0, 1]$ is the *transition function*, specifying the probability $P(s^{t+1}|s^t, a)$ that the agents will transition into world state $s^{t+1} \in S$ given that the agents performed joint action $a = \langle a_1, \ldots, a_i, \ldots, a_n \rangle \in A$ in state $s^t \in S$,
- $\Omega = \times_{i \in \mathcal{N}} \Omega_i$ is a finite set of *joint observations*, such that each agent i observes an observation $o_i \in \Omega_i$ with every transition that it makes,
- $O : A \times S \times \Omega \mapsto \mathbb{R}$ is the *observation function*, specifying the probability, denoted $O\left(o^{t+1}|a^t, s^{t+1}\right)$, is the probability that the agents receive joint observation $o^{t+1} = \langle o_1^{t+1}, \ldots, o_i^{t+1}, \ldots, o_n^{t+1} \rangle \in \Omega$ after taking taking actions $a^t = \langle a_1^t, \ldots, a_i^t, \ldots, a_n^t \rangle \in A$ and arriving in state $s^{t+1} \in S$,
- $R : S \times A \mapsto \mathbb{R}^n$ is the *reward function*, specifying the *team reward*, denoted $r^t = R(s^t, a^t)$, ascribed to the joint action $a^t \in A$ taken in state $s^t \in S$, and
- $T \in \mathbb{N}$ is the finite *time horizon*, specifying that the agents will face decisions at (discrete) time steps $\langle 0, 1, ..., T \rangle$.

Interactions among agents are manifested in the Dec-POMDP's transition, observation, and reward functions. The world state transition depends upon combinations of agents' actions. Similarly, an agent's observation (which is separate from the observation given to other agents) may depend on its own action, other agents' actions, and the new world state. A single team reward captures the immediate value of a joint action and resulting world state. Note that, just as in the single-agent MDP and POMDP, the reward is *not* explicitly observed, but simply provides a concrete specification by which to evaluate outcomes and policies.

Whereas in the single-agent POMDP the term *partial observability* referred to the agent's observations as distinct from the world state, the term takes on a richer meaning in the Dec-POMDP. Here, an agent's observation gives it a partial view of the world state as well as a partial view of the other agents' actions. Further, it may be the case that one agent completely observes part of the world state (where by part I mean either some state features' values or some regions of the state space) while another agent completely observes another part of the world state. In this sense, partial observability may also refer to the agents' differing views of their shared world.

Just as in the single-agent POMDP case, each Dec-POMDP agent i bases its decision at time step t on its *local observation history*, denoted $\vec{o}_i^t = \langle o_i^1, ..., o_i^t \rangle \in (\Omega_i)^t$.

Definition 2.4. A **local policy** $\pi_i : (\Omega_i)^T \mapsto A_i$ for agent i deterministically[8] specifies an action $a_i^t \in A_i$ that i will perform for each observation history $\vec{o}_i^t$.

The objective of a set of Dec-POMDP agents is, as in the single-agent case, to maximize the value function. In this dissertation (and in most finite-horizon Dec-POMDP planning), the value refers to the expected cumulative reward $E\left[\sum_{t=0}^{T} R(s^t, a^t)\right]$. In this case, the value function is dependent upon all agents' actions, as conveyed by the *joint policy*.

Definition 2.5. A **joint policy** $\pi = \langle \pi_1, ..., \pi_n \rangle$ is a vector of all agents' local policies.

Definition 2.6. The **value** of a joint policy π is the expectation of the summation of rewards received by following π:

$$V(\pi) = E\left[\sum_{t=0}^{T} R(s^t, a^t)|\pi\right] \tag{2.10}$$

Although Dec-POMDP agents' actions and observations are decentralized, the Dec-POMDP planning process need not be decentralized. In fact, at the present time, the vast majority of Dec-POMDP solution algorithms compute agents' policies centrally, either with a single computational process or by allowing arbitrary exchange of information between agents during the planning process. It is not until agents go off and *execute* their planned policies that the problem becomes "decentralized". During execution, agents do not explicitly share their observations nor communicate

[8] As described in Section 2.2.1.1, I restrict consideration in this dissertation to *deterministic* policies. Here and throughout, I will use the term *local policy* to mean an individual agent's deterministic local policy.

their actions.[9] Thus, even if the agents' decisions are all planned together, this does not guarantee that agents will necessarily execute these decisions in a synchronized, well-coordinated manner because they cannot be certain about the other agents' views. The fact that the agents' runtime awareness is disjoint but *not* independent makes the problem of optimal Dec-POMDP policy computation extremely challenging.

2.3.1.1 Complexity

While it comes as little surprise that planning for teams of agents is harder than planning for individual agents, it turns out that Dec-POMDP planning is in a whole different complexity class from that of POMDP planning (reviewed in Section 2.2.3). Bernstein et al. (2002) have proven the finite-horizon Dec-POMDP to be in complexity class *NEXP*, which is believed (though not proven) to be strictly harder than *NP*, and is widely considered *intractable* (Papadimitriou, 1994). Further, the NEXP-completeness holds for Dec-POMDPs with as few as two agents (Bernstein et al., 2002). This suggests[10] that the computation time of an optimal joint policy (by any algorithm) for a team of two Dec-POMDP agents is, in the worst case, *doubly-exponential* in the size of the Dec-POMDP problem description.

2.3.1.2 General Solution Methods

Despite the daunting complexity of these models, several optimal solution approaches have been developed for the general class of finite-horizon Dec-POMDPs. For instance, Bernstein et al. (2009) show that policy iteration using stochastic, correlated joint controllers converges on the optimal Dec-POMDP solution. Other optimal approaches include extensions of dynamic programming (Hansen et al., 2004) and A* heuristic search (Szer et al., 2005). Not surprisingly, none of the three optimal methods have been shown to scale beyond small 2-agent problems.

Approximate solution methods for the general class of Dec-POMDPs are more abundant. Oliehoek et al. (2008a) extend Szer et al.'s *multiagent* A* search to efficiently-computable approximate value functions. Seuken & Zilberstein (2007b) introduce heuristics into Hansen et al.'s optimal dynamic programming algorithm to reduce the number of joint policies considered. Nair et al. (2003) develop a policy-space

[9] Implicit communication may, however, be manifested by the Dec-POMDP's actions, observations, and transitions. Alternatively, there are extensions to the Dec-POMDP framework that augment the problem description with special communicative actions. I address these topics in a later chapter (Section 3.4.2).

[10] Formally, the double exponentiality of optimal Dec-POMDP planning is contingent upon the assertion that $NEXP \neq EXP$, which has yet to be proven.

search method, JESP (which I describe later on in Sec. 2.3.3), that converges upon a joint policy that is a Nash equilibrium but is not guaranteed to be optimal. Similarly, researchers have developed methods that search an approximate space by modeling policies with fixed-size local controllers. For instance, Bernstein et al. (2005) perform policy iteration on local stochastic finite-state controllers along with an additional shared controller that correlates the stochastic actions of the agents. Alternatively, Amato et al. (2007) optimize fixed-size local controllers using non-linear programming. Kumar & Zilberstein (2009) extend *point-based* methods (Pineau et al., 2006; Spaan & Vlassis, 2005) to approximate the Dec-POMDP belief-state space. Although these general-purpose approximate algorithms have enabled researchers to tractably solve problems with larger state and action spaces and longer time horizons than had the optimal algorithms, they have not been shown to scale to problems with more than two agents.

2.3.2 Structural Restrictions and Subclasses

With an eye towards avoiding the NEXP complexity of the general Dec-POMDP problem class, researchers have identified a variety of Dec-POMDP subclasses that are amenable to efficient, scalable solution methods, but that impose various restrictions on problem structure. Here, I survey the most common structural restrictions along with their associated Dec-POMDP subclasses.

2.3.2.1 Joint Observability

Intuitively, the difficulty of optimal coordination in Dec-POMDPs is due, in part, to agents' differing observations of their shared environment. The *Multiagent MDP* (*MMDP*) (Boutilier, 1996) sidesteps this problem by assuming that all agents completely observe the world state, effectively reducing the problem to a single-agent MDP with a joint action, whose computational complexity is just *polynomial* in the size of the problem description. An analogous assumption that agents receive the same *partial* observations reduces the *Multiagent POMDP* to a single-agent POMDP with a joint action (Messias et al., 2010).

A less restrictive assumption is the agents' observations *together* fully determine the world state. Bernstein et al. (2002) formalize this assumption as *joint observability* (whose definition I restate below), calling the resulting Dec-POMDP subclass the *Dec-MDP*.

Definition 2.7. A Dec-POMDP is **jointly observable** if there exists a mapping $J : \Omega \mapsto S$ such that whenever $O\left(\langle o_1^t, ..., o_n^t \rangle | a^{t-1}, s^t\right) > 0$, then $J\left(\langle o_1^t, ..., o_n^t \rangle\right) = s^t$.

Although joint observability dictates that agents' observations jointly determine the world state, it does not imply that any one agent will ever be aware of the true world state (since the agents are not assumed to share their observations during execution). Moreover, an agent's individual observation may not even be enough to establish awareness of a portion of the true world state. As such, only in combination with other structural restrictions (which I review in the subsections that follow) has *joint observability* led to computationally-efficient solution methods.

2.3.2.2 Local Full Observability

Another branch of work assumes that each agent i observes (exactly or partially) a portion of the world state s referred to as its **local state** s_i, where s_i consists of a subset of the feature values that make up the world state s (Becker et al., 2004b; Goldman & Zilberstein, 2004; Nair et al., 2005; Varakantham et al., 2009). In all of these models, the state is *factored* such that every feature appears in at least one agent's local state, and each agent's observations depend only on the values of its local state features.

Problems in which agents observe their local states exactly are commonly referred to as *locally fully observable* (Becker et al., 2004b; Goldman & Zilberstein, 2004), whose definition I review below.

Definition 2.8. A Dec-POMDP is **locally fully observable** if:

$\forall i \in \mathcal{N}, \forall o_i \in \Omega_i, \exists s_i \in S_i | Pr(s_i | o_i) = 1$, where S_i is agent i's local state space.

By Definition 2.8, an agent i's current local state s_i is uniquely determined from i's observation o_i, making local full observability a stronger assumption than joint observability. Whereas a Dec-POMDP that is *locally fully observable* is also *jointly observable* (Goldman & Zilberstein, 2004), the converse does not hold. In jointly-observable problems, an agent's individual observation alone might not determine any portion of the world state (but only in combination with other agents' observations provide awareness of the agent's local state).

The TI-Dec-MDP (Becker et al., 2004b), the EDI-Dec-MDP (Becker et al., 2004a), and the EDI-CR (Mostafa & Lesser, 2009), each of which I will describe in more detail (in Sec. 2.3.2.3–2.3.2.5) after reviewing more structural restrictions, are all subclasses that are locally fully observable.

2.3.2.3 Transition and Observation Independence

In addition to factoring the world state into local states, researchers have also imposed particular factored structure on the transition and observation functions. In particular, they have identified problems in which agents cannot affect the values of each others' local states (Becker et al., 2004b; Nair et al., 2005). These problems are referred to as *transition-independent* (Becker et al., 2004b).

Definition 2.9. A Dec-POMDP is **transition independent**[11] if:

$$Pr\left(s_i^{t+1}|s^t, a^t\right) = Pr\left(s_i^{t+1}|s_i^t, a_i^t\right)$$

By Definition 2.9, a *transition-independent* agent i's next local state value s_i^{t+1} depends only on its previous local state s_i^t and latest individual action a_i^t.

Along with transition independence, researchers have imposed *observation independence* (Becker et al., 2004b; Nair et al., 2005).

Definition 2.10. A Dec-POMDP is **observation independent** if:

$$O\left(o^{t+1}|a^t, s^{t+1}\right) = \prod_{i \in \mathcal{N}} O_i\left(o_i^{t+1}|a_i^t, s_i^{t+1}\right)$$

Here, the Dec-POMDP observation function $O()$ has been decomposed into a set of *local observation functions* $\{O_i()\}$, one for each agent i, dictating probabilities of individual observations which are assumed to be independent of peers' observations.

In problems that are both transition and observation independent, an agent i cannot affect the transitional outcomes of other agents' actions, nor can it affect other agents' observations, through any actions that i takes. Thus, the only form of interaction occurs through the reward function: one combination of agents' actions may be valued differently than another. Consider, for instance, a problem in which the successful delivery (*team reward* +1) of a package is contingent upon both a dockworker (agent 1) loading the package into a truck (*action* a_1) and a driver (agent 2) (*action* a_2) transporting the package to its destination.

Becker et al. (2004b) have identified the *Transition-Independent Dec-MDP* (TI-Dec-MDP) class, which restricts problems to be *locally fully observable* (Def. 2.8), *transition independent* (Def. 2.9), and *observation independent* (Def. 2.10). They have also proven that the TI-Dec-MDP is *NP-complete*, putting it in a complexity

[11] Definition 2.9 is simplified slightly from that given by Becker et al. (2004b) in its omission of *global features* s_0 (also referred to as *uncontrollable features* (Goldman & Zilberstein, 2004), and *unaffectable state* (Nair et al., 2005)), which do not depend on any agent's action. Here, I treat such features as jointly modeled in all agents' local states.

class (widely believed to be) easier than that of the general Dec-POMDP (Becker et al., 2004b). Subsequently, other researchers have developed algorithms for solving TI-Dec-MDPs approximately using mixed-integer linear programming (Wu & Durfee, 2006), and optimally using separable bilinear programming (Petrik & Zilberstein, 2009), which exploit the transition and observation independent structure.

Nair et al. (2005) have identified another class, the Network-Distributed POMDP (ND-POMDP), that is *transition and observation independent* but not locally fully observable, which has led to the development of a suite of exploitative algorithms (Kim et al., 2006; Kumar & Zilberstein, 2009; Marecki et al., 2008; Nair et al., 2005; Varakantham et al., 2007), and demonstrations of quality-bounded solution computation for problems with up to 10 agents, thereby making a significant leap in Dec-POMDP scaling. However, these algorithms remain limited in their applicability to *transition and observation independent* problems, examples of which include the control of distributed sensor networks wherein at each decision step agents choose only where to point their sensors and cannot not affect each others' observations or local states.

2.3.2.4 Reward Independence

As an alternative to *transition and observation independent* problems, Becker et al. (2004a) have developed a different class of problems that imposes a factoring of the team reward into *local rewards*.

Definition 2.11. A Dec-POMDP is **reward independent** if there are functions f and R_1 through R_n such that

$$R(s,a) = f\left(R_1(s_1,a_1), R_2(s_2,a_2), ..., R_n(s_n,a_n)\right)$$

and

$$R_i(s_i,a_i) \leq R_i(s_i,a_i') \Leftrightarrow f\left(R_1...R_i(s_i,a_i)...R_n\right) \leq f\left(R_1...R_i(s_i,a_i')...R_n\right)$$

In Definition 2.11, R_i is agent i's *local reward function*, valuing i's *local state* and individual action independently of the other agents' local states and individual actions. The reward composition function f, which is restricted to be monotonic, defines the resulting team reward. Reward independence, in the context of the models described in Section 2.3.2.5, has enabled researchers to decompose the Dec-POMDP value function into local value functions, and to exploit the resulting factored structure.

2.3.2.5 Event-Driven Interactions

Becker et al. (2004a) define a subclass called the *Dec-MDP with Event-Driven Interactions* (EDI-Dec-MDP), which combines *reward independence* and *local full observability* with another property that restricts agents' interactions to take a particular form. Due to its similarity to the model that I develop in Chapter 3, I describe the formal details of the EDI-Dec-MDP in Appendix A, which I briefly summarize here. Each interaction takes the form of a special transition dependency. For an agent i whose actions affect agent j, the EDI-Dec-MDP models a *dependency* that relates the occurrence of an *event*, which is a transition of agent i's local state, to the probability of a subsequent transition of agent j's local state.

EDI-Dec-MDP agents are always reward independent, and they are transition independent in all world states except those explicitly represented with a dependency. Using this insight, Becker et al. (2004a) develop a solution approach for EDI-Dec-MDPs that iteratively solves nearly-independent local models augmented with nonlocal event information, and demonstrate their algorithm to be much more efficient than exhaustive joint policy search. However, no EDI-Dec-MDP algorithms to date have been shown to scale beyond two agents.

2.3.2.6 Hierarchy of Methods With Fixed Execution Ordering

Another branch of work, in addition to requiring structured interactions, imposes restrictions on agents' local behaviors. The *Opportunity-Cost Dec-MDP* (OC-Dec-MDP), introduced by Beynier & Mouaddib (2005) and studied by Marecki & Tambe (2007), models a team of agents whose objective is to coordinate the execution times of *methods* with stochastic durations. Unlike the other models reviewed thus far, the OC-Dec-MDP specifies a fixed ordering over each agent's method executions, restricting the problem to one of determining only when to start each method and *not* which order to execute the methods in.

OC-Dec-MDP interactions take the form of precedence constraints, each dictating that a method executed by one agent will only complete successfully if a particular method of some other agent has already completed successfully. In combination with *local full observability* and the fixed ordering over method executions, this restricted form of transition dependence makes the OC-Dec-MDP more practical for scaling to problems with many methods (or many agents). Researchers have exploited this specialized structure to compute approximate solutions containing over a hundred methods (Marecki & Tambe, 2007).

2.3.2.7 Other Subclasses

In addition to those described in the previous subsections, researchers have identified several other specialized Dec-POMDP subclasses whose structure has allowed for efficient computation of optimal solutions. For instance, Goldman & Zilberstein (2004) defines a *Goal-Oriented Dec-MDP* (GO-Dec-MDP) wherein the objective is to minimize the cost of agents' actions en route to one of a subset of *goal* states, and proves that, when combined with *transition and observation independence*, the Go-Dec-MDP is *polynomial*. Guo & Lesser (2005) define a *Partially-Observable Stochastic Game with state-dependent action sets*, wherein each agent controls a separate MDP that is independent from the others except that the agent's set of available actions depends upon other agents' MDP states, and demonstrate that for such problems the joint policy space can be reduced significantly by iteratively removing dominated local policies. Dolgov & Durfee (2006) defines a flavor of Dec-MDP for *multiagent resource allocation*, wherein agents' individual MDPs are completely independent with the exception of constraints on joint actions that depend upon an initial allocation of resources. Wu & Durfee (2010) extend Dolgov & Durfee's formulation to the case of sequential resource re-allocation, defining the *multiagent resource-driven mission phasing problem* (M-RMP) and proving the computationally complexity of this subclass to be *NP-complete*.

Meanwhile, others have defined subclasses with fewer structural restrictions, but that have only been shown to accommodate efficient *approximate* solution methods. For instance, Varakantham et al. (2009) define the *Distributed POMDP with Coordination Locales* (DPCL), which requires only *observation independence* (Def. 2.10) and a decomposition of the team reward into local rewards. The authors demonstrate that by explicitly distinguishing all of those world states (or *locales*) in which agents can interact, an efficient distributed algorithm can exploit the underlying structure to compute solutions efficiently, but without guaranteeing optimality or near optimality. Guestrin et al. (2001) define a *hierarchical multiagent factored MDP* whose structure can be exploited by an efficient *approximate* linear programming algorithm. Lastly, Oliehoek et al. (2008b) describe a generalization of Guestrin et al.'s model called the *factored Dec-POMDP*, which explicitly represents factored value functions whose components depend on subsets of state variables and subsets of agents' actions. While the structure in these more general subclasses has allowed efficient and scalable computation of approximate solutions, no generally applicable algorithms have yet been developed that can compute quality-bounded solutions for problems with more than three agents.

2.3.3 Decoupled Joint Policy Formulation

By and large, the successes (some of which are cited in Section 2.3.2) in scaling planning to Dec-POMDPs with more than two or three agents have come though the use of *decoupled* solution methods. In contrast to centralized planning algorithms that formulate joint behavior by reasoning about all agents' decisions in combination, a decoupled algorithm breaks the computation up into coordinated local policy formulations. I now review the infrastructural groundwork (on which my own solution approach also rests), the results that others have attained, and the limitations of this past work.

2.3.3.1 Best Response

Central to the decoupled solution approach is the use of local models to separately compute each agent's individual policy. As derived by Nair et al. (2003), any Dec-POMDP can be transformed into a single-agent POMDP for agent i given that the policies of its peers have been fixed. Agent i uses the single-agent POMDP to compute its *best response* policy, which I will denote $\pi_i^*(\pi_{\neq i})$, that is optimal with respect to the policies of its peers ($\pi_{\neq i}$). The idea is to compute best responses to a series of candidate policies of i's peers, as illustrated in Figure 2.6.

Nair et al. (2003) provide a dynamic programming algorithm for computing best responses to the general class of Dec-POMDPs. Though computationally less expensive than computing a complete joint policy, Nair et al.'s best-response computation requires that the agent reason about the space of possible observation histories of its peers, which increases exponentially with the time horizon (T) and with the number of peers. Consequently, Nair's general best response computation has failed to scale to problems with more than two agents (Varakantham et al., 2009).

In a more restrictive context, researchers have devised best-response models that provide substantial leverage (Becker et al., 2004b; Nair et al., 2005) in reducing computational cost. They take advantage of the locality of agents' interactions (Nair et al., 2005), such that the agent reasons about only the observation histories of a subset of peers' and only a subset of state features. However, these specialized models are only applicable to *transition and observation independent* Dec-POMDPs (Becker et al., 2004b; Nair et al., 2005).

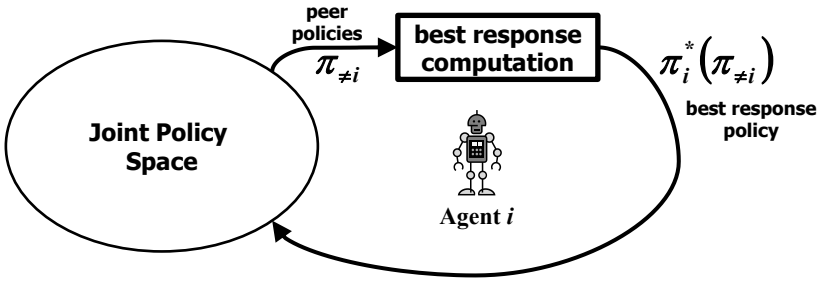


Figure 2.6: Decoupled joint policy search.

2.3.3.2 Policy-Space Search

Given the decoupling scheme that the best response model provides, planning the joint policy becomes a search through the space of combinations of optimal local policies (each found by solving a local best-response model). Nair et al. (2003) develop a general algorithm, *Joint Equilibrium-based Search for Policies* (JESP), for searching the joint policy space in this manner. Using JESP, agents iteratively revise their policies by computing best responses to each others' best responses, ultimately converging on a (Nash) equilibrium that is a local optimum but not necessarily a (Pareto-efficient) global optimum. A subsequent extension to JESP, the *Global Optimal Algorithm* (GOA), ensures that agents compute best responses to all interacting peers' policies, and thus returns the optimal joint policy (Nair et al., 2005). However, GOA is also limited in its scalability due to the intractable growth of the joint policy space.

Researchers have also employed joint policy search for solving problems in the ND-POMDP class (Nair et al., 2005; Marecki et al., 2008; Kim et al., 2006; Varakantham et al., 2007). By exploiting locality of interaction (Nair et al., 2005; Kim et al., 2006), using smart pruning techniques (Varakantham et al., 2007), and replacing policies with fixed-size controllers (Marecki et al., 2008), they have been successful in scaling up the computation of joint policies to transition-dependent problems with 10 agents. With one such algorithm, SPIDER (Varakantham et al., 2007), they have additionally been able to bound the quality of the solutions returned. However, none of these scalable algorithms are directly applicable to *transition-dependent* problems.

2.3.3.3 Adaptations

Researchers have developed other *decoupled joint policy formulation* methods, maintaining the same paradigm as I have just described, by adapting either the search process or the the mechanics of the best response calculation. For instance,

Varakantham et al. (2009) have designed an algorithm, *Team's REshaping of MOdels for Rapid execution* (TREMOR), for computing approximate solutions to DPCLs (as described in Sec 2.3.2.7). Like JESP (described in Section 2.3.3.2), TREMOR employs local models, iteratively computing individual policies in response to candidate peer behavior, and greedily converges on a local optimum. The difference is that instead of computing an optimal best response, TREMOR uses *social model shaping* to compute approximate best responses. Agents construct local models whose transitions have been shaped to account for expected peer effects (upon entering into predetermined coordination locales) and whose rewards have been shaped to encourage the agent to select harmonious actions (in coordination locales). By foregoing optimality, TREMOR's local response calculation has been shown to scale to problems with 10 agents. However, it provides no guarantees of near optimality, nor any bounds on the quality loss due to the approximate best response.

Another adaptation, the *Coverage Set Algorithm* (CSA) (Becker et al., 2004a,b), which was originally designed to solve TI-Dec-MDP problems, is built on the same best-response concept as JESP, GOA, and SPIDER. Becker et al. (2004b) defines an *optimal coverage set* as the set containing each local policy that is a best response to some combination of peer policies. By considering all policies in the coverage sets of all agents, CSA ensures that the optimal joint policy will not be overlooked, which is the same insight behind GOA. The novelty of CSA lies in its exploration of the coverage set. Policies are abstracted using a collection of parameters over which the best-response value function is piecewise-linear and convex. CSA then searches the parameter space by evaluating policies that correspond to hyperplane intersections along the surface of the optimal joint value function. Such a parameterization was first defined for TI-Dec-MDPs based on the joint reward structure (Becker et al., 2003). Another parameterization was later defined for EDI-Dec-MDPs (Becker et al., 2004a), but has since only been shown to be tractable on small two-agent event-driven problem instances.

Petrik & Zilberstein (2009) have since reformulated CSA as an optimization problem called a *separable bilinear program*, and developed a centralized solution approach, which I will refer to as SBP, that has been shown to significantly outperform the basic CSA implementation. SBP works by repeatedly solving an optimization problem according to an approximation bound, successively refining the bound from iteration to iteration. Aside from converging more quickly than CSA in practice, it also has the advantage of allowing the agents to compute anytime solutions with bounds on approximate solution quality. However, it has only been developed for

solving problems with two agents. Extension to more than two agents is nontrivial by nature, since the consequent mathematical formulation would no longer constitute a "bilinear" program.

2.3.4 Coordinating Abstract Behavior

Another paradigm central to this dissertation is the coordination of abstract interactions. Intuitively, agents don not always require detailed models of peers' individual behavior in order to coordinate their decisions. Instead, they only need to consider the portions of peers' behavior relating to their interactions. In the context of a decoupled solution approach (Sec. 2.3.3), agents can formulate coordinated joint behavior by negotiating abstract *commitments* to interactions and planning local behavior around those commitments.

Historically, this has been a dominant approach in multiagent planning. As early as 1980, the *Contract Net* protocol (Smith, 1980) provided a convention for agents to commit to executing necessary subtasks of a larger problem. Cohen & Levesque (1990, 1991) use the commitment paradigm to develop a theory of *Joint Intentions* by which agents commit to performing actions in states that will allow achievement of persistent goals. Grosz & Kraus (1996) formalize commitments into a model of agents' simultaneous completions of plan components with their *SharedPlan* framework. Durfee & Lesser (1991) develop a *Partial Global Planning* methodology (subsequently generalized by Decker & Lesser, 1992), wherein agents coordinate their interactions by exchanging group goals and integrating agents' commitments in the form of partial plans that can be used to complete those goals. Meanwhile, local plans are formed around the promised partial plans and revised (when the need arises) to adapt to dynamic environmental factors and unexpected circumstances. More recently, researchers have used the concept of partial global planning to develop algorithms wherein agents identify coordination points and coordinate using abstract characterizations of their interactions (Clement et al., 2007; Cox & Durfee, 2003; Xuan & Lesser, 1999). Other examples wherein agents coordinate abstract interactions include Tambe's *Shell for Teamwork* (*STEAM*) (Tambe, 1997) based on Cohen's theory of *Joint Intentions*, Jennings' *Generic Rules and Agent model Testbed Environment* (*GRATE**) (Jennings, 1995) which defines an extension to *Joint Intentions* called *Joint Responsibility*, and Rich & Sidner's Collaborative Agent toolkit (*COLLAGEN*) (Rich & Sidner, 1997), based on Grosz's *SharedPlans* theory.

The principle of coordinating abstract interactions has received considerably less attention in Dec-POMDP settings. Most of the *decoupled policy formulation* techniques

described in Section 2.3.3 involve coordination through the exchange of complete policies, that represent both local and interacting behavior. Alternatively, CSA (Becker et al., 2004b) employs abstraction by parameterizing one agent's policies using expectations about nonlocally-affecting events. However, this particular parametrization is limited to the TI-Dec-MDP (Becker et al., 2004b) and the EDI-Dec-MDP (Becker et al., 2004a). Musliner et al. (2006) develops a distributed planning algorithm wherein agents communicate commitments about the timings of their interdependent task executions, which are in turn modeled using local MDPs. However, this particular form of commitment does not incorporate uncertainty in agents' interactions, thereby providing only an approximate model of interaction. Similarly, TREMOR (Varakantham et al., 2009) employs approximate local models (using transition and reward shaping) that abstract agents' interactions. However, TREMOR's search process dictates that agents communicate complete policies without regard to the abstract interactions that they entail, leading to convergence on local optima and no guarantees that agents will consider any breadth of committed interactions.

2.4 Summary

In summary, I have reviewed work in single-agent sequential decision making (Section 2.2) and multiagent sequential decision making (Section 2.3) that forms the foundations of the work that I develop in this dissertation. I have also surveyed related approaches to coordination under uncertainty, and drawn attention to the limitations of past work when it comes to scalability, bounded solution quality, and applicability.

In particular, I find that researchers have developed several *general* algorithms, for computing quality-bounded solutions to transition-dependent flavors of Dec-POMDPs, that are limited to problems with just two or three agents (due to their computational overhead). Alternatively, researchers have developed algorithms that demonstrably scale to teams of 5 or 10 agents and guarantee bounds on solution quality, but that are limited in their applicability to specialized subclasses with restrictive assumptions (described in Section 2.3.2). There is no prior work that both solves a general flavor of transition-dependent problems *and* scales to more than three agents whilst guaranteeing bounds on quality.

The latter group of algorithms (that scale quality solution computation) have achieved their scalability by decoupling the joint policy formulation problem (Section 2.3.3) and by identifying and exploiting specialized structure in agents' interactions. It is important to identify instances of structure that lend themselves to efficient and

scalable solution methods. However, in this pursuit, I find that there is a tendency in past work for each instance of exploitable structure to be studied separately. This is evident from the large number of Dec-POMDP subclasses reviewed in Section 2.3.2 (e.g., TI-Dec-MDPs, OC-Dec-MDPs, GO-Dec-MDPs, ND-POMDPs, EDI-Dec-MDPs, EDI-CRs, DPCLs), each of which comes with its own set of restrictions, and each of which is accompanied by its own specialized solution algorithms. The field of multiagent sequential decision making lacks models that are both general (and hence of interest to a broad group of researchers) and exploitable (and hence enable solution methods that are efficient and scalable to the extent that exploitable structure is present).[12]

[12] A notable exception is the factored Dec-POMDP (Oliehoek et al., 2008b), whose exploitable structure I describe in the next chapter (Section 3.4.1).

CHAPTER 3

Exploiting Transition-Dependent Interaction Structure

In spite of the general intractability of Dec-POMDP planning, a large body of work surveyed in the last chapter has shown us that there exist subclasses of Dec-POMDP problems, even some involving more than two or three agents, that are tractable. Moreover, this past work has provided us with tools to compute solutions efficiently by exploiting restricted instances of problem structure. Inspired by the successful solving and scaling of *transition and observation independent* problems (Nair et al., 2005; Varakantham et al., 2007), and with the ambition of reproducing these results under less restrictive conditions, I now introduce a new Dec-POMDP subclass that is more general than other subclasses, along with a corresponding model description that articulates exploitable problem structure. My class of Transition-Decoupled POMDP (TD-POMDP) problems serves as the context for the remaining chapters of this dissertation.

The TD-POMDP is named for its structure: it consists a set of transition-dependent local POMDP models, one for each agent, that can be decoupled by fixing peer agents' policies and abstracting their transition influences. Before developing the mechanics of TD-POMDP decoupling and influence abstraction (in Chapter 4), here I provide a formal description of the TD-POMDP's exploitable interaction structure as well as a theoretical motivation for exploiting this structure. Intuitively, the computational leverage gained through decoupling and abstraction depends upon the extent to which conditional independencies exist among agents' decisions that render the agents *weakly coupled*. Extending past work, I characterize three complementary aspects of weakly-coupled interaction structure, relate each to the TD-POMDP problem description, and derive bounds on the TD-POMDP's computational complexity that depend upon the degree of agent coupling.

3.1 Overview

The contents of this chapter are structured as follows. I begin, in Section 3.2, by presenting the formal details of the TD-POMDP model, expressed as properties that constrain the Dec-POMDP formalism. The structure that these additional properties induce leads me, in Section 3.2.4, to specify the TD-POMDP problem description as a collection of interdependent local models whose *mutually-modeled features* (through which agents interact) are treated as first-class entities. In Section 3.3, I formally describe what it means to solve the TD-POMDP and how difficult this problem is.

Although just as complex as the general Dec-POMDP in the worst case, the benefit of the TD-POMDP lies in its emphasis of exploitable structure. In Section 3.4, I contrast the TD-POMDP with other models, comparing the structure that each articulates as well as the problem restrictions that each imposes. As a step towards exploiting TD-POMDP interaction structure, in Section 3.5 I develop theory for characterizing what it means for a problem to be weakly coupled, and for measuring the degree to which it is coupled. My characterization includes three different aspects of weak coupling that, when considered in concert, lead me to develop tighter bounds on the complexity of computing optimal TD-POMDP solutions. I conclude, in Section 3.6, with a summary of the formalisms I have introduced and theoretical results I have derived, and a discussion of their respective contributions.

Throughout this chapter, I refer to example problems of the form shown in Figure 3.1 and described in Example 3.1 below. Like Example 2.1 (shown in Figure 2.2), Example 3.1 is depicted in Figure 3.1 as a network of interdependent tasks whose relationships, indicated by connecting lines, may be described using a variation of the TÆMS modeling language (Decker, 1996). Figure 3.1 highlights one such task relationship, which constitutes a structured transition-dependent interaction between two TD-POMDP agents.

Example 3.1. Figure 3.1 presents a concrete example of a structured interaction from the planetary exploration domain described in Section 1.1.1. Here, a satellite agent (1) interacts with a rover agent (7) by building a path for the rover to travel from its present location to site A. The path-building task (representative of a lower-level path-planning routine) has two possible outcomes whose *durations* ("D"), *qualities* ("Q") and probabilities ("Pr") are given. As shown, in the case

that the satellite completes task "Build-Rover-Path-A" successfully (with outcome quality > 0), this influences the outcome of a rover's task, "Visit-Site-A", allowing the rover to visit site A more quickly (in 3 time units instead of 6) with high probability (0.9 instead of 0.1, as denoted by the arrow in the last column of the "Visit Site A" task). This task relationship is just one example of an interaction that may exist among the team of satellites and rovers shown in Figure 3.1.

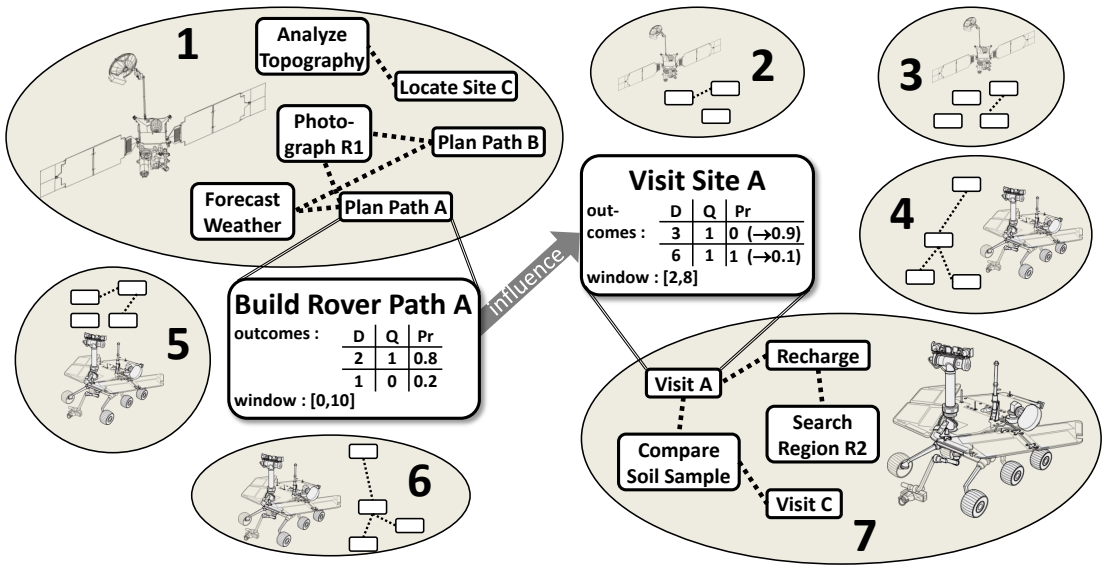


Figure 3.1: Example of structured interaction among TD-POMDP agents.

3.2 TD-POMDP Formalism

Decentralized Partially-Observable Markov Decision Processes (Dec-POMDPs), as reviewed in Section 2.3.1, provide a powerful, well-studied framework for multiagent planning under uncertainty. Here I present a formal specification of the subclass of Dec-POMDP problems that this thesis addresses: the *Transition-Decoupled POMDP*. In Subsections 3.2.1–3.2.3, I specify precisely how the key problem characteristics outlined in Section 1.2 translate into the formal properties that define the Transition-Decoupled POMDP. Then, in Subsection 3.2.4, I bring all of these properties together into a concise representation of TD-POMDP problem information.

3.2.1 Factored Decomposability

The world state s is *factored* into state features (denoted as $s = \langle b \in B, c \in C, d \in D, \ldots \rangle$, for instance), each of which represent a different aspect of the environment. Equivalently, the world state space S may be represented as the cross product of individual feature domains: $S = (B \times C \times D \times \ldots)$. Factoring of the world state allows us to express the conditional independence relationships that exist among the variables (i.e., state features, observation features, actions, and rewards) of the decision model (Boutilier, 1996; Guestrin et al., 2003). As reviewed in Chapter 2, factorization in multi-agent sequential decision making is not original to this dissertation. However, the way in which the TD-POMDP model is factored sets it apart from related models.[1] The definitions that follow serve to formalize the factorization particular to the TD-POMDP.

By factoring the world state, we can impose a distribution of environment information among the agents. Different state features are relevant to different agents as they make decisions about which activities to pursue. Moreover, some features may not be available to an agent. Limited sensory capabilities may restrict the agent's awareness to only a small subset of features. Using Definition 3.2 below, not all features are *observable* to all agents.

Definition 3.2. A state feature f, with domain $F = \{f, f', f'', ...\}$, is **observable** to agent i if and only if i's observation o_i^{t+1} depends upon the concurrent value of f (for some combination of state, action, and observation)[2]:

$$\exists o_i^{t+1} \in O_i, a^t \in A, \langle b, c, ..., f, ... \rangle^{t+1} \in S, \langle b, c, ..., f', ... \rangle^{t+1} \in S$$
$$\text{such that}$$
$$Pr\left(o_i^{t+1} | a^t, \langle b, c, ..., f, ... \rangle^{t+1}\right) \neq Pr\left(o_i^{t+1} | a^t, \langle b, c, ..., f', ... \rangle^{t+1}\right)$$

It is important to distinguish *observability* (Def. 3.2) from the (previously stated) concept of *full observability* (as in Def. 2.8, and sometimes referred to as "direct observability"). Whereas these previous terms refer to an agent's awareness of the exact value of a state feature f, Definition 3.2 makes no distinction between exact observations and partial observations. As long as the agent's observation o_i is not conditionally independent of the concurrent value of f (conditioned on other state

[1] A detailed comparison with closely-related models is deferred to Section 3.4.1, after the TD-POMDP has been formally specified.

[2] Just as in Chapter 2, here and throughout, the superscripts t and $t+1$ simply refer to two successive decision steps. Similarly, all components of model are *stationary*, such that they return the same value regardless of the particular value of t. These superscripts should *not* be confused with *time*, which is a feature of state, as described in Section 3.2.3.

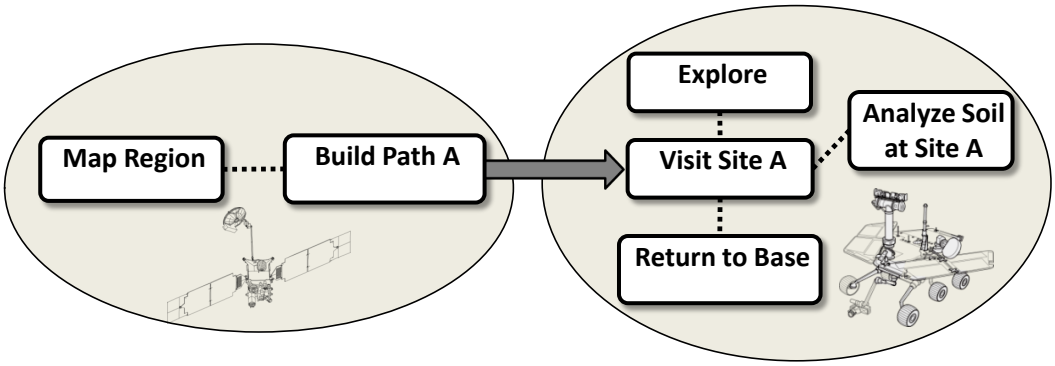


Figure 3.2: A simple satellite-rover example problem.

features' values), f is *observable* to the agent. Throughout this dissertation, the usage of "observes", "observable", and "observability" in relation to state features refers to Definition 3.2.

Example 3.3. Figure 3.2 depicts a simplified version of the running example. There are just two agents, a satellite and a rover, with a small number of activities. The dotted lines between activities indicate the existence of local constraints. For instance, the satellite cannot build a path for the rover until it has mapped the region, and the rover cannot analyze the soil at site A after it has returned to base. Moreover, neither agent can execute multiple tasks simultaneously. Here, there may be a number of different state features that the agents can model. Given that the rover performs all of its activities on the surface, and the satellite is located far above the surface looking down, the two agents will have vastly different perspectives of the world. Whereas the rover agent may be concerned with the composition of the soil sample it has just analyzed, which I will denote "soil-composition-at-site-A" (SCA), this is not relevant to the satellite agent's activities, nor is the satellite equipped with sensors for analyzing soil. "path-A-built" (PAB), on the other hand, is a feature that is relevant to both agents. The satellite should model PAB so that it does not perform redundant computations. The rover should model PAB because this feature impacts the rover's ability to visit Site A (as indicated by the arrow in Figure 3.2). PAB is observable to both agents during execution because the satellite broadcasts the completed path to the rover .

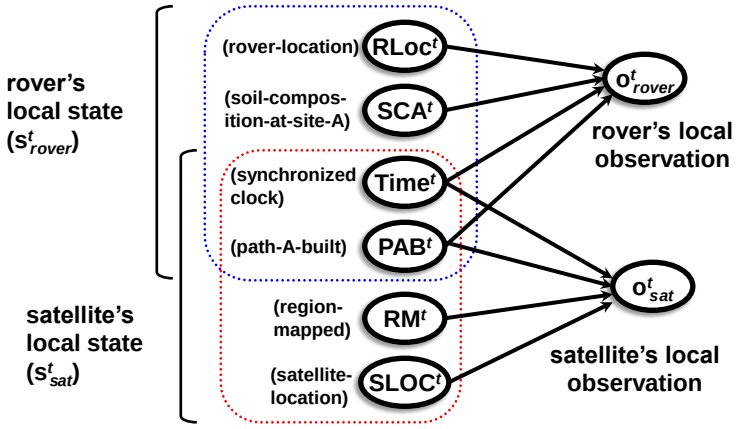


Figure 3.3: Example of local state representations and local observations.

Ultimately, it is the task of the problem designer to specify how the awareness of various state information is distributed among the agents. By making a feature observable to an agent, through manipulation of sensor infrastructure or communication infrastructure (as discussed in Section 3.4.2), the designer can alter which information is used for decision making by which agents as they execute their activities. For this purpose, as in other related models (e.g., those discussed in Section 3.4.1), the TD-POMDP world state s is aggregation[3] of agents' *local states*:

$$s = \langle s_1, ..., s_n \rangle . \tag{3.1}$$

The designer of a TD-POMDP problem indicates which features are relevant to each agent i by specifying i's local state s_i according to the constraints given in Definition 3.4. Figure 3.3 portrays the local state representations for the satellite and rover (from Example 3.3) in the form of a Dynamic Bayesian Network (DBN), where the arrows represent dependencies between state feature variables and observation variables.

Definition 3.4. The **local state** for TD-POMDP agent i, denoted $s_i = \langle f_{i1}, f_{i2}... \rangle$, represents a subset of world state features, such that the following properties hold:

1. For every world state feature f, if f is not contained in s_i, f must be contained within some other agent's world state.

[3]Unlike related models (e.g., Becker et al., 2004a; Nair et al., 2005), TD-POMDP agents' local states may share features.

2. If a world state feature f is observable to agent i, f must be contained within i's local state representation.

Property 1 of Definition 3.4 requires that every world state feature be represented in at least one agent's local state. Property 2 implies that an agent may observe (partially or fully) only those features that make up its local state. As such, agent i's local observation o_i satisfies the following equality:

$$Pr\big(o_i^t | a^t, s^t = \langle ..., s_i^t, ... \rangle\big) = Pr(o_i^t | a^t, s_i^t). \tag{3.2}$$

Local state thereby allows for a separation of features observable to one agent from features observable to another agent. It is important to note that the separation need not be strict. For instance, certain state features (such as "PAB" in Figure 3.3) may be observed by more than one agent, and thus shared by more than one agent's local state representation. Equation 3.2 describes one aspect of the *local observation function* fully specified in the following definition.

Definition 3.5. The **local observation function** $O_i : A_i \times S_i \times \Omega_i \to \mathbb{R}$, functionally denoted $O_i\big(o_i^{t+1} | a_i^t, s_i^{t+1}\big)$, dictates the probability with which agent i will receive observation $o_i^t \in \Omega_i$ after taking action $a_i^t \in A_i$ and transitioning into local state $s_i^{t+1} \in S_i$. All agents' local observation functions together define the probabilities of joint observations:

$$Pr\left(o_1^{t+1}, ..., o_n^{t+1} | a^t, s^{t+1}\right) = \prod_{1 \le i \le n} O_i\left(o_i^{t+1} | a_i^t, s_i^{t+1}\right). \tag{3.3}$$

The TD-POMDP's local observation functions allow agents possible observability of their local state features but not of features outside their local states. Although factored, the agents' observations are not necessarily independent (a requirement of other related models (Becker et al., 2004b; Nair et al., 2005)). They may be dependent on features shared across local states (as well as joint action choices that are correlated because of features shared across local states). However, by Equation 3.3, the observation probabilities are conditionally independent given values of the shared state features. For instance, if two rover agents are traveling the same path looking for a particular landmark, and if there is a probability that each agent will fail to detect the landmark as it passes by, the probability with which rover 1 finds the landmark and the probability with which rover 2 finds the landmark are assumed to be conditionally independent of one another (conditioned on the current world state

and the agents' latest actions).[4]

The reward function for the TD-POMDP is similarly decomposed into local reward functions, each dependent on local state and local action.

Definition 3.6. The **local reward function** $R_i(s_i^t, a_i^t)$ indicates the local component of the immediate team reward, which is ascribed to agent i's transition from local state to next local state given local action. The agents' local reward functions combine by summation to yield the *team reward* (represented in the general Dec-POMDP model):

$$R\left(s^t, a^t\right) = \sum_{i=1}^{n} R_i\left(s_i^t, a_i^t\right). \tag{3.4}$$

In the example problem, an agent's local rewards are the qualities attained from the tasks that the agents execute.

Not only is the team reward decomposable, but additionally, the value of any given joint policy can be expressed as the composition of local values. The composition of TD-POMDP agents' local values, which is sometimes referred to as a *quality accumulation function* (Decker, 1996), must again be *summation*, by Theorem 3.8 below. This makes the TD-POMDP somewhat more restrictive than the general Dec-POMDP. For instance, it would unnatural (if at all feasible) to use the TD-POMDP to model situations in which the joint value is the *maximum* of agents' local values. I provide further discussion of this issue in in Section 3.4.3.2.

Definition 3.7. The **local value** $V_i(\pi)$ is the expectation of the non-discounted summation of *local rewards* (Def. 3.6) for agent i given that the team of agents adopts joint policy π:

$$V_i(\pi) = E\left[\sum_{t=0}^{T} R_i(s^t, a^t)|\pi\right] \tag{3.5}$$

Theorem 3.8. *The* (joint) value V *of a joint policy* π *is the summation of* local values*:*

$$V(\pi) = \sum_{i=1}^{n} V_i(\pi) \tag{3.6}$$

Proof. This follows directly from the definition of Dec-POMDP value function (Def. 2.6) and the definition of the local reward function (Def. 3.6):

[4]Without loss of generality, dependencies among TD-POMDP agents' observations occur through their shared state features. That is, arbitrarily-complex observational dependencies are representable by sharing additional features among agents' local states.

$$
\begin{aligned}
V(\pi) &= E\left[\sum_{t=0}^{T} R(s^t, a^t)|\pi\right] && \textit{by Definition 2.6} \\
&= E\left[\sum_{i=1}^{n}\sum_{t=0}^{T} R_i(s^t, a^t)|\pi\right] && \textit{by Definition 3.6} \\
&= \sum_{i=1}^{n} E\left[\sum_{t=0}^{T} R_i(s^t, a^t)|\pi\right] && \textit{by the linearity of expectation} \\
&= \sum_{i=1}^{n} V_i(\pi) && \textit{by Definition 3.7}
\end{aligned}
$$

□

Notice that *local value* is defined as a function of joint policy (and not simply local policy). This is due to the combination of the following properties: (1) the TD-POMDP local rewards are dependent on local state feature values, and (2) given that local state features may be shared, local state values may be affected by other agents' actions (as described in detail in Section 3.2.2). As such, TD-POMDP agents are not strictly reward independent (Def. 2.11).

3.2.2 Nonconcurrently-Controlled Nonlocal Features

The transition dynamics of the TD-POMDP are also factored in such a way that enable agents to reason individually about the values of the features of their local states. Before formally developing the TD-POMDP transition function, let me begin by defining the concepts of *controllability* and *affectability*.

Definition 3.9. A state feature f_{ix} is **controllable**[5] **by** agent i if and only if:

$$
\begin{gathered}
\exists \langle a_1, ..., a_i, ..., a_n\rangle \in A, a'_i \in A_i, s^t \in S \\
\text{such that} \\
Pr\left(f_{ix}^{t+1}|s^t, \langle a_1, ..., a_i, ..., a_n\rangle\right) \neq Pr\left(f_{ix}^{t+1}|s^t, \langle a_1, ..., a'_i, ..., a_n\rangle\right)
\end{gathered}
$$

Definition 3.9 states that agent i can control a feature f_{ix} if the value of f_{ix} may depend upon i's latest action. However, it does not say anything about i's actions in previous time steps. By relaxing the condition from Definition 3.9, I define a slightly more general concept that I refer to as *affectability*.

[5]My definition of controllability is an extension of Goldman & Zilberstein's (2004) definition of *uncontrollable* features. It is a departure from the concept of *controllability* developed in the control theory literature (e.g., Ogata, 1997). Here, regardless of whether or not an agent can manipulate a feature f_{ix} deterministically, and whether or not it can set the value of f_{ix} at its will, f_{ix} is controllable as long as the agent can alter the probability of f_{ix} taking on some value in the next time step.

Definition 3.10. A state feature f_{ix} is **affectable by** agent i if and only if:

$$\exists \vec{a}_i^t \in (A_i)^t, \vec{a}_i^{t\prime} \in (A_i)^t$$
$$\text{such that}$$
$$Pr\left(f_{ix}^{t+1} | \vec{a}_i^t\right) \neq Pr\left(f_{ix}^{t+1} | \vec{a}_i^{t\prime}\right)$$

A feature f_{ix} is affectable by an agent i as long as its value is dependent on past actions taken by i. If a feature is controllable by i, it must also be affectable by i. However, a feature that is affectable need not be controllable. For instance, feature Rloc in Figures 3.3–3.4 is affectable by the satellite (since the satellite controls PAB and the value of RLoc depends upon PAB) but not controllable by the satellite. Any feature that is not affectable by agent i is *unaffectable*.

Definition 3.11. A state feature f_{ix} is **unaffectable by** agent i if and only if:

$$\forall \vec{a}_i^t \in (A_i)^t, \vec{a}_i^{t\prime} \in (A_i)^t,$$
$$Pr\left(f_{ix}^{t+1} | \vec{a}_i^t\right) = Pr\left(f_{ix}^{t+1} | \vec{a}_i^{t\prime}\right)$$

In the TD-POMDP model, each state feature can be controlled by at most one agent (though it may be affected by more than one agent). Furthermore, if agent i can control a feature f_{ix}, then that feature must be represented in i's local state (Definition 3.4). These properties induce a further decomposition of the state features within each agent's local state:

Definition 3.12 (**Local State Constituents**)**.** Agent i's local state s_i is comprised of three disjoint feature sets, $s_i = \left\langle \bar{u}_i, \bar{l}_i, \bar{n}_i \right\rangle$, where:

- i's **unaffectable features** $\bar{u}_i = \langle u_{i1}, u_{i2}, ... \rangle$ are those features that are not affectable by any agent, but may be observable by multiple agents. Examples include *time-of-day* or *temperature*.

- i's **locally-controlled features** $\bar{l}_i = \langle l_{i1}, l_{i2}, ... \rangle$ are those features that are controllable by agent i. Features $\bar{l}_i$ are not controllable by any other agent. For example, a rover's position is a locally-controlled feature. Additionally, $\bar{l}_i$ may contain features that are not controllable by any agent, but that are affectable by agent i and are not contained within any other agent's locally-controlled feature set.[6]

[6]Special care must be taken in treating features that are not controllable by any agent but affectable by multiple agents. For instance, an agent pushes the first domino, and 10 time steps later the last domino falls. In this case, the "last-domino-down" feature could be included in agent i's locally-controlled feature set. However, if there are any other agents that can affect this feature, these other agents must model it as a *nonlocal feature* (see below).

- i's **nonlocal(ly-controlled) features** $\bar{n}_i = \langle n_{i1}, ... \rangle$ are those remaining features, each of which is in the locally-controlled feature set of exactly one other agent, and whose values may affect the transitions of i's locally-controlled features (formalized in Equation 3.9).

According to the composition of agents' local states given in Definition 3.12, there may exist world state features that are not modeled exclusively by a single agent. First, a feature may appear as an *unaffectable features* in more than one agent's local state. Second, each *nonlocal feature* in agent i's local state appears as a *locally-controlled feature* in the local state of exactly one other agent. In the example from Figure 3.1, the rover models whether or not the satellite agent has planned a path for it, so *path-A-planned* would be a nonlocal feature in the rover's local state as well as a locally-controlled feature in the satellite's local state. I refer to such features as *mutually-modeled features*. Conceptually, mutually-modeled features are aspects of the environment that are relevant to more than one agent as they plan their decisions.

Definition 3.13. Agent i's **mutually-modeled features**, $\bar{m}_i$, are those state features that appear in i's local state representation s_i, as well as one or more other agents' local state representations:

$$\bar{m}_i \equiv \langle f \in s_i | \exists j \neq i, f \in s_j \rangle \tag{3.7}$$

Mutually-modeled features make the TD-POMDP model *transition dependent*. Referring back to Definition 2.9, transition independence is violated when the change in value of *nonlocal feature* $n_{jx} \in \bar{n}_j$ in agent j's local state depends upon agent i's action:

$$Pr\left(n_{jx}^{t+1} | s^t, a_i, a_j\right) \neq Pr\left(n_{jx}^{t+1} | s^t, a_i', a_j\right) \tag{3.8}$$

The transition dependencies that exist between TD-POMDP agents are structured as follows. Within agent j's local state $s_j = \langle \bar{u}_j, \bar{l}_j, \bar{n}_j \rangle = \langle f_{j1}, ..., f_{jk} \rangle$, the transition probability of any feature $f_{jx} \in s_j$ at time $t+1$, denoted f_{jx}^{t+1}, given that the agents have just performed joint action $a^t = \left\langle a_1^t, ..., a_j^t, ... a_n^t \right\rangle \in A$ in world state $s \in S$ at time t is:

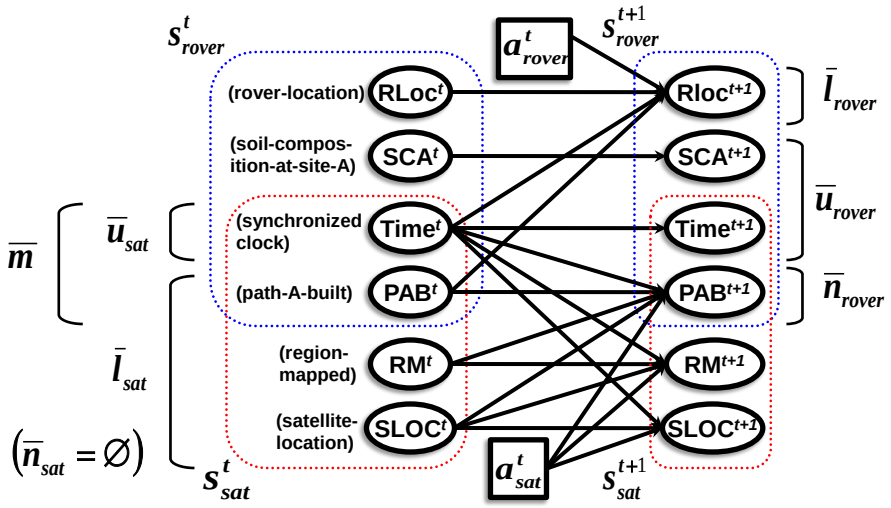


Figure 3.4: Example of the dependencies among feature transitions.

$$Pr\left(f_{jx}^{t+1}|s^t, a^t\right) = \begin{cases} Pr\left(f_{jx}^{t+1}|\bar{u}_j^t\right) & \text{for unaffectable feature } f_{jx} \in \bar{u}_j \\ Pr\left(f_{jx}^{t+1}|s_j^t = \left\langle \bar{u}_j^t, \bar{l}_j^t, \bar{n}_j^t \right\rangle, a_j^t\right) & \text{for locally-controlled feature } f_{jx} \in \bar{l}_j \\ Pr\left(f_{jx}^{t+1}|s_i^t, a_i^t\right) & \text{for nonlocal feature } f_{jx} \in \bar{n}_j \\ & \quad \text{(locally-controlled by agent } i\text{)} \end{cases} \tag{3.9}$$

Equation 3.9 indicates that the transitions of all unaffectable features and locally-controllable features depend on only the local state and local action. However, the transitions of each nonlocal feature depend on world features outside of the local state and on the actions of exactly one other agent (i, for instance), respectively. In Figure 3.4, these dependence relationships are represented graphically with a two-stage DBN for the running example problem (Ex. 3.3). Here, the features are grouped by agent as well as by feature type, with the mutually-modeled features labeled $\bar{m}$. The semantics of the DBN are such that a particular feature is conditionally independent of all non-parent features conditioned on the feature's parents. Although this particular DBN is specific to the example problem, notice that its conditional independencies (denoted by the absence of arrows) conform to Equation 3.9.

Additionally, as I formalize in Equation 3.10, the values for the three groups of features $\left\{\bar{u}_j, \bar{l}_j, \bar{n}_j\right\}$ are conditionally independent of one another given the previous

state and joint action. Bringing the terms from Equation 3.9 together and generalizing to multiple nonlocal features (dependent on one or more other agents) leads to a formal definition of the TD-POMDP's factored transition function.

Definition 3.14. Agent j's **factored local transition function** is the probability distribution over j's next local state conditioned on world state and joint action:

$$\begin{aligned} Pr\left(s_j^{t+1}|s^t,a^t\right) &= \underbrace{Pr\left(\bar{u}_j^{t+1}|\bar{u}_j^t\right) Pr\left(\bar{l}_j^{t+1}|\bar{l}_j^t,\bar{n}_j^t,\bar{u}_j^t,a_j^t\right)}_{\text{locally-dependent component}} \underbrace{Pr\left(\bar{n}_j^{t+1}|s^t,\left\{a_{\neq j}^t\right\}\right)}_{\text{nonlocally-dependent component}} \\ &= P_j^U\left(\bar{u}_j^{t+1}|\bar{u}_j^t\right) P_j^L\left(\bar{l}_j^{t+1}|s_j^t,a_j^t\right) \prod_{\forall i|\exists \bar{l}_{ix}\subset\bar{l}_i\wedge\bar{l}_{ix}\subseteq\bar{n}_j} P_i^L\left(\bar{n}_{jx}^{t+1}\equiv\bar{l}_{ix}^{t+1}|s_i^t,a_i^t\right) \end{aligned} \tag{3.10}$$

denoted as the product of j's **unaffectable feature transition function** $P_j^U()$, j's **locally-controlled transition function** $P_j^L()$, and other agents' locally-controlled feature transition probabilities.

Equation 3.10 factors the transition of j's local state features, explicitly distinguishing between the features dependent on previous values of the local state (and local action) and features dependent on nonlocal state and action. Moreover, the TD-POMDP model explicitly specifies the locally-dependent factored transition function components $P_j^U()$ and $P_j^L()$. As shown, agent j's nonlocally-dependent components are encoded in the locally-dependent components of other agents.

The result of this factorization is a structured transition dependence whereby agents may affect the consequences of each others' actions sequentially *but not concurrently*. An example of this is depicted graphically in Figure 3.5, where agent i may affect the value of one of agent j's nonlocal state features and agent j's subsequent (but *not* simultaneous) locally-controlled feature transitions are influenced by the new value. I defer a discussion of the limitations of non-concurrent interactions to Section 3.4.3.1.

The name "Transition-Decoupled" POMDP comes from the advantage of non-concurrent interaction effects: restricting interactions in this manner enables a decoupling of the TD-POMDP into efficiently-solvable local models (as I develop in Chapter 4). In essence, by assigning control of each feature to a single agent, and disallowing concurrent control, agents avoid having to reason about each others' simultaneous actions.

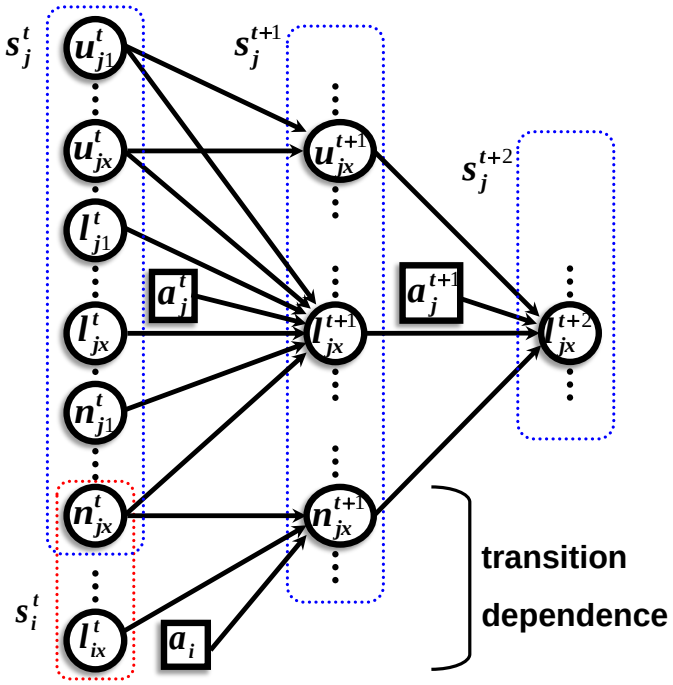


Figure 3.5: DBN illustrating the TD-POMDP's structured transition dependence.

3.2.3 Temporal Synchronization

Included as a key feature of TD-POMDP world state is *time*, which is an *unaffectable feature* with deterministic transitions, and which is mutually-modeled by all agents. This feature serves to synchronize agents' executions and to practically facilitate coordination, particularly in domains where frequent communication is not possible. Typically, $time = 0$ in the TD-POMDP start state. Similarly, successor states always have a larger *time* value than their predecessors. A side effect is that the state space is non-recurrent: no world state may be visited more than once over the course of a single execution.[7]

3.2.4 Decoupled Representation

The preceding subsections (3.2.1–3.2.3) described the TD-POMDP in the context of the conventional Dec-POMDP specification, formalizing the structural properties that delineate the TD-POMDP as a proper Dec-POMDP subclass, and along the way introducing the essential structural components. In Section 3.4, I provide a detailed discussion of the expressiveness of the TD-POMDP along with its representational

[7]My approach is also applicable to problems with recurrent state spaces, but I do not consider those problems in this thesis.

limitations that come with this added structure. Here I summarize the compilation of components that specifies a TD-POMDP problem.

Definition 3.15. A **TD-POMDP** $\mathcal{M}$ is specified by the following tuple: $\mathcal{M} = \langle \mathcal{N}, \{S_j\}, \{A_j\}, \{\Omega_j\}, \{O_j\}, \{R_j\}, \{\bar{m}_j\}, \{P_j^U\}, \{P_j^L\}, T \rangle$, where

- $\mathcal{N}$ is a team of n TD-POMDP agents, indexed by j.
- $S_j \subseteq U_j \times L_j \times N_j$ is agent j's *local state* space (Def. 3.4), which is (possibly a subset of) the cross product of *unaffectable*, *locally-controlled*, and *nonlocally-controlled* feature spaces (Def. 3.12), with explicitly distinguished initial state s_j^0;
- A_j is j's *local action* space (as in Def. 2.3);
- Ω_j is j's *local observation* space (as in Def. 2.3);
- $O_j : A_j \times S_j \times \Omega_j \mapsto [0, 1]$ is j's *local observation function* (Def. 3.5);
- $R_j : S_j \times A_j \mapsto \mathbb{R}$ is j's *local reward function* (Def. 3.6);
- $\bar{m}_j$ is the set of j's *mutually-modeled features* (Def. 3.13), where each feature is explicitly associated with at least one other agent;
- $P_j^U : U_j \times U_j \mapsto [0, 1]$ is the *unaffectable feature transition function* (Def. 3.14);
- $P_j^L : S_j \times A_j \times L_j \mapsto [0, 1]$ is the *locally-controlled feature transition function* (Def. 3.14); and
- $T \in \mathbb{N}$ is the finite *time horizon* of execution (as in Def. 2.3).

Unlike the conventional Dec-POMDP specification, most of the TD-POMDP model information is inherently distributed. For instance, instead of representing the world state space with a set S, the TD-POMDP explicitly specifies individual local state spaces. S may be recovered by aggregating all of the local state spaces (though with mutually-modeled features, S is *not* simply a cross product of local state spaces $\times_{1 \leq j \leq n} \{S_j\}$). Transition, observation, and reward information is similarly distributed into local components. The TD-POMDP model also distinguishes those local state features that are mutually modeled, explicitly characterizing the type of feature as well as the controlling agent (unless unaffectable). In fact, explicit in the TD-POMDP specification is the notion of a *local model* that represents the dynamics of the world as they relate to an individual agent j.

Definition 3.16. Agent j's **local model** $\mathcal{M}_j$ for TD-POMDP $\mathcal{M}$ is specified by tuple $\mathcal{M}_j = \langle S_j, A_j, \Omega_j, O_j, R_j, \bar{m}_j, P_j^U, P_j^L, T \rangle$, where each component is taken from the joint model $\mathcal{M}$.

Representing the joint decision model as a collection of local models has several advantages. First, for problems involving agents that each deal (primarily) with a different realm of their shared environment, it is natural to decompose the model information as such. Specifying joint versions of the components (i.e., joint state, joint action, joint transition, joint observation) would involve unnecessary aggregation of largely-independent agent dynamics. Further, the space required to store the aggregated information (when naïvely represented such as in the conventional Dec-POMDP specification) could be unnecessarily large. Instead, the TD-POMDP factors the model information so as to break up an otherwise very large joint state space, joint transition matrix, and joint observation matrix into potentially more compact local components.

However the aforementioned advantages of the TD-POMDP begin to disappear as agents' mutually-modeled feature sets grow large relative to the number of features in the world state. The more features are shared, the more information will be duplicated in the TD-POMDP agents' local models. Consequently, the TD-POMDP representation is most appropriate for weakly-coupled problems (detailed in Section 3.5) with a relatively low density of nonlocal features.

At first glance, agent j's local model $\mathcal{M}_j$ closely resembles a single-agent POMDP (defined in Section 2.2.2). However, it is important to note that $\mathcal{M}_j$, when studied in isolation from $\mathcal{M}$'s other local models, does not constitute a proper POMDP. In general, $\mathcal{M}_j$ will include *nonlocal features* whose transitions depend on other agents' behavior. $\mathcal{M}_j$ does not include information about the transition probabilities of these features. However, even if $\mathcal{M}_j$ were to include nonlocal feature transition information, the local model would not constitute a POMDP due to the non-Markovian dynamics. Recall that nonlocal features may depend on features not modeled in j's local state, and on other agents' actions, that may in turn depend on histories of features in the local state. As such, the agents' local models are tied to one another by the transition dependencies of their nonlocal features. Only in the absence of nonlocal features does $\mathcal{M}_j$ define a proper POMDP.

However, as the name "(T)ransition (D)ecoupled POMDP" implies, the local models *can* be decoupled from one another and be made into independently-evaluable decision models. As developed formally in Chapter 4, decoupling is accomplished by holding peer agents' policies fixed and abstracting the transition influences. Once

decoupled, the local models can be used to plan individually in the context of a distributed solution methodology (developed in Chapter 6).

3.3 Optimality and Tractability

Next, I discuss what it means to solve the TD-POMDP (in Section 3.3.1) and express the worst-case time complexity of solving it (in Section 3.3.2). The result is forbidding: just like the general Dec-POMDP, in the worst case, computing an optimal policy for a problem in the TD-POMDP subclass is intractable. However, this complexity result does not mean that the TD-POMDP should be condemned as a model that is less general than the Dec-POMDP and just as impractical. The merit of the TD-POMDP lies in its emphasis and explicit representation of problem structure, which I summarize in Section 3.3.3. By exploiting TD-POMDP structure, we will see (theoretically in Section 3.5 and empirically in Chapters 4 and 6) that portions of the TD-POMDP space yield efficiently-computable solutions.

3.3.1 Solution Concept

The solution concept that I adopt in this dissertation is that of maximizing expected value. Thus, (optimally) solving a TD-POMDP problem involves computing a set of agent policies that maximizes the team's expected cumulative reward (Def. 2.6). This set is referred to by the Dec-POMDP community as the optimal joint policy, whose definition I now restate.

Definition 3.17. An **optimal joint policy** π^* of a TD-POMDP $\mathcal{M}$ is a combination of local policies $\left\langle \forall i, \pi_i : \vec{O}_i \mapsto A_i \right\rangle$, each of which assigns an agent's local action to each of its local observation histories (as per Definition 2.4), that maximizes the joint value function: $\pi^* \in \underset{\pi}{\mathbf{arg}\max}\, V(\pi)$.

Henceforth, I use the term *solution* and *optimal joint policy* interchangeably, and *solving* a TD-POMDP problem to mean *optimally solving* it by computing an optimal joint policy. For certain problems, optimality may not be tractable. I refer to an *approximate solution* as a joint policy that is returned by some planning algorithm, but that is not guaranteed to be optimal. When optimal solutions are intractable, the next best thing is to compute quality-bounded approximate solutions, which are those that attain a solution quality (i.e., value) that is provably close (by some standard of proximity) to the optimal solution quality. However, computing quality-bounded approximate solutions may also be intractable, and provably as hard as computing

optimal solutions (as is the case for the general finite-horizon Dec-POMDP problem class (Rabinovich et al., 2003)). As such, I will use the term *approximate method* to refer to any solution method not guaranteed to return optimal solutions, regardless of whether or not it produces quality-bounded solutions.

Note that optimality is defined in relation to, and is sensitive to, the problem specification $\mathcal{M}$. That is, a policy is optimal if it maximizes the expected team utility given each agent's representation of features in its local state, constraints implied by its local observation function, and so on. If two problems, specified by $\mathcal{M}^a$ and $\mathcal{M}^b$ respectively, differ only in one agent's observability of one feature, a joint policy π^* that is optimal for $\mathcal{M}^a$ may be suboptimal for $\mathcal{M}^b$. Moreover, for two problems with slightly different specifications, computing the solution to one may take seconds, but computing the solution to another may take hours. I refer the reader to a classical example comparing the Dec-POMDP model with the Multiagent MDP (MMDP) model (Boutilier et al., 1999), which is a Dec-POMDP having the property that each agent observes the complete world state. Whereas the worst-case time complexity of the MMDP class is polynomial in the size of the state space (Goldman & Zilberstein, 2004), the worst-case time complexity of the Dec-POMDP class is NEXP-complete, even if we restrict consideration to Dec-POMDP problems where agents' observations collectively (though not individually) determine the world state (Bernstein et al., 2002). The intuition behind why *collectively-observable* problems are so much harder to solve (than *completely-observable* problems) is that, when agents receive different (though not independent) observations, optimal joint behavior requires each agent to reason about (the exponentially-growing space of) what other agents' may have observed in addition to what the agent itself observes.

3.3.2 General Complexity

I now turn to the worst-case time complexity of computing optimal solutions for the class of TD-POMDP problems. The class of TD-POMDP problems is a subclass of the finite-horizon Dec-POMDP class (as emphasized in Section 3.2), so its complexity can be no greater than that of the Dec-POMDP. Further, the TD-POMDP imposes several structural restrictions on top of the Dec-POMDP model, so one might expect it to have a worst-case complexity strictly lower than that of the Dec-POMDP. Building on the work of others (Becker et al., 2004a; Allen, 2009), I have derived that this is unfortunately not the case. Instead, the TD-POMDP's worst-case complexity has the same asymptotic lower-bound and upper-bound as does that of the more general Dec-POMDP.

Theorem 3.18. *The TD-POMDP is NEXP-complete.*

Proof. The NEXP-hard lower bound follows directly from the reduction of the EDI-Dec-MDP (Becker et al., 2004a), proved to be NEXP-complete (Allen, 2009), to the TD-POMDP. I present the reduction in Appendix A. The NEXP upper bound is proven given that the TD-POMDP is subclass of the Dec-POMDP (and was formally specified as such in Section 3.2). Therefore, the TD-POMDP is NEXP-complete. □

The fact that the TD-POMDP is in the same complexity class as the Dec-POMDP suggests that, although the TD-POMDP requires additional problem structure beyond that of the Dec-POMDP, it does not strongly constrain the problems that may be represented (an issue discussed in detail in Section 3.4). The TD-POMDP can represent problems that are (asymptotically) just as hard.

To be precise, the complexity result given in Theorem 3.18 is a statement relating problem description size to the worst-case computation required to verify that a solution is optimal. For NP-complete problems, such verification requires a number of computations polynomial in the problem size. For NEXP-complete problems like the TD-POMDP, verification requires a number of computations that is irreducibly exponential in the problem size. Thus, computation time of optimal solutions to NEXP-complete problem requires at least exponential time in the problem size, though this lower bound relies on the successful reduction of NEXP to EXP. It is widely believed (Papadimitriou, 1994) that NEXP and EXP are distinct complexity classes, and that solving NEXP-problems requires time *doubly exponential* in the problem size. For more information on this issue, I refer the reader to the theses of Daniel Bernstein (2005) and Martin Allen (2009).

Observation 3.19. *The worst-case computation time of optimal solutions for TD-POMDP problems is believed to be doubly exponential in the size of the problem description.*

Theorem 3.18 and Observation 3.19, by themselves, do not say anything about how the computation time of TD-POMDP solutions relates to the number of agents in the system. Discussion of Dec-POMDP complexity in the literature is centered around the complexity of two-agent problems, wherein the size of a problem's specification is commonly interpreted (Allen, 2009; Bernstein, 2005) as some function of the state or observation space, the action space, and the time horizon T. The focus of this dissertation is on scaling solution computation to problems with more than two agents. In this context, let us be more concrete about the ramifications of the NEXP complexity result.

Observation 3.20. *If the size of the TD-POMDP description* $\|\mathcal{M}\| \propto n$*, the TD-POMDP's worst-case computation time is believed to be doubly exponential in the number of agents* n*.*

The assumption that the problem description grows with the number of agents is not terribly restrictive. By placing more agents in a shared environment, the number of combinations of agent circumstances (each specified by a state in the state space) should increase linearly if not exponentially. Linear growth in the state space translates to doubly-exponential worst-case solve times (by Observation 3.20).

All of these complexity results should be taken with a grain of salt. For instance, from Observation 3.20, it is tempting to conclude that the class of problems that I consider in this dissertation, for which I endeavor to scale solution computation to many agents, do not admit tractable scalable solution methods, and that such a pursuit is doomed to fail. However, recall that these are worst-case bounds. The TD-POMDP's ability to represent problems with doubly exponential solve times does not imply that all TD-POMDP problems that we face will have this property. I prove in Section 3.5 and demonstrate in Chapter 6 the existence of sets of problems that avoid such menacing complexity and scale efficiently to large teams of agents. At the meta-level, I have introduced the TD-POMDP model in order to provide the formal specification of an umbrella class *and* the language with which to characterize its underlying subspace.

3.3.3 Significance of Structure

The significance of the TD-POMDP lies in its emphasis of structure in problems that can be exploited to yield tractable solutions. I now briefly summarize the key elements of exploitable structure, referencing the details of their exploitation that appear in subsequent sections of this dissertation.

Decoupled Representation. The TD-POMDP's wholly factored representation decouples the joint model $\mathcal{M}$ into a set of local models $\{\mathcal{M}_i\}$ that are tied together by the transition dependencies of agents' nonlocal features. Local model $\mathcal{M}_i$ compactly specifies the subset of problem dynamics relevant to agent i's own behavior. When augmented with information relating to other agents' choices (as I detail in Section 4.2.2), $\mathcal{M}_i$ suffices as a complete local planning model that i can use to compute its own policy without having to reason about superfluous details of other agents' behavior. Hence, the benefit of the TD-POMDP's decoupled representation is that it enables

efficient individual reasoning in the context of a distributed joint policy search.[8]

Locality Of Interaction. Through its explicit representation of mutually-modeled features $\{\bar{m}_i\}$ (Def. 3.13), the TD-POMDP specification emphasizes the team of agents' *locality of interaction*; each agent has a subset of other agents that it interacts with and a subset of features through which it interacts. In Section 3.5.1, by accounting for locality of interaction in the TD-POMDP, I derive a tighter complexity bound than that last derived in Section 3.3.2. Additionally, in Chapter 6, I present an algorithm that achieves the subsequent reduced complexity by exploiting this structure.

Distinguished Interaction Features. Within agents' mutually-modeled feature sets, the TD-POMDP distinguishes agents' nonlocal features $\{\bar{n}_i \subseteq \bar{m}_i\}$ (Def. 3.12). It is through these nonlocal features that agents can influence one another. Moreover, these features' transitions constitute the information that TD-POMDP agents must jointly address when coordinating their behaviors. Instead of coordinating over whole policies, agents can coordinate over compact abstractions, which I term *influences*, that encode nonlocal feature transition information. Before developing a methodology for influence-based policy abstraction in Chapter 4, I derive in Section 3.5.2 that, utilizing such an abstraction can yield a potentially substantial reduction in complexity (on top of the reduction achieved by exploiting locality of interaction).

By leveraging each of the above structural characterizations, I demonstrate in this dissertation that, despite its intractable general complexity, the TD-POMDP's landscape of representable problems contains rich regions whose solutions are efficiently computable. In Section 3.5.1, I explore and characterize the landscape.

3.4 Expressiveness of the Representation

Having presented the formal details of the TD-POMDP model along with its worst-case computational complexity, I now discuss the expressiveness of the TD-POMDP specification. In particular, I address two issues relating to expressiveness: (1) the space of problems that can be represented in the model, and (2) the problem structure that the model's specification makes explicit.

Section 3.2 began with the well-established, general Dec-POMDP model and introduced several additional structural properties, each one potentially carving away

[8]The degree to which individual reasoning is efficient depends upon a structural metric *state factor scope* that I describe in Section 3.5.

portions of the overarching Dec-POMDP class. What we are left with is a Dec-POMDP subclass, the TD-POMDP, that is the focus of this thesis. With the progression from Dec-POMDP to TD-POMDP, there is the danger that, even though each additional restriction individually appears reasonable, the combination of additional properties narrows the representation to a space of problems that is no longer interesting.

In response to these concerns, I contend that the TD-POMDP class—although it comes with some inherent restrictions—remains an interesting space, and that the TD-POMDP model specification conveys a rich array of useful problem structure. I defend this claim in Section 3.4.1 by contrasting my class with a variety of problem classes studied by others, for many arguing (and for some proving) that the TD-POMDP is a more general class and for others arguing that the TD-POMDP specification expresses just as much (if not more) exploitable problem structure. I extend my comparison in Section 3.4.2 to models explicitly representing communication, incorporating a discussion of how to model agent communication in the TD-POMDP. Examining the relationship of the TD-POMDP with existing models exposes its limitations. In Section 3.4.3, I summarize the key representational restrictions and provide suggestions of how they might be overcome in future work so as to expand the TD-POMDP (and the methodologies presented in this dissertation) to a broader sphere of problems.

3.4.1 Comparison with Existing Models

Prior to the work presented in this dissertation, researchers have introduced a variety of other Dec-POMDP subclasses (reviewed in Section 2.3.2). At a high level, the motivation for defining these subclasses has been the same as mine: *to identify problem structure that can be exploited by solution methods to yield solutions tractably.* In some cases, the structure has allowed for impressive scaling of optimal solution methods to systems of several agents (Beynier & Mouaddib, 2005; Marecki & Tambe, 2007; Nair et al., 2005; Varakantham et al., 2007). However, in such cases, not only was problem structure identified, but the problem representation was also severely restricted from that of the Dec-POMDP. Moreover, in such cases, generality was lost to the extent that some Dec-POMDP problems with closely-related structure could no longer be represented given the constraints of their specifications.

Other subclasses have sought to identify structure with little or no loss of generality of the representation (Oliehoek et al., 2008b; Varakantham et al., 2009), but as-of-yet have not achieved both scalability (to more than three agents) and optimality. This is due, in part, to the unavoidable complexity that comes with considering a general space of Dec-POMDPs. However, I posit that another reason is that the

specifications of these latter subclasses do not articulate enough identifiable structure whose exploitation would allow scalability of optimal solution methods.

Inspired by the successful scaling achieved by the former group of (exploitable, yet restrictive) subclasses, I have collected what I see as the most useful elements of problem structure (amenable to tractable scalability and optimality) and have incorporated those into the TD-POMDP's specification. However, I have done so in such a way as to impose as few restrictions as possible on the problem representation. In this sense, the TD-POMDP strives for a balance in its articulation of exploitable structure and its loss of generality.

In the subsections that follow, I enumerate the most closely related Dec-POMDP subclasses. For each, I contrast its representational restrictions, as well as the elements of exploitable structure that it articulates (in the context of the problems that it can represent), with those of the TD-POMDP.[9] The results are summarized in Table 3.1, whose rows are ordered roughly from the least general model to the most general model. The second and third columns indicate the "representational restrictions" and "exploitable structure" of each of the eight models. So as to provide a very rough overview of the scalability of solution methods of each class, I have included a "scalability results to date" column, populated with data taken from published empirical demonstrations of solution computations to problems of more than two agents.[10] For cases in which only approximate solutions were computed, I prefix the scalability results accordingly (giving priority to results involving quality-bounded solutions if available).

Table 3.1 shows that, based on my analysis, the only problem classes that are more general than the TD-POMDP have not been shown to scale optimally beyond three agents. Additionally, many of the classes that are less general than the TD-POMDP

[9]I have attempted to remain objective in my comparison of subclasses to the extent possible. However, given that I am characterizing a diverse collection of models in like terms according to my own dimensions (which were not necessarily those used by the authors of the respective works I am characterizing), a degree of subjectivity is bound to have crept in. For a more complete description of each model, along with the biases of its creators, see the respective papers I reference.

[10]The "scalability results to date" are not meant to be compared to each other at face value. Each result represents solution computation on a different set of problems. Some results were published years before others. Although most classes of problems are defined around a set of agents, the OC-Dec-MDP is defined around a network of methods (which could perhaps be distributed among agents), as are the corresponding scalability results. Furthermore, developers of TI-Dec-MDP, EDI-Dec-MDP, and EDI-CR algorithms emphasized aspects of performance other than scalability, and did not extend their algorithmic implementations to more than two agents (Becker et al., 2004a,b; Mostafa & Lesser, 2009; Petrik & Zilberstein, 2009). Nevertheless, this column provides a coarse overview of published achievements that motivates the development of scalable TD-POMDP solution methods.

Model	**Representational Restrictions**	**Exploitable Structure**	**Scalability Results To Date**
OC-Dec-MDP	hierarchy of methods with fixed execution ordering, LFO	precedence relationships among methods	(approximate) 100+ methods
TI-Dec-MDP	TOI, LFO, structured utility decomposition	decoupled representation, locality of interaction	2 agents
ND-POMDP	TOI, structured utility decomposition	decoupled representation, locality of interaction	(quality-bounded approximate) 7 agents
EDI-Dec-MDP	LFO, RI, event-driven NIE, structured utility decomposition	decoupled representation, explicit interaction features, locality of interaction	2 agents
EDI-CR	LFO, event-driven NIE, structured utility decomposition	decoupled representation, explicit interaction features, locality of interaction	2 agents
TD-POMDP	NIE, structured utility decomposition	decoupled representation, explicit interaction features, locality of interaction	*See Chapter 6*
DPCL	OI, structured utility decomposition	decoupled representation, explicit interaction features, locality of interaction	(approximate) 10 agents
Factored Dec-POMDP	–	locality of interaction	3 agents

(LFO=*local full observability*, TOI=*transition and observations independence*, RI=*reward independence*, OI=*observations independence*, NIE=*nonconcurrent interaction effects*)

Table 3.1: Comparison of Dec-POMDP subclasses

have not been shown to scale beyond two agents. We also see that, with the exception of *precedence relationships among methods*, the TD-POMDP specification expresses the same types of exploitable structure as do any of the other subclasses. This result suggests that the TD-POMDP model defines a sweet spot where other models were too restrictive, not optimally scalable, or lacking in explicit representation of useful problem structure.

3.4.1.1 Opportunity Cost Dec-MDP (OC-Dec-MDP)

The OC-Dec-MDP (Beynier & Mouaddib, 2005; Marecki & Tambe, 2007) is a model that exploits specialized structure for planning the execution times of agents' activities, called methods. Strictly less general than the TD-POMDP, its representation is restricted in the following ways. First, the OC-Dec-MDP specifies a fixed ordering over agents' method executions, restricting the problem to one of determining only when to start each method. More general models such as the TD-POMDP can also represent the problem of determining which method to execute. Second, the observation function of the OC-Dec-MDP takes a special form such that each agent observes exactly the status of its own method executions. This particular observational restriction is a special case of *local full observability* (LFO) (Def. 2.8). The TD-POMDP observation function is less restrictive, allowing for *partial* observations that may depend upon the values of any features (locally-controlled or nonlocally-controlled) in their local states.

The only kind of interaction that can be represented by an OC-Dec-MDP is a precedence constraint dictating that a method executed by one agent will only complete successfully if a particular method of some other agent has already completed. This is strictly more restrictive that the TD-POMDP, which can represent more complex method dependencies through the specification of nonlocal features. Nevertheless, the OC-Dec-MDP's method precedence relationships constitute powerful, though specialized, problem structure that researchers have exploited to compute approximate solutions to problems containing over a hundred methods (Marecki & Tambe, 2007).

3.4.1.2 Transition-Independent Dec-MDP (TI-Dec-MDP)

The TI-Dec-MDP (Becker et al., 2004b), like the TD-POMDP, represents factored state dynamics that are exploited so as to decouple the joint model into local decision models with local states, local transitions, and local rewards. Unlike the TD-POMDP, the TI-Dec-MDP assumes *local full observability* (Def. 2.8), restricting an agent to observe its local state features exactly at every time step. Furthermore, the TI-Dec-

MDP requires *transition and observation independence* (Defs. 2.9–2.10), restricting an agent's local state transitions to be independent of other agents' local states and local actions. This property puts the TI-Dec-MDP in a different complexity class (NP instead of NEXP) than more general flavors of Dec-POMDPs (Allen, 2009). With the TD-POMDP model, local state transitions (as well as local observations) can depend on other agents' states and actions by way of structured nonlocal feature dependencies (Sec. 3.2.2). Alternatively, the TI-Dec-MDP models dependencies in the reward function, whereby particular combinations of agents' actions can result in additional reward or penalty.[11]

Like the TD-POMDP's nonlocal feature dependencies, the TI-Dec-MDP's reward dependencies emphasize agents' *locality of interaction* (a concept that I develop formally in Section 3.5.1), leading to a decomposition of the team's value function that can be exploited to yield efficiently-computable optimal solutions for problems with two agents (Becker et al., 2004b; Petrik & Zilberstein, 2009) and likely more, but scaling of TI-Dec-MDPs has never been demonstrated empirically. In contrast to the TD-POMDP, whose reward is composed of a summation of local rewards, the TI-Dec-MDP's reward consists of a summation of local rewards and of special reward dependency terms that account for agents' joint actions.

3.4.1.3 Network-Distributed POMDP (ND-POMDP)

The ND-POMDP (Nair et al., 2005) is also less general than the TD-POMDP in that it requires *transition and observation independence* like the TI-Dec-MDP, but unlike the TI-Dec-MDP it does not require local full observability. Moreover, instead of modeling individual reward dependencies, the ND-POMDP specifies a particular decomposition of the team utility into *local neighborhood* utilities. This utility structure, as with the TD-POMDP's transition dependency structure, enables an exploitation of *locality of interaction* by explicitly representing interactions among groupings of agents. Exploiting the ND-POMDP's locality of interaction along with its factored, decoupled representation has led to efficient optimal solution methods (Nair et al., 2005) and impressive scalability of quality-bounded solutions to teams of 7 agents (Varakantham et al., 2007; Marecki et al., 2008), not to mention efficiently-computed (unbounded) approximate solutions to even larger agent teams (Kumar & Zilberstein, 2009; Marecki et al., 2008).

[11]The TD-POMDP too can represent reward dependency, where agent i's actions affect agent j's rewards, by (1) modeling two versions of each reward-dependent local state in j's local state space, (2) assigning separate local rewards to transitions into these states, and (3) modeling a special nonlocal feature controlled by i that drives j's transitions into these states.

3.4.1.4 Dec-MDP with Event-Driven Interactions (EDI-Dec-MDP)

The EDI-Dec-MDP (Becker et al., 2004a), whose description I elaborate in Appendix A, is perhaps the most closely-related model to the TD-POMDP. Like the TD-POMDP, the EDI-Dec-MDP decouples the joint model into (largely independent) local models tied together by structured transition dependencies that are made explicit by the problem specification. Where the EDI-Dec-MDP differs is in its representation of the features involved in the transition dependencies. For an agent i whose transitions affect agent j, each EDI-Dec-MDP dependency relates the occurrence of an *event*, which is a transition of agent i's local state, to a dependent transition of agent j's local state. The dependency is modeled by agent j using an unobservable boolean variable denoting whether or not the event has occurred, which can be viewed as a special type of nonlocal feature (Def. 3.12) whose dynamics and observability are more restricted than those of the TD-POMDP. Like the TD-POMDP, this dependency involves *nonconcurrent interaction effects* (a concept introduced in Section 3.2.2 and formalized in Section 3.4.3.1).

The EDI-Dec-MDP also requires *local full observability* (such that all locally-controlled features are exactly observed), and *reward independence* (such that an agent's rewards are independent of all other agents' actions conditioned on the values of its locally-controlled features).[12] Consequently, the EDI-Dec-MDP is less general than the TD-POMDP (as I derive in Appendix A) and also, for some problems that it can represent, more awkward to specify. In particular, for problems with temporally-uncertain interactions, the TD-POMDP can be specified with a single boolean nonlocal feature per interaction whereas the EDI-Dec-MDP requires several boolean dependency features (one for each time at which the interaction could occur) per interaction. Though its specification of dependencies captures some degree of *locality of interaction*, EDI-Dec-MDP solution methods have not been shown to scale beyond two agents (Becker, 2006).

3.4.1.5 Event-Driven Interactions with Complex Rewards (EDI-CR)

The EDI-CR model (Mostafa & Lesser, 2009), described by its authors as a hybrid of the TI-Dec-MDP and the EDI-Dec-MDP, represents both reward dependencies and event-driven transition dependencies. I argue that it is no more general than the EDI-Dec-MDP because reward dependencies could be implemented as transition

[12] In contrast to decomposable joint value, *reward independence* is stronger in that it requires the individual local reward components to be independent on one another.

dependencies by folding the additional rewards or penalties into transition-dependent local state outcomes.[13] Further, it is more restrictive in its *local full observability*, and suffers from the same representational disadvantages as the EDI-Dec-MDP with respect to event-driven nonconcurrent interaction effects. State-of-the-art solution methods for the EDI-CR model support scaling to teams of more than two agents (Mostafa & Lesser, 2009) in theory, however, this has not yet been demonstrated empirically.[14]

3.4.1.6 Distributed POMDP with Coordination Locales (DPCL)

Like the TD-POMDP, the DPCL (Varakantham et al., 2009) is geared towards exploiting structure and locality of interaction. It is less general than the TD-POMDP in that it requires *observation independence* (Def. 2.10), whereas the TD-POMDP allows an agent's observations to depend on another's actions. However, DPCL is more general than the TD-POMDP in that it represents transition dependencies with concurrent effects using structures that the authors refer to as "same-time coordination locales". The presence of same-time coordination locales appears to complicate optimal planning. As of yet, researchers have not found a way to compute *optimal* best responses efficiently for these problems. Instead, the DPCL has only been shown to afford efficient heuristically-guided approximate solutions with no bounds on solution quality. As I show in later chapters, by excluding problems with same-time coordination locales, the TD-POMDP model can be decoupled into efficiently-solvable, provably-optimal local best response models, thereby achieving both scalability and optimality. Additionally, I suggest in Section 3.4.3.1 how the TD-POMDP might be extended in future work so as to overcome the nonconcurrency restriction.

3.4.1.7 Factored Dec-POMDP

The factored Dec-POMDP, as studied by Oliehoek et al. (2008b), is fully general, capable of representing any Dec-POMDP problem, but additionally allowing exploitation of factored state, transition functions, and value functions. Oliehoek et al. demonstrate how reductions in the scope of each value function component (to a small subset of feature values and a small subset of agents), may be used to decompose and

[13] Generality aside, there may be computational advantages to modeling certain types of reward dependencies with EDI-CR *joint reward structures* as opposed to transition dependencies, though this issue has never been explored in the literature, and is beyond the scope of my analysis.

[14] As I describe when presenting empirical comparisons in Section 6.5, the only available implementation of a EDI-CR solution method is restricted to two-agent problems.

tractably approximate the overall value function. As such, the factored Dec-POMDP specification emphasizes a *locality of interaction* inherent in the structure of the problem. By exploiting this structure, Oliehoek (2010) has demonstrated efficient computation of optimal solutions for 3-agent problems, not to mention computation of (unbounded) approximate solutions to problems with many more agents.

The TD-POMDP is a special case of a factored Dec-POMDP whose factorization has the properties described in Sections 3.2.1–3.2.2. In effect, the factorization of world state into local states and subsequent factorization of local state into locally-dependent and nonlocally-dependent components controls the scope of the TD-POMDP's factored value function. This particular factorization has the benefit of accommodating decoupled efficiently-solvable local best response models (described formally in Section 4.2). However, the factorization requires that the TD-POMDP be restricted to non-concurrent, pairwise agent interactions and a particular decomposition of the joint value function into a summation of local value functions. In Section 3.4.3, I suggest how these restrictions might be relaxed.

3.4.2 Communication

In addition to the models I enumerated in Section 3.4.1, researchers have developed special Dec-POMDP classes that explicitly represent communication actions: the Dec-POMDP-Com (Goldman & Zilberstein, 2003) and Com-MTDP (Pynadath & Tambe, 2002) are notable extensions to the general Dec-POMDP, but there are also communication models that extend some of the subclasses discussed above, such as the ND-POMDP-Comm (Tasaki et al., 2010) and the TI-Dec-MDP-Com (Goldman & Zilberstein, 2004). Using these models, agents plan *what* or when to communicate in addition to planning their usual actions. The purpose of such communication is to exchange information at runtime so as synchronize agents' views. However, there is typically a cost associated with communicating, such that the simple communication policy that always communicates everything to everyone is not necessarily the optimal communication policy.

For the Dec-POMDP-Com and related communicative models, agents are said to employ *direct communication*[15] (Goldman & Zilberstein, 2004) because they are explicitly selecting communication actions that broadcast information over a special communication channel. This is not the only means of exchanging information, however. When agents perform actions that affect each other's observations, this is referred

[15]The concept of *direct communication* is often referred to as "explicit communication", and *indirect communication* as "implicit communication" in other work (Seuken & Zilberstein, 2008).

to as *indirect communication*. For instance, one agent applies the brakes of a car, illuminating a brake light observed by another agent, implicitly communicating that there is a speed trap ahead. It turns out that, from the standpoint of representational power, these two forms of communication are equivalent; even though the Dec-POMDP-Com specifies communication actions and communication observations in addition to the usual Dec-POMDP actions and observations, it is no more general than the Dec-POMDP (Seuken & Zilberstein, 2008).[16]

Likewise, the TD-POMDP implicitly includes agent communication, and is fully capable of representing decisions about what and when to communicate. Here, the channel over which information is communicated between agents is the set of *nonlocal features* (Def. 3.12). Each nonlocal feature is controllable by one agent, the speaker, and (partially) observable to another agent (the listener). Further, the TD-POMDP can model a noisy communications channel by specifying that agents receive only partial observations of these *communication-specific* nonlocal features. Similarly, the TD-POMDP can model communication cost (or lack thereof) through the specification of rewards associated with actions that set the nonlocal features. By instantiating nonlocal features as such, the problem designer may outfit a group of TD-POMDP agents with the desired communication capabilities.[17]

3.4.3 Overcoming Representational Limitations

The comparison of the TD-POMDP with related models (Section 3.4.1) exposed restrictions that make it a less general representation than the subsuming Dec-POMDP class. Specifically, the TD-POMDP requires nonconcurrency of interaction effects and structured local utility decomposition. In the subsections that follow, I reintroduce each restriction and discuss the degree to which it is an inherent limitation of this work (on which the results of this thesis tightly hinge) rather than a detail imposed

[16] Although no more general, models with direction communication include additional structure that may improve efficiency of planning communicative actions.

[17] One could pose the question of what communication capabilities agents *should* be given. This is a question of modeling, and not one that I address directly in this dissertation. Instead, I assume that the TD-POMDP model's state features, observations, and rewards have been determined exogenously. However, the results that I present later on do shed some light on the issue of modeling. In particular, my theoretical analysis presented in Section 3.5 and my empirical results presented in subsequent chapters suggest the following: the more features that agents jointly observe and, by extension, the more communication capabilities that agents are given, the harder the problem of planning becomes. Intuitively, the more information agents share, the more strongly-coupled they are, and the more computationally expensive it becomes to perform optimal decoupled planning and reasoning. On the other hand, reducing agents' communication capabilities may lead to problems whose optimal solutions are of lower value.

for the sake of convenience. I also suggest, for each, how it might be overcome in future work and speculate on the implications of doing so.

3.4.3.1 Concurrent Interaction Effects

In its present form, the TD-POMDP model disallows concurrent interaction effects by requiring that an agent's locally-controlled feature values be independent of the concurrent values of its nonlocal features (conditioned on its latest local action and local state). Depicted graphically in Figure 3.5, and restated mathematically in Equation 3.11 below, this concurrency property is a consequence of the TD-POMDP's factored local transition function (Definition 3.14).

$$Pr\left(\vec{l}_j^{\,t+1} | \vec{n}_j^{\,t+1}, s_j^t, a_j^t\right) = Pr\left(\vec{l}_j^{\,t+1} | s_j^t, a_j^t\right) \tag{3.11}$$

This means that the TD-POMDP cannot represent a problem in which a single feature's value at time $t+1$ is dependent on the actions taken at time t by more than one agent. Each of agent i's locally-controlled features may depend upon only i's latest action (and no other agent's latest action), and each of i's nonlocal features, by definition, depend on at most one other agent's action. Thus, only a single agent initiates each TD-POMDP interaction. The reason that the TD-POMDP imposes this nonconcurrency property is that it facilitates the decoupling of the joint decision model into optimal local decision models.[18]

Example 3.21. A common toy example to illustrate Dec-POMDP dynamics, the *cooperative box pushing* problem (Kube, 1997; Oliehoek, 2010; Seuken & Zilberstein, 2007a), is centered around a concurrent interaction effect. Here, two agents navigate a two-dimensional grid and receive rewards for pushing objects from starting locations to goal locations. Among these objects are large boxes, whose movement requires both agents to push simultaneously from adjacent grid locations. The location of a box thereby depends upon the latest joint action (and not just a single agent's latest action).

[18] As I develop in Chapter 4, optimal decoupling is achieved by augmenting each agent's local model with compact *influence* information. If agents are allowed to concurrently control a given feature, it is unclear how to distill one agent's individual *influence* on that feature and incorporate the respective nonlocal policy information into the other agent's optimal local decision model without exploding the computational complexity of the local model.

Strictly speaking, the TD-POMDP is incapable of modeling concurrent interaction effects. However, this does not rule out the possibility of transforming a problem with concurrent interaction effects into a TD-POMDP problem. For instance, we can, in principle, model the box pushing problem from Example 3.21 as a TD-POMDP using two nonlocal features {*agent-1-pushing*, *box-pushed-by-both-agents*}. The trick is to introduce a 1 time step delay. Consider that agent 1 executes its *push* action at time step t, in turn setting *agent-1-pushing* to true at time $t+1$, and agent 2 executes its *push* action at time step $t+1$, in turn setting *box-pushed-by-both-agents* at time step $t+2$, and causing the box to move. Feature *box-pushed-by-both-agents* does not depend on the agents' concurrent actions, but instead depends on the two-step sequence of agent 1's action followed by agent 2's action. Adding a delay does change the problem, but it need not change the problem significantly if we consider time step $t+1$ as an intermediate decision step whose real-world time value is arbitrarily close to that of time step t.

Further investigation is needed to determine whether or not such a transformation is applicable to any general class of concurrent interaction effects, and to ensure that the solution returned by the equivalent TD-POMDP is realistically implementable. As future work, formalizing the transformation from a Dec-POMDP with (structured) concurrent interaction effects would ensure that my theoretical results and influence-based policy formulation algorithms apply to a larger space of problem than I have carved out here, thereby expanding the scope of this dissertation's contributions.

3.4.3.2 Generalized Utility Structure

As indicated by Theorem 3.8, the TD-POMDP's joint value function is the sum of local values, each dependent on agent i's local rewards:

$$V(\pi) = \sum_{i=1}^{n} V_i(\pi)\,. \tag{3.12}$$

Another way of stating this property is that the agents' *quality accumulation function* is summation, which is a common assumption shared by much the cooperative planning literature (Becker et al., 2004a; Beynier & Mouaddib, 2005; Guestrin et al., 2001; Marecki & Tambe, 2009) and distributed constraint optimization literature (Atlas, 2009; Modi et al., 2005; Petcu & Faltings, 2005). Despite its frequent usage, this property makes the TD-POMDP more restrictive than the general Dec-POMDP model, which allows for arbitrarily complex aggregation of utility values.

Generalizing to a broader class of quality aggregation functions is the subject of future work. Intuitively, the crucial requirement of the TD-POMDP is that the value decomposes, such that the planning problem can solved in a distributed, decentralized manner. The applicability and usefulness of the model should be less dependent on the actual aggregation function. For instance, I speculate that generalizing my solution framework to a broader space of *monotonic* quality aggregation functions is possible. However, this hypothesis has yet to be substantiated.

3.5 Weak Coupling

The TD-POMDP specification is quite general (as I argued in Section 3.4), but not all problems in its expressible repertoire are efficiently solvable (as I proved in Section 3.3). In this section, I endeavor to illuminate the tractable subspace consisting of problems that I call "weakly coupled", driven by the realization that the TD-POMDP actually emphasizes exploitable structure (summarized in Section 3.3.3) present in these tractable problems. The structure is significant in that it decouples the joint planning model into a set of local models that are bound together with the transition dependencies of agents' nonlocal features. By explicitly distinguishing those nonlocal features, and acknowledging the resulting factoring of the agents' transition and observation functions, a specification emerges that, given sufficient factored structure, is substantially more compact than the general Dec-POMDP specification. Likewise, the exploitation of TD-POMDP structure during planning is shown in later chapters to enable exponential gains in computational efficiency over prior methods.

Neither representational compactness nor computational efficiency are particularly surprising, as these advantages are well documented consequences of factored models in single-agent decision theoretic planning (Boutilier et al., 1999; Guestrin et al., 2003). (The interaction structure made explicit by the TD-POMDP is a special case of factorization.) Less obvious, however, are the structural conditions under which the TD-POMDP representation is advantageous. At one extreme, a multiagent problem might involve no agent interaction (and hence no nonlocal features), represented in the TD-POMDP framework as a collection of completely independent single agent POMDP models, translating to an exponential reduction in (worst-case) time and space complexity from the general Dec-POMDP representation. For problems with relatively sparse interactions, there may be something to gain by explicitly representing each nonlocal feature (as in the TD-POMDP) and exploiting the resultant conditional independencies. However, for problems with dense agent interactions, where agents

are heavily dependent on one another, and where the number of nonlocal features is of the same order of magnitude as the number of world state features, it is not clear that we could gain anything from representing the problem as a TD-POMDP.

Intuitively, the amount of advantage (over the naïve Dec-POMDP representation) afforded by the TD-POMDP representation on any given planning problem depends on the *weakness of agent coupling*. Outside of the TD-POMDP model, researchers have previously used the term "weakly-coupled" (and alternatively, "loosely-coupled") to refer to multiagent sequential decision making problems under various structural properties that impose conditional independencies among agents' subproblems (Bernstein et al., 2001; Cavallo et al., 2006; Dolgov & Durfee, 2004b; Guo & Lesser, 2005; Meuleau et al., 1998; Mostafa & Lesser, 2009). However, in all of these works, the "weakly-coupled" qualifier specified either a binary classification (weakly-coupled versus *not* weakly-coupled) or a qualitative assessment. Instead, I pose the question *exactly how weakly coupled is a given problem?* in order to develop a quantitative scale that can be used to determine the degree to which advantageous structure is present and, ultimately, to predict the amount of computation needed to solve the problem. If accurate, a quantitative measure of weak coupling could be extremely useful for the meta-level control of systems with a variable amount of computational resources, where predicting the relative complexity of a problem could determine an appropriate allocation of resources for solving it. The measure could also be useful, when facing a set of problems with diverse problem sizes, in deciding whether a problem will be tractable to solve optimally or the system would be better off employing an approximate solution method.

In the subsections that follow, I consider several aspects that contribute to my formulation of a measure of *weakness of coupling*. For each aspect (in Sections 3.5.1–3.5.2), I describe its respective structural assumptions (referencing prior work where appropriate), relate it to the TD-POMDP specification, and formalize how it affects computational complexity, successively refining a bound on the worst-case time required to compute optimal solutions to the TD-POMDP. In Section 3.5.3, I summarize the results and their ramifications, and in Section 3.5.4, I contrast my analysis with other work that relates problem structure to problem hardness.

3.5.1 Locality of Interaction

The first two aspects of weakness of coupling are both instances of a broader concept referred to in the literature as *locality of interaction* (Kim et al., 2006; Melo, 2008; Nair et al., 2005; Oliehoek et al., 2008b; Tasaki et al., 2010). Interaction (in its

most general form) in Dec-POMDP problems consists of arbitrary dependence of one agent's rewards, state-action outcomes, and observations, on another agents' actions. Interaction may be very broad in the sense that it involves all the agents' actions and all the features of the world state. Alternatively, it may be relatively local, involving only a small subset of agents that interact through a small subset of features. Because the TD-POMDP explicitly distinguishes the (mutually-modeled) features through which agents interact, it serves as a natural setting for formal analysis of locality of interaction.

In the subsections that follow, I decompose *locality of interaction* into *agent scope*, which I refer to as the portion of the agent population on which an interaction depends, and *state factor scope*, which I refer to as the portion of state features on which an interaction depends. These two aspects affect the complexity of joint planning along orthogonal dimensions. Before delving into the details of how *agent scope* (Sec. 3.5.1.2) and *state factor scope* (Sec. 3.5.1.3) affects complexity, I establish the preliminary background context in Section 3.5.1.1, where I relate Dec-POMDP planning to constraint optimization.

3.5.1.1 Relationship to Constraint Optimization

The TD-POMDP complexity results that I present throughout this section rely on a reduction of the planning problem to a *constraint optimization problem* (*COP*). Here I briefly review the classical COP model as it was defined by Dechter (2003) and show how any Dec-POMDP can be mapped into an equivalent COP.

Definition 3.22. A **constraint optimization problem (COP)** is specified as a tuple $\mathcal{C} = \langle X, D, C \rangle$, whose components and auxiliary notations I explicate:

- $X = \{x_1, ..., x_n\}$ is a set of n *variables*,
- $D = \{D_1, ..., D_n\}$ is comprised of the domain D_i of values assignable to each variable x_i,
- an *assignment* $\bar{a} = \langle a_1, ..., a_n \rangle$ specifies a value $a_i \in D_i$ for each variable x_i,
- $C = \left\{C_1, ..., C_{\|C\|}\right\}$ is a set of *constraints*, each taking the form of a *cost function*[19] C_k with scope $Q_k \subseteq \{1, ..., n\}$ such that $C_k : \prod_{k \in Q_k} D_k \mapsto \{\mathbb{R}, \infty\}$

[19] In my treatment of constraint optimization, I do not distinguish "hard" and "soft" constraints since both flavors can be represented without loss of generality using cost functions that map assignments to {real numbers, infinity}.

and the application of C_k to the restricted scope of an assignment $\bar{a}$ is denoted $C_k(\bar{a}) \equiv C_k(a_i, \forall i \in Q_k)$,

- the *global cost* of an assignment $\bar{a}$ is $C(\bar{a}) = \sum_{k=1}^{\|C\|} C_k(\bar{a})$, and
- a *solution* to $\mathcal{C}$ is an assignment $\bar{a}^*$ that minimizes the global cost.

Constraint optimization (Def. 3.22) is a useful formulation in that it emphasizes particular problem structure. Each cost function is defined over a subset of variables, and all costs are aggregated via summation to yield the global cost. This structure is often depicted graphically using a constraint graph (Definition 3.23). An example of a constraint graph is shown in Figure 3.6.

Definition 3.23. The **constraint graph** $\mathcal{G}_\mathcal{C}$ for a COP $\mathcal{C}$ is an undirected hypergraph whose vertices $V = \{v_1, ..., v_n\}$ contain a vertex v_i for each variable $x_i \in X$ and whose edges $E = \{e_1, ..., e_{\|C\|}\}$ contain a hyperedge $e_k \subseteq \{1, ..., n\}$ for each constraint $C_k \in C$ that connects the vertices indexed $\{v_i, \forall i \in Q_k\}$ by the corresponding scope Q_k.

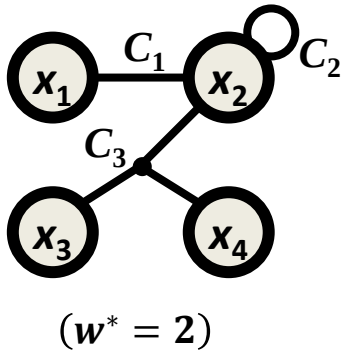


Figure 3.6: An example of a constraint graph.

In turn, solution methods for COPs exploit the graphical structure. One such method, *bucket elimination* (Dechter, 1999), performs dynamic programming on an ordered constraint graph (Def. 3.24), traversing edges from the bottom of the graph to the top of the graph in order to eliminate individual variable assignments in an efficient manner (thereby avoiding consideration of superfluous combinations of variables' values). Bucket elimination is most efficient when the connectivity of the graph is sparse. For the purpose of evaluating bucket elimination, Dechter (2003) quantifies the sparsity of the constraint graph using a measure of *induced width*, the definition of which I review below.

Definition 3.24. An **ordered graph** $\langle \mathcal{G}, d \rangle$ prescribes an *ordering* $d = \langle v_1, .., v_n \rangle$ of the vertices V in $\mathcal{G}$. In an ordered constraint graph, the *parents* of a vertex v_j denote those vertices preceding v_j in the ordering that are connected to v_j: $parents(v_j) = \{v_i \in V | ((i < j) \wedge (\exists e_k \in E) \wedge (\{i, j\} \subseteq e_k))\}$. The *width* $w(\mathcal{G}, d)$ of the ordered graph $\langle \mathcal{G}, d \rangle$ denotes the maximum number of parents of any vertex: $\omega(\mathcal{G}, d) = \max_{v_j \in V} \|parents(v_j)\|$.

Definition 3.25. The induced width $w^*(\mathcal{G}, d)$ of an ordered graph is the width obtained by processing nodes from last to first, such that a node is processed by connecting its parents. The **induced width** $w^*(\mathcal{G})$ is the minimum induced width of any ordering of $\mathcal{G}_\mathcal{C}$: $w^*(\mathcal{G}_\mathcal{C}) = \min_d [w(\mathcal{G}_\mathcal{C}, d)]$.

The induced width of the constraint graph in Figure 3.6 is 2, which is the width of the ordered graph that orders the vertices $\langle x_1, x_2, x_3, x_4 \rangle$.

The details of Dechter's analysis (Dechter, 1999) are beyond the scope of this dissertation, but her results (which we will extend) are as follows. The worst-case time and space complexity of bucket elimination for constraint optimization is $O\left(\|C\| \cdot \|D_i^{max}\|^{w^*+1}\right)$, when applied to a problem with a maximum variable domain size of $\|D_i^{max}\| = \max_{D_i \in D} \|D_i\|$, $\|C\|$ constraints, and induced width w^* (Dechter, 2003). Notice that the complexity is not a function of the number of variables. The asymptotic complexity is the same for problems with many variables as it is for problems with few variables as long as the induced width is equal, the maximum variable domain sizes are equal, and the number of constraints is bounded. Since complexity is exponential in the induced width, Dechter's analysis suggests that problems with a small induced width should be easier to solve. This general trend is intuitive: fewer agents connected by a single constraint means fewer combinations of behavior to consider in minimizing the constraint's cost. In my analysis of TD-POMDP complexity that follows, induced width will play a crucial role in characterizing a measure of weak coupling.

In a constraint optimization problem, the objective is to minimize the summation of local cost values of the variable assignments. For Dec-POMDP problems, the objective is to maximize the expected utility value of the joint policy. Additionally, for certain Dec-POMDP problems, and notably any TD-POMDP problem, the value function can be decomposed into component value functions. The inherent similarity between the two problems leads me to the following mapping of a Dec-POMDP to an equivalent COP:

Observation 3.26. *A Dec-POMDP $\mathcal{M}$, whose value function $V(\pi = \langle \pi_1, ..., \pi_n \rangle)$ is decomposable into component value functions $\{V_1(\pi), ..., V_k(\pi)\}$ such that $V(\pi) = \sum_{i=1}^{k} V_k(\pi)$ may be reduced to a COP $\mathcal{C}_{\mathcal{M}} = \langle X, D, C \rangle$ with the following specification:*

- *X $\{x_1, ..., x_n\}$ contains exactly one variable x_i for each agent,*
- *The domain D_i of each variable x_i is the set of agent i's possible (deterministic) local policies Π_i,*
- *an assignment $\bar{a} = \langle a_1 \equiv \pi_i, ..., a_n \equiv \pi_n \rangle$ specifies a joint policy (i.e. a policy $\pi_i \in \Pi_i$ for each agent),*
- *$C = \{C_1, ..., C_k\}$ consists of a single cost function C_i for each component V_i of $\mathcal{M}$'s value function, each taking the form $C_i(\bar{a} = \pi) = -V_i(\pi)$.*

Using the mapping in Observation 3.26, a solution assignment for $\mathcal{C}_{\mathcal{M}}$ equates to a joint policy that maximizes the Dec-POMDPs value function. In the case that the Dec-POMDP's value function does not decompose into component value functions, the COP will contain a single constraint (and the constraint graph a single edge) that connects all vertices. However, the benefit of the COP representation, its potentially-sparse graphical structure, is only present when the value function is decomposable into local component value functions with limited scopes.

For the problems that I consider in this thesis, the TD-POMDP specification explicitly factors the joint value function into agents' local value functions (Def. 3.7). Each $V_i()$ corresponds to a single agent's expectation of the summation of its local rewards. Thus, for a TD-POMDP, the equivalent COP will contain a single constraint for each agent. The scope of each constraint, equivalently the scope of each local value function, is not immediately obvious. In section 3.5.1.2, I describe how the local value scopes can be extracted from the TD-POMDP specification.

I am not the first to observe an equivalence between multiagent sequential decision making and constraint optimization. In the context of the TI-Dec-MDP (reviewed in Section 2.3.2), Becker et al. (2004b) propose a reformulation of their coverage set algorithm as one of distributed[20] constraint optimization (DCOP) (Yokoo et al., 1998) with a mapping of variable domains to local policy sets identical to that given in

[20]In *distributed constraint optimization*, the prefix "distributed" conveys a distribution of the COP specification among multiple agents as well as distribution of computational resources for solving the problem. In the case of the DCOP proposed by Becker et al. (2004b), each agent is charged with computing its own policy and communicating that policy to a subset of other agents that are linked to it via constraints.

Observation 3.26. The authors describe special constraints that correspond to the TI-Dec-MDP's structured reward dependencies between pairs of agents. Nair et al. (2005) map a related class of transition-independent problems, ND-POMDPs, into DCOPs containing *local neighborhood utility* constraints. Like the mapping I suggest, their local neighborhood constraints each involve a restricted scope of agents that contribute to a given component of the joint value function. Nair and colleagues exploit the resulting *locality of interaction* in their development of locally-optimal and globally-optimal solution algorithms (Kim et al., 2006; Nair et al., 2005). Whereas these previous works both focus on classes involving agents that interact through the reward function but that are transition-independent, TD-POMDP agents interact through their transitions, complicating the analysis of how individual agents affect the joint value. Thus, the study of value-based COP constraints that I present here is intended to supplement these past works with more general analysis.

More recently, Oliehoek et al. (2008b) describe an extension of *local neighborhood constraints* to the more general class of factored Dec-POMDPs (reviewed in Section 2.3.2). Instead of mapping the overall planning problem to a static COP, they analyze how the scope of local value functions change over the course of policy execution, thereby uncovering a dynamic constraint structure (that is dependent not only on the factored structure of the value function, but also on the particular decision stage). The complexity analysis that I present in the following subsection is complementary in that it too allows for general constraints, but assumes a static COP (wherein variables represent complete policies) that could be extended to account for Oliehoek et al.'s dynamics.

3.5.1.2 Agent Scope

The COP reformulation from Observation 3.26 touched upon the significance of structure in the value function of multiagent planning problems. In particular, worst-case complexity is heavily dependent on the *agent scope*, or the number of agents whose decisions can affect a given component value function. To make this relationship more concrete, we now turn to the TD-POMDP, where a careful examination of problem specification allows for a formalization of *agent scope* and further analysis of complexity.

As detailed in Section 3.2.2, the effects that TD-POMDP agents have on one another are represented as *nonlocal features* shared by the agents' local states. It is through changes to these features that agents interact. We can depict their potential interactions graphically using an agent interaction digraph (as shown in Figure 3.7).

Definition 3.27. A **agent interaction digraph**[21] $\mathcal{D}_{\mathcal{M}}$ **for TD-POMDP** $\mathcal{M}$ is a directed graph with a vertex v_i representing each agent i. For any two agents i and j, there is an edge from v_i to v_j for each of agent i's *locally-controlled features* that is modeled as a *nonlocal* feature in agent j's local state representation.

Definition 3.28. An agent i's **ancestors**, denoted $\Lambda(i)$, is the set of agents (apart from i) that correspond to ancestor vertices of vertex i in the interaction digraph. Formally, for any agent $j \neq i$, if there is a directed path from v_j to v_i in the interaction digraph, $j \in \Lambda(i)$.

Definition 3.29. An agent i's **descendants**, denoted $\Psi(i)$, is the set of agents that correspond to descendant vertices of vertex i in the interaction digraph. Formally, for any agent $j \neq i$, if there is a directed path from v_i to v_j in the interaction digraph, $j \in \Psi(i)$.

Note that here and throughout, I use the word **peer** to mean "some other agent" without assuming or implying any graphical relationship between the agents. Alternatively, to indicate a graphical relationship, I will instead use the term *ancestor* or *descendant*.

For a team of agents, the interaction digraph summarizes the interdependencies that exist between agents' activities. Each outgoing edge represents one attribute through which an agent can affect another agent; and each incoming edge represents one attribute through which an agent can be affected by its ancestors. For any two agents i and j, there may be more than one edge leading from i to j, one for each nonlocal feature controlled by i and affecting j. Note also that there can be no self loops (leading both out of and into any given vertex) in the interaction digraph because, by definition, a nonlocal feature n_{ix} modeled by agent i is controlled only by another agent j (and not by i itself). The interaction digraph can, however, contain directed cycles containing two or more nodes.

As is the case with other graphical models (Dechter, 2003; Jordan, 1999; Koller & Friedman, 2009), the motivation for representing the TD-POMDP problem using

[21] This definition is adapted from that of Brafman and Domshlak's *agent interaction digraph* (Brafman & Domshlak, 2008) used for multi-agent classical planning and that of Guestrin's *coordination graph* (Guestrin et al., 2002) for multiagent MDP coordination. It can be viewed as a generalization of the former (Brafman's digraph), where each edge represents a special kind of activity dependence: the precondition of one agent's planning operator is fulfilled by another agent's operator. The edges in the digraph presented here encode more general state feature dependencies. It extends the latter (Guestrin's coordination graph) in that it explicitly labels edges with the dependent features, and contains one such edge for each dependent feature (whereas the coordination graph represents at most one edge from i to j without specifying the particulars of the dependence).

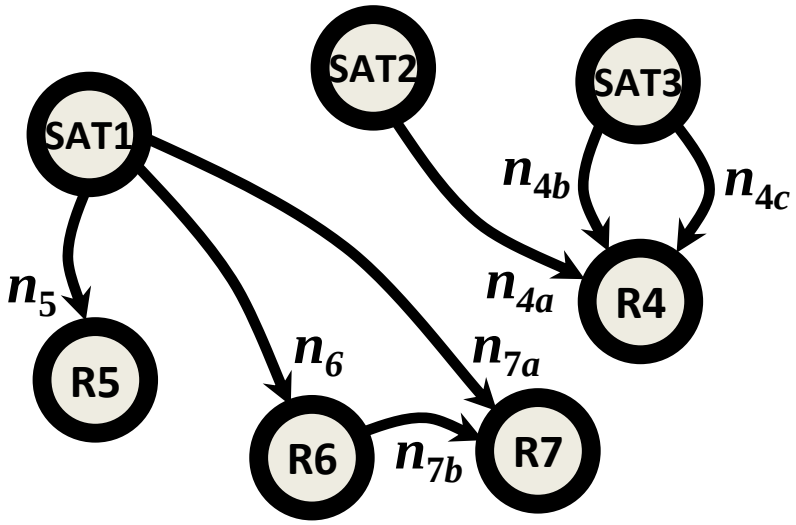


Figure 3.7: The interaction graph for Example 3.1.

an interaction digraph is to exploit structure in the connectivity of the graph. In the literature on multiagent planning under uncertainty, there has been a variety of work that exploits graphical interaction structure in specialized contexts (Beynier & Mouaddib, 2005; Brafman & Domshlak, 2008; Dolgov & Durfee, 2004a; Guestrin et al., 2002; Marecki & Tambe, 2009; Nair et al., 2005; Oliehoek et al., 2008b). Although these prior methods are not directly applicable here, we can make use of common concepts that have emerged.

The degree of agent coupling is directly related to the density of edges in the interaction digraph. In the extreme case of weak coupling, the agents are uncoupled, such that there are no nonlocal features and we are left with a graph of n unconnected vertices. An isolated vertex refers to an agent whose policy formulation problem is independent of all other agents' problems, since the agent cannot be influenced by others choices, nor can its own choices influence its peers. Thus, in the uncoupled case, the optimal joint policy can be computed by simply combining the n optimal local policies, each computed independently of the others.

In the extreme case of strong coupling, every locally-controlled feature in each agent's local state corresponds to a nonlocal feature in every other agent's state. Let there be n agents and k such local features per agent. In this case, all $k \cdot n$ features of the world state are contained in every agent's local state. The interaction digraph is not only fully-connected (with edges running in both directions between every pair of vertices), but each directed edge is duplicated k times, yielding a total of $n \cdot (n-1) \cdot k$

edges. Because each agent in the system is interacting with every other agent, no single agent's decision problem can be solved independently of any of its peers' decision problems (as in the uncoupled case). Moreover, no single local feature can be reasoned about independently because its value affects peer agents' decisions.

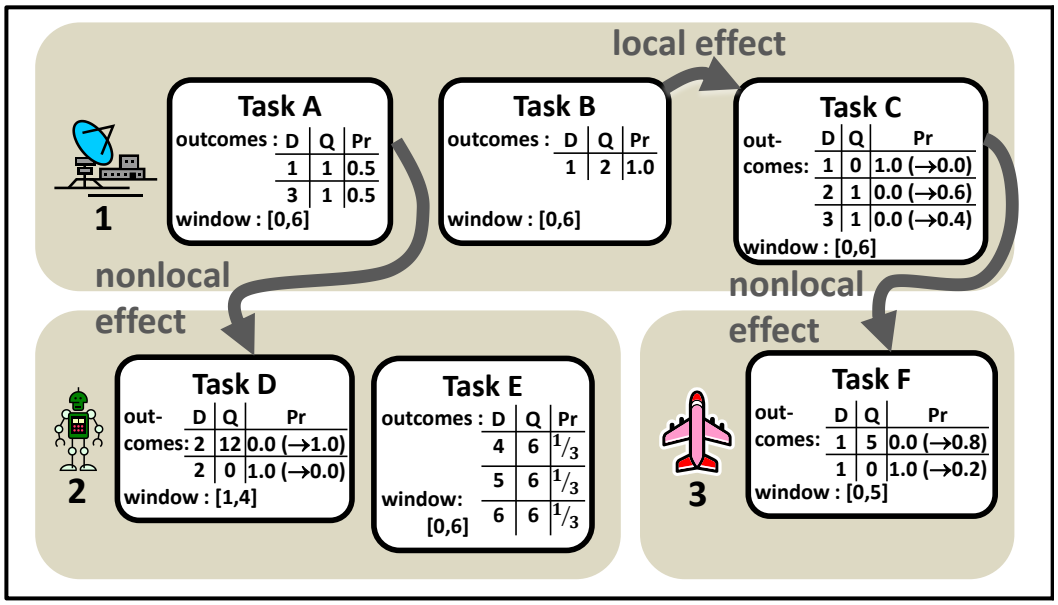
Both the unconnected and fully-connected cases are degenerate in the sense that the former conveys complete independence and the latter does not convey any conditional independence relationships whatsoever. The more interesting cases are those falling in between the two extremes, where there are some conditional independence relationships that can be taken advantage of, but the agents' activities are not completely independent. A simple example of one such case appears in Example 3.31 and Figure 3.8.

Definition 3.30. The **agent scope**, denoted Q_i, of a value function $V_i()$ is the subset of agents on whose policies its value depends. Equivalently, $V_i : \left[\prod_{j \in Q_i} \Pi_j\right] \mapsto \mathbb{R}$.

Example 3.31. Consider the interaction digraph in Figure 3.8, where two edges connect three agents. The first edge indicates that agent 1's activities can affect the activities of agent 2 and the second edge indicates that agent 1's activities can affect the activities of agent 3. Notice that there is no directed path leading from node 2 to node 3, implying that the outcomes of agent 2's activities cannot affect the outcomes of agent 3's activities. As such, agent 3's local value function $V_3(\pi)$ is independent of agent 2's policy decisions and can be rewritten $V_3(\pi_1, \pi_3)$. Likewise, $V_2(\pi) \equiv V_2(\pi_1, \pi_2)$.

Theorem 3.32. *The only agents that can affect the values of the features in agent i's local state are i's ancestors $\Lambda(i)$ and i itself.*

Proof. Mathematically, Theorem 3.32 states: $\forall t Pr(s_i^{t+1}|\vec{a}^t) = Pr(s_i^{t+1}|\vec{a}_i^t, \vec{a}_{\Lambda(i)}^t)$, where $\vec{a}^t$ is the joint action history, $\vec{a}_i^t$ is agent i's local action history, and $\vec{a}_{\Lambda(i)}^t$ is the history of actions performed by i's ancestors. This is a statement about conditional independence and, in particular, one that can be inferred from the DBN shown in Figure 3.5, which captures the conditional independencies among the TD-POMDPs factored state and action variables. It suffices to prove that, for any agent $j \notin (\{i\} \cup \Lambda(i))$, there cannot be a path of any length $n \geq 1$ in the DBN leading from variable a_j^{t-n+1} to s_i^{t+1}. I prove this by induction on n.



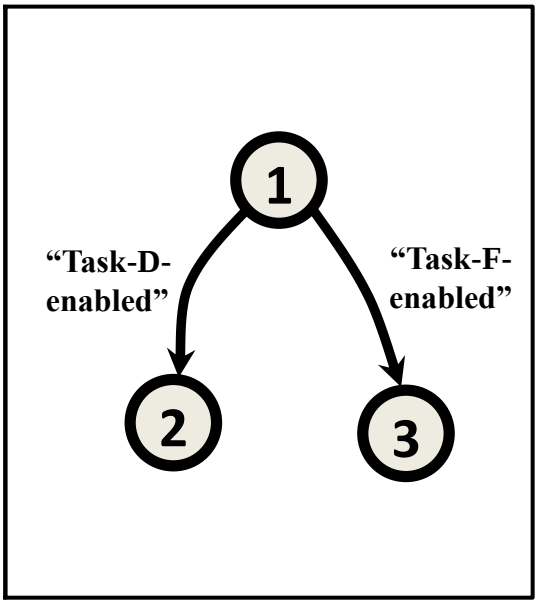


Figure 3.8: An example of exploitable interaction digraph structure.

Base Case ($n = 1$): By Definition 3.14, the only action variables whose edges lead into s_i^{t+1} are actions a_i and those actions of other agents who control i's nonlocal features (who are thus necessarily digraph ancestors of i by Definition 3.27). Since $j \notin (\{i\} \cup \Lambda(i))$, there is no path leading from j to i.
Inductive Hypothesis (IH): For any agent $j \notin (\{i\} \cup \Lambda(i))$, there cannot be a path of length n in the DBN leading from variable a_j^{t-n+1} to s_i^{t+1}.
Inductive Step: Here, we assume IH and deduce that, for any agent $k \notin (\{i\} \cup \Lambda(i))$, there cannot be a path of length $n+1$ in the DBN leading from variable a_k^{t-n} to s_i^{t+1}. Let us assume that this deduction is false, or in other words, that there exists an agent $k \notin (\{i\} \cup \Lambda(i))$ for which a path of length $n+1$ in the DBN leads from variable a_k^{t-n} to s_i^{t+1}. If this is the case, a_k^{t-n} connects to another variable from which there is a path of length n to s_i^{t+1}, which, according to the DBN from Figure 3.5, must be a state feature variable f^{t-n+1}. There are two possibilities for f:

1. $f \in s_i$: In this case, there is path of length n from s_i^{t-n+1} to s_i^{t+1}, and a path of length 1 leading from a_k^{t-n} to s_i^{t-n+1}. Since we were assuming that $k \notin (\{i\} \cup \Lambda(i))$, this directly contradicts the result derived in our base case.

2. $f \notin s_i$. If there is such a path, we can deduce that f must be a nonlocal feature, because the only DBN connections between two separate agents' state features are by way of nonlocal features. Further, we can deduce that f must be controlled by k and is modeled by some other agent j. Using the same logic, we can also deduce that j is in turn controlling a nonlocal feature, and that there must therefore also be a path of length n from a_j^{t-n+1} to s_i^{t+1}. This contradicts our inductive hypothesis.

Having derived a contradiction in both cases, the inductive step must be correct. Therefore, agents that can affect the values of the features in agent i's local state are i's ancestors $\Lambda(i)$ and i itself.

□

Theorem 3.33. *For TD-POMDP agent i, the* agent scope Q_i *of i's local value function $V_i()$ does not include agents outside of i and i's ancestors: $Q_i \subseteq (\{i\} \cup \Lambda(i))$.*

Proof. By Definition 3.7, $V_i(\pi)$ is an expectation of the summation of agent i's local rewards, each of the form $R_i\left(s_i^t, a_i^t, s_i^{t+1}\right)$, and is thus a function of i's local state s_i and local action a_i. It suffices to prove that the only agents that can affect i's local state and local action are $(\{i\} \cup \Lambda(i))$:

1. By Theorem 3.32, the only agents that affect i's local state are $(\{i\} \cup \Lambda(i))$.

2. Agent i's local actions are dictated by agent i's policy π_i, which is a mapping of agent i's local observation history $\vec{o}_i^t$ to local action a_i^t. By Definition 3.5, $\vec{o}_i^t$ can only depend on past values of agent i's local state. By Theorem 3.32, the only agents that affect i's local state are $(\{i\} \cup \Lambda(i))$. Thus, the only agents that can affect agent i's actions are $(\{i\} \cup \Lambda(i))$.

Therefore, $V_i(\pi) = V_i(\pi_i, \bar{\pi}_{\Lambda(i)})$, and equivalently $Q_i \subseteq (\{i\} \cup \Lambda(i))$. □

When the agent scope is reduced (from the set of all agents), it implies a conditional independence that can be exploited during individual agent reasoning.

Example 3.31 (continued). A practical consequence of the reduced agent scope in the example problem (Figure 3.8) is that, given agent 1's planned decisions, agent 3's decisions may be planned independently of agent 2's decisions (without sacrificing optimality of the agents' planned policies). In other words, if agent 3 knows the policy decisions of agent 1, agent 3 does not need to reason about agent 2's decisions in order to plan its local policy component of the optimal joint policy. Nor does agent 2 need to reason about agent 3's decisions. In other words, agent 2's policy decisions are conditionally independent of agent 3's policy decisions given agent 1's policy.

The definitions and theorems that follow formalize the conditional independence relationships contained in the TD-POMDP agent interaction digraph and their implications on the multi-agent planning problem.

Definition 3.34. An agent i is **conditionally decision-independent** of peer agent j conditioned on the decisions of $c \in \{0, 1, ..., n-2\}$ other agents $K = \{k_1, ..., k_c\}$ if, in maximizing[22] the joint value of the team, i's optimal decisions do not differ depending on j's decisions (given any fixed settings of K's decisions): $\forall \left\{\pi_j^x, \pi_j^y\right\} \subset \Pi_j, \forall \pi_{k_1}, \ldots, \pi_{k_c}$,

$$\underset{\pi_i \in \Pi_i}{\mathbf{arg}\max}\, V\left(\pi_i, \pi_j^x, \pi_{k_1}, \ldots, \pi_{k_c}\right) = \underset{\pi_i \in \Pi_i}{\mathbf{arg}\max}\, V\left(\pi_i, \pi_j^y, \pi_{k_1}, \ldots, \pi_{k_c}\right)$$

The equation in Definition 3.34 contains terms of the following form, with which agent i can compute local policies that maximize the team's joint value given candidate

[22] Here, I notate maximization with $\underset{x}{\mathbf{arg}\max}\, f(x)$, which returns the set of all values of argument x that achieve the maximal value of the expression $f(x)$ (to which **arg** max is applied). Elsewhere, I use the notation $\underset{x}{\arg\max}$ (in plain, not bold, text) to refer to the maximization that returns a *single* maximizing argument instead of a set.

policies of i's peers:

$$\pi_i^*(\pi_j, \ldots) = \arg\max_{\pi_i \in \Pi_i} V(\pi_i, \pi_j, \ldots) \tag{3.13}$$

One such maximizing argument setting, $\pi_i^*(\pi_j, \ldots)$, is commonly referred to as agent i's **best response** to peer policies $\pi_{\neq i} = \{\pi_j, ...\}$. This is the same paradigm that was reviewed in the discussion of decoupled solution methodologies (Sec. 2.3.3). As we will see, conditional decision-independence may be exploited within a decoupled solution method to substantially reduce the computational complexity of optimal planning.

Theorem 3.35. *If (1) there is no directed path connecting (distinct) nodes i and j in the interaction digraph and (2) nodes i and j share no common descendants, then i is* decision-independent *of any agent j* conditioned on *i's ancestors' policies $\bar{\pi}_{\Lambda(i)}$*

Proof. By definition, the theorem states that $\forall \pi_j^x, \pi_j^y, \forall \bar{\pi}_{\Lambda(i)}, \arg\max_{\pi_i} V\big(\pi_i, \pi_j^x, \bar{\pi}_{\Lambda(i)}\big) = \arg\max_{\pi_i} V\big(\pi_i, \pi_j^y, \bar{\pi}_{\Lambda(i)}\big)$. Recall, from Theorem 3.8, that the value function is composed of local value functions: $V(\pi_1, ..., \pi_n) = \sum_i V_i(\pi_1, ..., \pi_n)$. By the monotonicity of summation, it suffices to prove that for each local value function $V_k()$, $\forall \pi_j^x, \pi_j^y, \forall \bar{\pi}_{\Lambda(i)}, \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^x, \bar{\pi}_{\Lambda(i)}\big) = \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^y, \bar{\pi}_{\Lambda(i)}\big)$.

- Case A ($k = i$) : By Theorem 3.33, $V_i : \Pi_i \times \Pi_{\Lambda(i)} \mapsto \mathbb{R}$. Clause 1 of Theorem 3.35 states that there is no path connecting i and j. Thus, $j \notin \Lambda(i)$, and consequently, $V_{k=i}()$ is independent of π_j. Trivially, $\forall \pi_j^x, \pi_j^y, \forall \bar{\pi}_{\Lambda(k)}, \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^x, \bar{\pi}_{\Lambda(i)}\big) = \arg\max_{\pi_i} V_k\big(\pi_i, \bar{\pi}_{\Lambda(i)}\big) = \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^y, \bar{\pi}_{\Lambda(i)}\big)$.
- Case B ($k \in \Psi(i)$) : Here, conversely agent i is an ancestor of k ($i \in \Lambda(k)$), and hence $V_k()$ may depend upon π_i. By clause 2 of Theorem 3.35, i and j cannot share any descendants, so $j \notin \Lambda(k)$. Thus, $V_k()$ is independent of π_j, and $\forall \pi_j^x, \pi_j^y, \forall \bar{\pi}_{\Lambda(i)}, \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^x, \bar{\pi}_{\Lambda(i)}\big) = \arg\max_{\pi_i} V_k\big(\pi_i, \bar{\pi}_{(\Lambda(k)-i)}\big) = \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^y, \bar{\pi}_{\Lambda(i)}\big)$.
- Case C (otherwise) : Given that neither Case A nor Case B applied, it must be that $i \neq k$ and $i \notin \Lambda(k)$. By Theorem 3.33, $V_k()$ must be independent of π_i. Thus, $\forall \pi_j^x, \pi_j^y, \forall \bar{\pi}_{\Lambda(i)}, \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^x, \bar{\pi}_{\Lambda(i)}\big) = \arg\max_{\pi_i} V_k\big(\pi_i, \pi_j^y, \bar{\pi}_{\Lambda(i)}\big)$.

Having derived the necessary arg max equality for all of the local value components $\{V_k\}$ of the joint value function $V()$, it must in turn hold for $V()$ because the joint value composition function, *summation*, preserves the order (including the arg max)

of input values for each parameter. Therefore, given clauses 1 and 2 of the theorem, i is *decision-independent* of agent j *conditioned on* the decisions of the agents indexed by $\Lambda(i)$. □

Given the preceding characterization of TD-POMDP *agent scope* and the accompanying conditional independencies, we can now relate it back to the COP mapping developed in Section 3.5.1.1. Recall that maximizing the joint value of the TD-POMDP is equivalent to minimizing the sum of costs of particular constraints pertaining to component value functions. Converting the TD-POMDP into a COP thus involves creating a constraint C_i for each local value function $V_i()$. Each constraint C_i constrains those variables that correspond to the policies of the agents in the respective agent scope Q_i. Similarly, the constraint graph for the mapped COP includes, for each local value function, a hyperedge linking together those vertices in the respective agent scope.

There are similarities between the constraint graph and the interaction digraph, but also notable differences. Like the TD-POMDP interaction digraph, the constraint graph contains a single vertex x_i for each agent i. However, whereas the interaction digraph contains a directed edge for each nonlocal feature connecting a pair of vertices, the constraint graph contains an undirected hyperedge C_i (connecting 1, 2, or perhaps more vertices) for each local value function V_i. As dissimilar as the connections in the two representations may appear, the translation from interaction digraph to equivalent constraint graph is straightforward. By Theorem 3.33, in general, the agent scope of a local value function $V_i()$ includes i and $\Lambda(i)$. Thus, for each agent i, there is a hyperedge C_i in the constraint graph connecting i and i's (interaction digraph) ancestors. To illustrate this translation, the interaction digraphs for Examples 3.1 and 3.31 are shown alongside their corresponding constraint graphs in Figure 3.9.

Definition 3.36. The **induced width** ω *of a TD-POMDP* $\mathcal{M}$ denotes the induced width (Def. 3.25) of its (unordered) equivalent constraint graph $\mathcal{G}_{\mathcal{M}}$ (converted from the interaction digraph $\mathcal{D}_{\mathcal{M}}$).

There are also useful relationships to be drawn between the induced width and agents' scope sizes. In general, $\omega \geq (\max_k \|Q_k\| - 1)$ (which follows from the definitions of ω and Q_k). I have observed $\omega \approx (\max_k \|Q_k\| - 1)$ to be a robust estimate of induced width for a wide variety of interaction digraph topologies (some of which are shown in Figure 3.9). Although there do exist instances for which $\omega > (\max_k \|Q_k\| - 1)$, as we will see later in Observation 3.37, approximating induced width from agent scope enables the subsequent approximation of TD-POMDP complexity without the

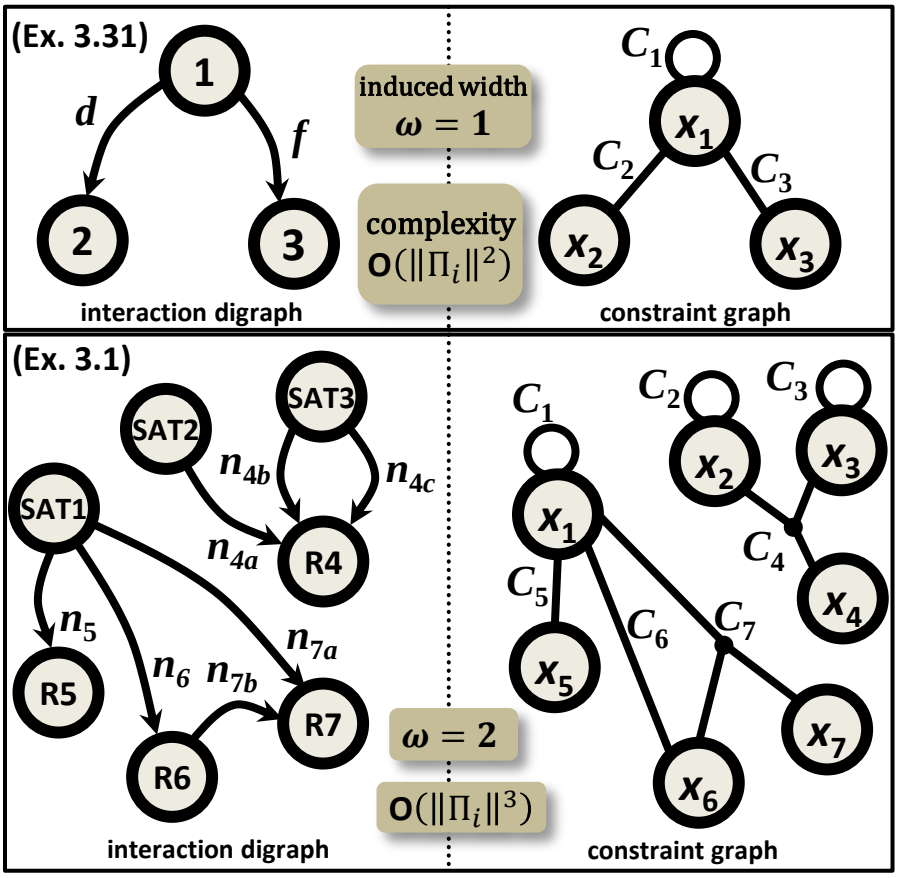


Figure 3.9: Examples of COP constraint graphs derived from interaction digraphs.

need to first convert the TD-POMDP interaction digraph to a constraint graph and evaluate its induced width.

Ultimately, gaining the advantages of reduced agent scope size (and reduced width in the COP reformulation) requires the use of a decoupled solution methodology (such as is reviewed in Section 2.3.3) wherein each agent computes its local component of the joint policy through a series of *best response* calculations in response to candidate peer policies. While searching through the space of policies, decision independence allows an agent to avoid reasoning about a peer's policy decisions, effectively pruning an entire cross-section of the joint policy space at no cost, as is the case for the problem described in Example 3.31 and Figure 3.8.

Example 3.31 (continued). Returning to the problem shown in Figure 3.8, we can deduce from the interaction digraph (by Theorem 3.35) that agents 2 and 3 are decision-independent of one another conditioned on agent 1's policy. By definition, this means that given a policy π_1 selected by agent 1, the optimal policy of agent 2 does not depend upon the policy chosen by agent 3 (and *vice versa*):

$$\begin{aligned}
\forall \pi_3^x, \pi_3^y, \forall \pi_1, \arg\max_{\pi_2} V(\pi_1, \pi_2, \pi_3^x,) &= \arg\max_{\pi_2} V(\pi_1, \pi_2, \pi_3^y,) = \arg\max_{\pi_2} V(\pi_1, \pi_2) \\
&= \arg\max_{\pi_2} [V_1(\pi_1) + V_2(\pi_1, \pi_2)] \\
&= \arg\max_{\pi_2} V_2(\pi_1, \pi_2) \\
\forall \pi_2^x, \pi_2^y, \forall \pi_1, \arg\max_{\pi_3} V(\pi_1, \pi_3, \pi_2^x,) &= \arg\max_{\pi_3} V(\pi_1, \pi_3, \pi_2^y,) = \arg\max_{\pi_3} V(\pi_1, \pi_3) \\
&= \arg\max_{\pi_3} [V_1(\pi_1) + V_3(\pi_1, \pi_3)] \\
&= \arg\max_{\pi_3} V_3(\pi_1, \pi_3)
\end{aligned}$$

Notice that each equation has been expanded to show the local value functions with their reduced agent scopes. The two equalities describe a simple brute-force method for computing the optimal joint policy. For each policy π_i of agent 1, agent 2 can compute its part of the optimal joint policy by evaluating each of its policies in conjunction with π_1 and creating a vector of best responses $\langle \pi_2^*(\pi_1) = \arg\max(...) \rangle$ of length $\|\Pi_1\|$. Here, agent 2 need only maintain a single best response per policy π_1 of agent 1.[23] This computation involves $\|\Pi_1\| \cdot \|\Pi_2\|$ policy evaluations. Similarly, agent 3 can compute its part of the optimal joint policy with another $\|\Pi_1\| \cdot \|\Pi_3\|$ policy evaluations, producing a vector of best responses $\langle \pi_3^*(\pi_1) \rangle$ of length $\|\Pi_1\|$.

Finally, the optimal joint policy can be determined by iterating through the two vectors and selecting the π_1^* (and associated $\pi_2^*(\pi_1^*)$ and $\pi_3^*(\pi_1^*)$) that maximize the joint utility value $[V_1(\pi_1) + V_2(\pi_1, \pi_2) + V_3(\pi_1, \pi_3)]$. When all is said and done, the agents will have computed the optimal joint policy using just $O\big(\|\Pi_i\|^2\big)$ policy evaluations. In contrast, the worst-case complexity of a 3 agent problem (in the absence of exploitable structure) is $O\big(\|\Pi_i\|^3\big)$ (as per Observation 3.20).

[23] Recall that arg max (without the bold font) returns a single maximizing argument. Here, the overarching problem is the computation of a single optimal joint policy, not all optimal joint policies. Moreover, the reason that agent 2 can get away with maintaining a single best response per π_1 is that π_2 does not appear at all outside of the term $V_2(\pi_1, \pi_2)$, indicating that maximizing $V_2(\pi_1, \pi_2)$ once and for all, with any arbitrary best response, will maximize the joint value function $V()$. Given

The solution process described in Example 3.31 is a simplified execution trace of the *bucket elimination* algorithm (introduced in Section 3.5.1.1) for solving constraint optimization problems (Dechter, 2003). A more detailed algorithmic description of Bucket Elimination appears in Chapter 6 (where I develop an extension for solving TD-POMDP problems).

The complexity reduction in Example 3.31 afforded by Bucket Elimination relies on a very simple 3-agent graph structure and will certainly not hold for all 3-agent TD-POMDPs. However, we can generalize our analysis by making use of the complexity result cited in Section 3.5.1.1. Recall that the worst-case time and space complexity of Bucket Elimination for constraint optimization is $O\left(c \cdot \|X_i^{max}\|^{\omega^*+1}\right)$, when applied to a problem with a maximum variable domain size of $\|X_i^{max}\|$, c (hard or soft) constraints, and induced width ω^* (Dechter, 2003). Instantiating each COP metric with the corresponding details from the TD-POMDP specification results in the following observation.

Observation 3.37. *The worst-case time and space complexity of solving a TD-POMDP problem is bounded by* $O\left(n \cdot \|\Pi_i^{max}\|^{\omega+1}\right)$*, where* n *is the number of agents,* Π_i^{max} *is the largest policy space of any agent, and* ω *is the induced width of the interaction digraph (Def. 3.36).*

As indicated in Figure 3.9, the computed complexity $\|\Pi_i\|^2$ of solving Example 3.31 is just as predicted by Observation 3.37. Even more substantial is the reduction in complexity of Example 3.1 (whose interaction digraph is shown in Figure 3.9), which has been tightened to $O(\|\Pi_i\|^3)$, down from $O(\|\Pi_i\|^7)$ when ignoring the problem's locality of interaction.

3.5.1.3 State Factor Scope

Locality of interaction manifests itself in a reduction of the scope of dependence of individual agent subproblems. As my theoretical results suggest thus far, the fewer the number of dependent agents (i.e. the smaller the agent scope), the simpler planning joint behavior becomes. An analogous statement can be made about the number of dependent world state features. The *state factor scope* (Guestrin et al., 2001; Oliehoek et al., 2008b) refers to the subset of world state features on which an agent's local

two choices of best response $\pi_2^{*_x}(\pi_1)$ and $\pi_2^{*_y}(\pi_1)$ for which $V_2(\pi_1, \pi_2^{*_x}) = V_2\left(\pi_1, \pi_2^{*_y}\right)$, there is no valuation that would differentiate between $\pi_2^{*_x}(\pi_1)$ and $\pi_2^{*_y}(\pi_1)$.

value depends.[24] Regardless of the number of agents involved, the state factor scope controls the complexity of each individual agent's reasoning.

To better understand how state factor scope affects the complexity of planning, consider the derivation of the previous complexity result in Section 3.5.1.2, which was based upon a reduction to a constraint optimization problem. Solving the equivalent COP involved evaluations of the form $\arg\max_{\pi_j} V(\pi_i, \pi_j, \ldots)$ (as in Equation 3.13) that constitute agent j's best-response calculation to potential policy π_i of peer agent i. The complexity result in Observation 3.37 assumes a naïve algorithm for performing this calculation: enumeration of all of i's policies and, for each, an explicit evaluation of $V_i(\pi_j)$. In a classical COP, enumeration of variable domains would be the only way to compute a best response. However, the COP that we are solving involves structured variable domains containing TD-POMDP policies. Instead of using simple enumeration, an agent can calculate its best response by solving a special POMDP model seeded with peers' policy information (like the one that I develop in Section 4.2). For a weakly-coupled TD-POMDP agent, its local best-response model does not necessarily need to represent all world state features. Intuitively, there may be world features that have no bearing on the value ascribed to the agent's own behavior.

Example 3.38. For instance, in the Example 3.31, whose TD-POMDP transition structure is shown in Figure 3.10, whether or not Task F is enabled (encoded by feature F*en*, appearing in agent 1's local state) has no bearing on agent 2's computation of best response $\pi_2^*(\pi_1)$. Using Definition 3.39 below, we say that feature F*en* is not in agent 2's state factor scope.

Definition 3.39. An agent i's **state factor scope** $\mathcal{X}_i$ is a minimal set of features that are sufficient for i to represent and reason about when computing a best response.

Definition 3.39 refers to a minimal set of features because, as I describe in Section 4.2, there are various flavors of the best-response model that could be used to compute the same best response but that represent different feature sets. Here, I am most interested in those that exploit weakly-coupled problem structure by reducing their modeled set of features as much as possible.

In addition to the set of features in the state factor scope, we can also discuss their

[24] Here I refer to *state factor scope* in the particular context of agents' local value functions. Note, however, that it is a more general mathematical concept that can also be used to characterize other functions (Guestrin et al., 2001; Oliehoek et al., 2008b).

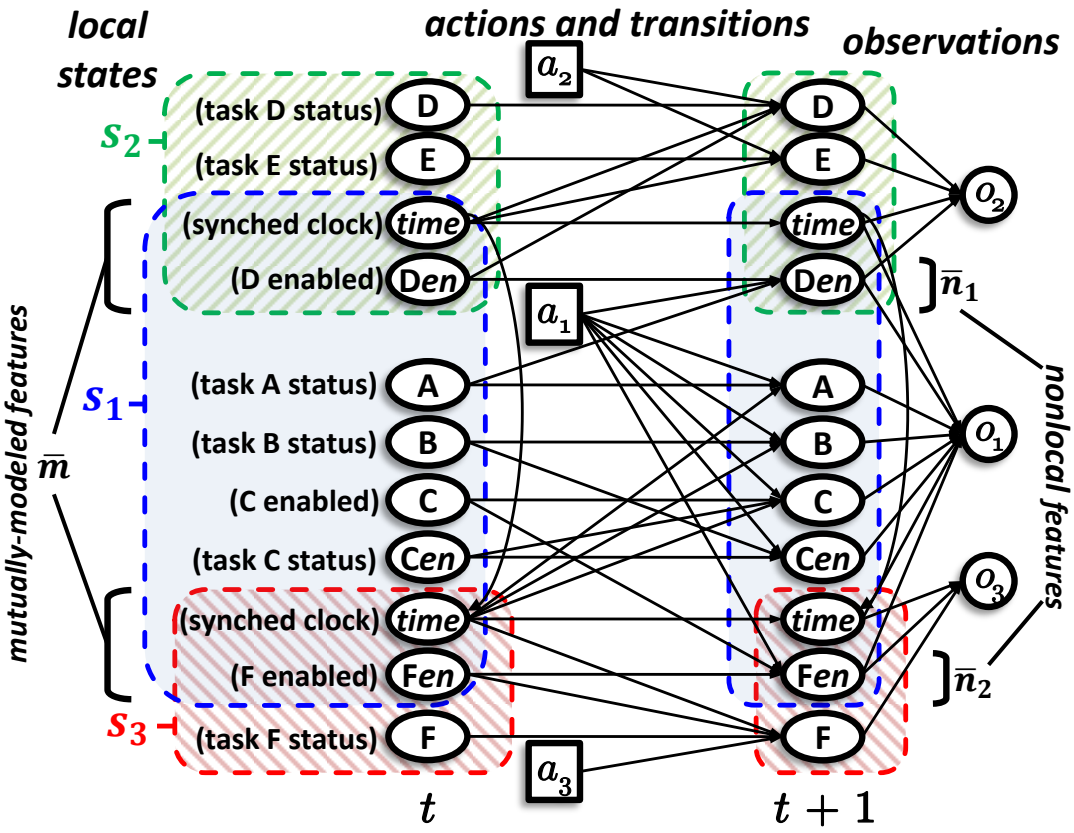


Figure 3.10: The TD-POMDP description for Example 3.31.

domains. The *state factor scope magnitude* is particularly useful because it serves as a measure of the size of the state space of the best response POMDP model.

Definition 3.40. An agent's **state factor scope magnitude** $\mathbb{X}_i$ is the product of the sizes of the domains of the features in $\mathcal{X}_i$.

Given this additional structure, we can refine our bound on computational complexity of the TD-POMDP.

Theorem 3.41. *The worst-case time and space complexity of solving a TD-POMDP problem is bounded by* $O\left(n \cdot \text{EXP}(\mathbb{X}_i^{max}) \cdot \|\Pi_i^{max}\|^{\omega}\right)$*, where* n *is the number of agents,* $\mathbb{X}_i^{max}$ *is the largest state factor scope magnitude of any agent,* Π_i^{max} *is the largest policy space of any agent, and* ω *is the induced width of the interaction digraph (Def. 3.36).*

Proof. The derivation of the complexity result from Observation 3.37 entails every best response computation requiring an $\arg\max_{\pi_i}$ to be taken, enumerating the local

policy space bounded by $\|\Pi_i^{max}\|$, for all combinations of policies of ω peers, yielding complexity $\|\Pi_i^{max}\|\|\Pi_i^{max}\|^{\omega}$ for each of the n agents. By replacing each best response calculation with one POMDP solution, we can substitute the first term $\|\Pi_i^{max}\|$ in the bound from Observation 3.37 with the complexity of solving a finite-horizon best-response POMDP, which is known to be $O(EXP(S) = EXP(\mathbb{X}_i^{max}))$ (Bernstein et al., 2002) given that the state space is bounded by $\mathbb{X}_i^{max}$ (by Definitions 3.39–3.40). □

For the TD-POMDP, it turns out that there is concrete measure of state factor scope encoded in the problem specification. As I derive in Section 4.2, there exists a best response model for any given TD-POMDP that requires agent i to consider only (1) the values of features from its local state (Definition 3.12) and (2) the history of values of features from its mutually modeled feature set (Definition 3.13). As such, for TD-POMDP problems, the scope magnitude can be replaced in Theorem 3.41 with $\mathbb{X} = \|S_j\|\|M_j\|^{T-1}$. For TD-POMDP problems that are *locally fully observable* (Definition 2.8), a slightly stronger result holds: complexity is polynomial in $\|S_j\|\|M_j\|^{T-1}$ (as I derive in Section 4.2.4).

Example 3.42. Let us examine the TD-POMDP specification for the problem depicted in Figure 3.8. Agent 2's local state consists of features {*time*, *task-D-execution-status*, *task-E-execution-status*, *task-D-enabled*}, of which *time* and *task-D-enabled* are mutually-modeled. The domain of *time* is the set of time steps $\{0, .., 6\}$ until the global horizon ($T = 6$). The domain of each task's execution status is {*not-started*,*started-at-time-0*,...,*started-at-time-5*,*completed*}, containing a total of 7 values. The last feature, *task-D-enabled* can be either *true* or *false*. As such, the size of agent 2's local state space is bounded by $\|S_2\| \leq (7 \cdot 7 \cdot 7 \cdot 2 = 686)$. The domain of agent 2's mutually modeled features is bounded by $\|M_2\| \leq (7 \cdot 2^7 = 896)$. Thus, we can bound agent 2's state factor scope magnitude as $\mathbb{X}_2 \leq 614,656$, the product of these two figures.

The relationship between complexity and scope magnitude supports an intuitive characterization: the smaller the portion of the world state that an agent observes and interacts with, the easier its local planning and reasoning becomes. Moreover, the fewer the interaction features that it shares with other agents, the easier the problem becomes.

3.5.2 Degree of Influence

Locality of agent interaction is an important aspect of TD-POMDP structure whose exploitation can lead to dramatic reductions in the complexity of optimal planning, but it is not the only important aspect. So far we have looked at which agents in the team can impact each others' decisions as well the features through which they interact. Next, I introduce a aspect of structure that characterizes the degree to which agents impact each others' decisions. With the addition of this metric, I extend the theory from Section 3.5.1 so as to refine the bound on worst-case TD-POMDP complexity.

Making use of Definition 3.34, for any two agents i and j, i is either decision-dependent on j or decision-independent of j (conditioned on some other agents' policies). Considering the rich space of dependencies that may exist between the two agents, a binary relation such as decision-independent lacks the precision to characterize weakly-coupled problems satisfactorily. For instance, i may be able to reason independently of some of j's decisions but not of others. Moreover, there may be circumstances under which j's decisions do not affect i' decisions.

Example 3.43. Returning to the problem depicted in Figure 3.8, Agent 2 is *decision-dependent* on agent 1, but only dependent on those decisions relating to the execution of "Task A". For instance, whether or not agent 1 idles for one time step or executes "Task B" (which will necessarily take 1 time step) is of no consequence to agent 2 as it plans its own decisions. Furthermore, after agent 1 has completed "Task A", any decisions that it makes cannot impact agent 2's decisions in any way. In other words, any two of agent 1's possible policies, π_1^x and π_1^y, that differ only in the decisions made after completing "Task A" will induce the same best response from agent 2.

Definition 3.44. Two policies, π_i^a and π_i^b, of agent i are **impact-equivalent**, denoted $\pi_i^a \underset{\bar{\pi}_K}{\overset{I}{\equiv}} \pi_i^b$, conditioned on some other agents' policies $\bar{\pi}_K$, if adopting π_i^b instead of π_i^a will not cause any other agent $j \notin K$ to change its best response decisions:

$$\pi_i^a \underset{\bar{\pi}_K}{\overset{I}{\equiv}} \pi_i^b \Leftrightarrow \left[\forall j \notin K, \mathbf{arg}\max_{\pi_j \in \Pi_j} V(\pi_i^a, \pi_j, \bar{\pi}_K) = \mathbf{arg}\max_{\pi_j \in \Pi_j} V\left(\pi_i^b, \pi_j, \bar{\pi}_K\right)\right]$$

Definition 3.45. An **impact equivalence class** $E_{i,x}^{\bar{\pi}_K}$ (subscripted with the agent

index i and class index x and superscripted by policies $\bar{\pi}_K$ of other agents K) is a set of impact-equivalent policies (conditioned on $\bar{\pi}_K$): $\forall \left\{\pi_i^a, \pi_i^b\right\} \in E_{i,x}^{\bar{\pi}_K}, \pi_i^a \underset{\bar{\pi}_K}{\overset{I}{\equiv}} \pi_i^b$.

In principle, an agent i's local policy space could be partitioned into disjoint equivalence classes, each of which can be thought to impact other agents in the system in a different way, and thereby each inducing a different best response. Figure 3.11 illustrates the equivalence class partitions in a simple two-agent problem. Notice that the influence classes in Figure 3.11 are each labeled using notation $E_{1,x}$ without any superscript. Here, $E_{1,x}$ specifies an *unconditional* equivalence class containing a subset of agent 1's policies all of which induce an identical best response from agent 2. There are no other agents in the system on which to condition the equivalence.

For problems with more that two agents, policies $\pi_1^a \in \Pi_1$ and $\pi_1^b \in \Pi_1$, for instance, may induce the same best response from agent 2 only under the condition that agent 3 adopts π_3^c. In this case, we would write $\pi_1^a \underset{\pi_3^c}{\overset{I}{\equiv}} \pi_1^b$. For example, whether or not a colleague (agent 1) expresses interest in collaborating may only cause a researcher (agent 2) to change her plans under the condition that a program funding manager (agent 3) allocates the necessary funds. Alternatively, agent 1 might influence two other agents, such that $\pi_1^a \in \Pi_1$ and $\pi_1^b \in \Pi_1$ evoke the same best response in agent 2, but different best responses in agent 3. In this case, π_1^a and π_1^b are not impact equivalent. By definition, two policies of agent i are impact-equivalent (and may in turn be partitioned into separate impact equivalence classes) only in the case that they evoke identical best responses in each and every other agent (not including those in the conditioned set K).

Definitions 3.44–3.45 enable the discussion of a spectrum of varying degrees of dependence. At one end of the spectrum, agent j is *decision-independent* (Def. 3.34) of agent i conditioned on agents K, meaning that all policies that i could adopt induce the same best response from agent j. This implies that, by Definition 3.44, any two policies of agent i are impact-equivalent conditioned on any polices $\bar{\pi}_K$ of agents K. Moreover, all of agent i's policies may be grouped into a single impact equivalence class (Def. 3.45). At the opposite end of the spectrum, agent j is decision-dependent on i in such a way that j's best response is highly sensitive to the policy that i adopts. At this extreme, no two policies of agent i are impact-equivalent and the minimum number of i's impact equivalence classes is equal to the size of its policy space $\|\Pi_i\|$.

In comparing the policy space to the impact equivalence class space, I am highlighting one of the primary intuitions of this work. When agents impact each other with some of their decisions but *not* all of their decisions, they do not need to jointly consider each and every joint policy. They really only need to coordinate the policy

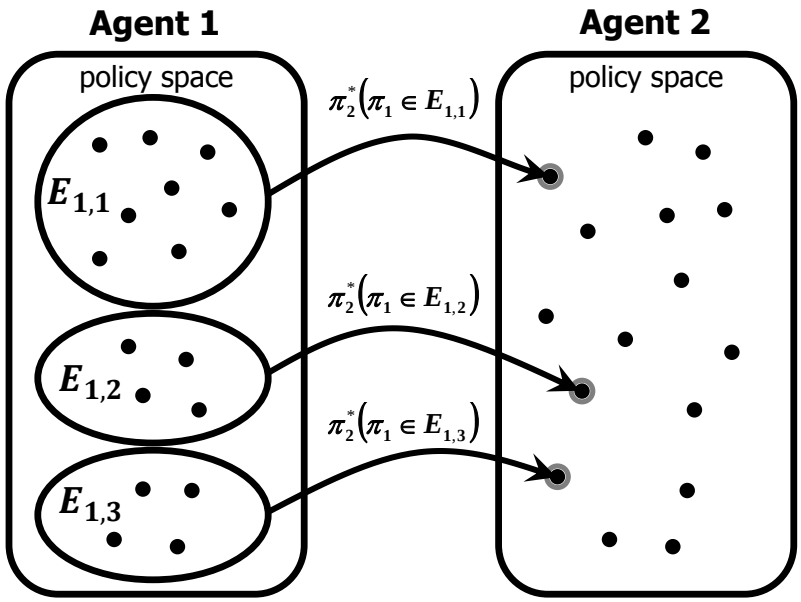


Figure 3.11: Example of equivalence classes.

decisions that matter, which are those that separate the different impact equivalence classes from one another.

Example 3.43 (continued). Consider the problem shown in Figure 3.8. Agent 1 and agent 2 interact when agent 1 completes Task A thereby *enabling* agent 2 to achieve a positive outcome quality for subsequent execution of Task D. Examination of the possible policies of agent 1 and the corresponding best responses of agent 2 reveals the following three impact equivalence classes:

• $E_{1,1}$**:** For any policy in which agent 1 begins Task A at time 0, agent 2's best response is *wait for one time step and then, at time 1, if Task D is enabled*[25] *begin Task D, but otherwise begin Task E.* The probability (0.5) that Task D is enabled at time 1 and the quality (12.0) of completing it before its deadline (4) are such that the potential benefit of waiting outweighs the potential loss associated with starting Task E a time step late. If D is not enabled at time 1, then agent 1 can infer that Task A has not yet completed and will not complete until time 3, which does not allow enough time to complete Task D before its deadline. In this case, it should not wait any longer but instead begin Task E at time 1.

• $E_{1,2}$**:** For any policy in which agent 1 begins Task A at time 1, agent 2's best response is *wait for two time steps and then, at time 2, if Task D is enabled begin Task D, but otherwise begin Task E.* The rationale is the same as before. In this case, Task A will complete at either time 2 or time 4 with equal probability. Completion at time 2 gives agent 2 enough time to complete Task D successfully. Agent 2's expected local utility using this best response is $(\frac{1}{2})(12.0) + (\frac{1}{2})(\frac{1}{3})(6.0) = 7.0$, which is better than it would do if it began Task E at time 0.

• $E_{1,2}$**:** For any policy in which agent 1 begins Task A any later than time 1, agent 2's best response is *begin Task* E *at time 0.* In this case, there is no chance that D will be enabled early enough for agent 2 to complete it before its deadline. Thus, agent 2 should begin its only other task, Task E, as early as possible to maximize the probability of completing it successfully.

The policies contained within each partition are trivially impact-equivalent with respect to agent 3's best response. (Since Agent 3 only has a single task to execute, its best response to any of agent 1's policies is simply to begin Task F as soon as F becomes enabled.)

By relating the magnitude of the equivalence class set to that of the local policy space, I can now extend the theory developed in Section 3.5.1. Recall the COP reformulation from Section 3.5.1.1, where the problem of optimal joint policy computation was reduced to selecting the best combination of values for variables $\{x_j\}$, each pertaining to an agent j's local policy. As suggested in Section 3.5.1, agents can solve the problem using *bucket elimination* by each iterating through the domains $D_{\neq j}$ of decision-dependent peers' local policy variables and computing a best response to each. The resulting set of best response policies is referred to in other work as agent j's *coverage set* (Becker et al., 2004b).

Definition 3.46. An agent j's **coverage set**, denoted $\mathbb{C}_i^{\bar{\pi}_K}(\Pi_j)$, with respect to agent i and a setting $\bar{\pi}_K$ of other peers' policies, is the set of policies that meets the following condition: $\forall \pi_i, \mathbb{C}_i^{\bar{\pi}_K}(\Pi_j) \cap \left[\underset{\pi_j \in \Pi_j}{\textbf{arg}\max}\, V(\pi_i, \pi_j, \bar{\pi}_K) \right] \neq \emptyset$.

The complexity results developed in Section 3.5.1.1 assume a brute force method for computing agent j's coverage set: enumeration of all possible combinations of

[25]For this TD-POMDP problem, "task-D-enabled" is a nonlocal feature that is completely observable to agent 2. As such, in any given state, agent 2 knows whether or not its Task D has become enabled, and can infer the corresponding outcome distribution.

peer agents' policies and computation of a best response to each. If several of a peer i's policies are impact-equivalent (conditioned on a particular setting of other peers' policies $\bar{\pi}_k$), then they will result in the same best response from agent j. Moreover, all policies $\pi_i \in E_{ix}^{\bar{\pi}_K}$ in an equivalence class will induce the same best response, suggesting the possibility of redundant best response calculations that could be avoided by taking into account equivalence class structure.

Lemma 3.47. *In order to compute an agent j's* coverage set $\mathbb{C}_i^{\bar{\pi}_K}(\Pi_j)$*, it suffices to compute a best response to a single (arbitrarily selected) policy π_{ix} from each of i's impact equivalence classes $E_{ix}^{\bar{\pi}_K}$.*

Lemma 3.47, which follows directly from Definitions 3.45 and 3.46, implies that the fewer the number of agent i's equivalence classes, the fewer the number of j's necessary best responses. We can use this result to refine the bound on TD-POMDP complexity, but before doing so, one other consideration must be addressed. Unlike the locality of agent interaction, which can be directly and trivially assessed from a TD-POMDP problem's interaction digraph, the problem's equivalence class structure is not known *a priori*. As such, the solution process must itself perform a partitioning of agents' local policy spaces into equivalence classes in order to take advantage of the underlying equivalence class structure.

Definition 3.48. An **impact equivalence partitioning scheme** $\mathbb{P}$ is a method that takes as input an agent i and a setting of some other agents' policies $\bar{\pi}_K$, and partitions agent i's local policy space into a set of disjoint impact equivalence classes $\mathbb{P}(i, \bar{\pi}_K) = \left\{E_{ix}^{\bar{\pi}_K}\right\}$. I denote the complexity of the partitioning scheme with a term $C_{\mathbb{P}}$, which refers to the worst-case computational complexity required for $\mathbb{P}$ to partition any agent's local policy space into equivalence classes conditioned on any other agents' policies.

There are a variety of different partitioning schemes that could be used to partition agents' local policy space, each involving a different amount of computational overhead and thus a different value of $C_{\mathbb{P}}$. In Chapter 5, I present a specific partitioning scheme grounded in my *influence-based policy abstraction* methodology (Chapter 4) and characterize its complexity. Another scheme requiring only constant time would be to do nothing, thereby leaving each individual policy $\pi_{ix} \in \Pi_i$ in its own partition. This scheme is valid in the sense that it creates classes of i's policies $\{E_{ix}\}$ that are equivalent, but it is not useful for reducing the number of best response calculations that j must perform.

Definition 3.49. For a given problem, the **degree of influence** $d_{\mathbb{P}}$ afforded by a partitioning scheme $\mathbb{P}$ (Def. 3.48), is the maximal ratio, for all agents and all combinations of peers, of the number of impact equivalence classes to the number of local policies:

$$d^{\mathbb{P}} = \max_{i=1,\ldots,n} \frac{\max_{\forall K, \forall \bar{\pi}_K} \left\| \mathbb{P}_i^{\bar{\pi}_K} \right\|}{\|\Pi_i\|} \tag{3.14}$$

Example 3.43 (continued). For the example problem shown in Figure 3.8, we derived earlier that a mere 3 partitions $\{E_{1,1}, E_{1,2}, E_{1,3}\}$ can be used to classify all of agent 1's possible policies. Even in this very simple problem, given the time horizon 6 and agent 1's three different tasks in addition to a "wait" action, agent 1 has a total of 483,729,408 possible policies.[26] The degree of influence (given this partitioning) is thus $\frac{3}{483,729,408} \approx 2.067 \times 10^{-9}$.

The degree of influence quantifies the (worst-case) reduction in best-response calculations achievable with a particular partitioning scheme. For any given problem, there is a minimal number of equivalence classes that, if found by a partitioning scheme, would minimize the degree of influence. However, the computation required to execute the partitioning scheme with the lowest degree of influence may be prohibitive, possibly canceling out the associated benefit of the reduced number of best response calculations. Thus, in selecting a partitioning scheme, it is desirable to achieve a balance in the degree of influence and the computational overhead of partitioning (across all problems that the partitioning scheme is expected to face). Theorem 3.50 sheds some light on the problem of finding such a balance.

Theorem 3.50. *The worst-case time and space complexity of solving a TD-POMDP problem is bounded by:*

$$O\left(n \cdot \mathrm{EXP}(\mathbb{X}_i^{max}) \cdot (d_{\mathbb{P}} \|\Pi_i^{max}\|)^{\omega} + n \cdot C_{\mathbb{P}} \cdot (d_{\mathbb{P}} \|\Pi_i^{max}\|)^{\omega - 1}\right) \tag{3.15}$$

where n is the number of agents, $\mathbb{X}_i^{max}$ is the largest state factor scope magnitude (Def. 3.40) of any agent, $d_{\mathbb{P}}$ is the degree of influence (Def. 3.49) given partitioning scheme $\mathbb{P}$, Π_i^{max} is the largest policy space of any agent, $C_{\mathbb{P}}$ is the worst-case complexity of $\mathbb{P}$, and ω is the induced width of the interaction digraph (Def. 3.36).

[26] The number of possible policies was calculated by multiplying together the number of available local actions in every reachable local state of the corresponding TD-POMDP.

Proof. The complexity bound presented in Theorem 3.50 follows from the analysis of my BE-OIS algorithm presented in Section 6.6.3.

□

Theorem 3.50 is an extension of the complexity results developed in Section 3.5.1. The differences between Equation 3.15 and the bound in Theorem 3.41 are (1) a reduction in the base of the exponent by a factor of $d^{\mathbb{P}}$ and (2) the addition of term $C_{\mathbb{P}}$ accounting for the computational overhead of $\mathbb{P}$ (in the context of the bucket elimination algorithm). The new bound suggests that, all else being equal, problems involving agents with a low degree of influence (whose local policy spaces can be partitioned into a relatively small number of influence equivalence classes) should be easier to solve than problems with a high degree of influence *contingent* upon the efficiency of impact equivalence partitioning. If the partitioning complexity $C^{\mathbb{P}}$ is bounded $O\left(d^{\mathbb{P}} \cdot \text{EXP}(\mathbb{X}_i^{max}) \cdot \|\Pi_i^{max}\|\right)$, the second term of Equation 3.15 vanishes. However, if it is of significantly larger magnitude, it overwhelms the first term, indicating that the overhead of partitioning potentially outweighs any computational benefit of the smaller local policy space search sizes.

Note that the bound given in Theorem 3.50 is no longer purely a statement about the problem. That is, it includes information, $C^{\mathbb{P}}$ and $d^{\mathbb{P}}$, specific to the algorithm that is used to solve the problem. As such, it is actually a bound on the complexity of exploiting influence structure, where the exploitation is inexorably tied to the solution algorithm.

3.5.3 Summary of Weak Coupling Characterization

Over the course of this section, I have developed theory relating TD-POMDP complexity to weakly-coupled problem structure. Subsections 3.5.1.2, 3.5.1.3, and 3.5.2 synthesize three key aspects of problem structure {*agent scope*, *state factor scope*, and *degree of influence*} into an integrated characterization of weak coupling. The end result is a refined bound on the worst-case time and space complexity of optimal planning presented in Equation 3.15 that accounts for the three weak coupling aspects. As I summarize in the list below, each aspect manifests itself in a different set of problem parameters, and each affects the overall complexity in a different manner.

- **Agent scope** refers, conceptually, to which agents in the system are affecting each others' decisions (and hence, which peers' influences need be reasoned about in order for the affected agent to plan its own behavior). In general, the fewer the agents that affect one another, the smaller the worst-case computation

time. With respect to agent scope, the strength of coupling of a problem can be quantified as the *induced width* ω of the interaction digraph, which bounds the number of peers that can affect any given agent. All else being equal, a smaller value of ω indicates a more weakly-coupled problem. In the context of a decoupled joint policy search method, worst-case computation time is exponential in ω (regardless of the total number of agents).

- **State factor scope** refers to the portion of world state features that must be reasoned about by an individual agent when planning its local policy (in the context of a decoupled search method). With respect to state factor scope, I quantify strength of coupling as the magnitude of the largest state factor scope magnitude $\mathbb{X}_i^{\max}$ (which is the largest number of combinations of values that may be taken by the features in an agents' state factor scope). In general, worst-case computation time is exponential in $\mathbb{X}_i^{\max}$. For TD-POMDP problems, $\mathbb{X}_i^{\max}$ is bounded by $\mathbb{X}_i^{\max} \leq \max_j \left[\|S_j\| \|M_j\|^{T-1} \right]$. Thus, weak coupling is directly related to the density of mutually-modeled state features $\bar{m}_j \subseteq s_j$ and the sizes of their joint domain $\|M_j\|$. For TD-POMDP problems in which agents directly observe their local state, worst-case computation time is polynomial in $\max_j \left[\|S_j\| \|M_j\|^{T-1} \right]$.

- For agents that do affect each others' decisions, the **degree of influence** relates to the proportion of different ways that they can affect each other's decisions. I have derived, in Section 3.5.2, that all of agent i's policies that have the same *impact* on agent j's decisions can (in principle) be grouped together, thereby partitioning agent i's local policy space so as to reduce the number of policy combinations that need be jointly considered by the group of agents. Given a *partition scheme* $\mathbb{P}$, parameter $d_{\mathbb{P}}$ quantifies a problem's degree of influence as the worst-case ratio of partitions to local policies. In regard to the worst-case time complexity bound, which is exponential in the induced width, $d_{\mathbb{P}}$ is situated in the base of the exponent and thus has a potentially-significant effect on computation time. More weakly-coupled problems, whose values of $d_{\mathbb{P}}$ are smaller, are likely to be easier to solve than problem with larger values of $d_{\mathbb{P}}$ (all else being equal). However, this statement is contingent on the efficiency of the partitioning scheme $\mathbb{P}$, whose complexity $C_{\mathbb{P}}$ affects the overall worst-case computation time polynomially.

The three problem parameters $\{\omega, \mathbb{X}_i^{\max}, d_{\mathbb{P}}\}$ can be thought of as orthogonal dimensions whose combination provides a concrete measure of the the degree of coupling of a TD-POMDP problem. A problem's worst-case complexity depends on where it lies along the spectrum of *agent scope*, along the spectrum of *state factor scope*, and along the spectrum of *degree of influence*. For any two problems, we can now compare their worst case complexities by evaluating (or estimating) the values of the three parameters and positioning each in the 3-dimensional space.

Evaluating the first two parameters, ω and $\mathbb{X}_i^{\max}$, is straightforward given the problem specification. The induced width ω may be obtained by converting the interaction digraph (whose connectivity is made explicit by the TD-POMDP description $\mathcal{M}$) into a constraint graph (as described in Section 3.5.1.2) and computing its induced width using one of the algorithms reviewed by Dechter (2003). The state factor scope magnitude $\mathbb{X}_i^{\max}$ may be computed by evaluating $\max_j \left[\|S_j\|\|M_j\|^{T-1}\right]$, whose terms are also explicitly described in $\mathcal{M}$. A problem's degree of influence $d_{\mathbb{P}}$ is not readily assessable from the TD-POMDP description, but it can be estimated heuristically. In Section 4.6, I supplement this theoretical analysis by proposing and evaluating several heuristics for estimating $d_{\mathbb{P}}$ that are specific to my influence-based abstraction scheme of partitioning.

Aside from the ability to compare problems' worst-case computation times, the theory that I presented in this section has broader consequences. In Section 3.3.2, where I proved the intractable general worst-case complexity of the TD-POMDP class, I argued that, given its explicit description of exploitable structure, the class contains regions wherein problems can be solved efficiently. Within my three-dimensional characterization of weakly-coupled problem structure lies a map for navigating the TD-POMDP class and uncovering those efficient regions. For each of the three dimensions, my analysis allows for determination of the worst-case complexity with respect to that dimension, and for borders to be drawn between regions that fall into one complexity class and those that fall into another. Worst-case complexity aside, my analysis provides guiding signs that suggest the orientation and slope of problem difficulty, and that can be used to better understand why some problems take hours to solve and others seconds. Finally, with respect to the degree of influence, the theory presented here justifies the exploration into influence-based abstraction that is the focus of this dissertation.

3.5.4 Related Work on Characterizing Weak Coupling

In contrast to related studies of weakly-coupled problems in sequential decision making, the primary distinction of my analysis is that it synthesizes several different aspects of weak-coupling into a single unified characterization. Each of these aspects have appeared individually in some shape or form in past work. For instance, the work of Guestrin et al. (2001, 2003) on exploiting restricted scope in factored value functions, though limited in context to approximate solution computation, plays a foundational role in my analysis. Dolgov & Durfee (2004a) analyze structure in graphical multiagent MDPs, relating agent scope to the complexity of optimizing local value functions. Another branch of work characterizes agent scope for systems of transition-independent agents, developing exploitative distributed joint policy search algorithms on which my analysis is based (Nair et al., 2005; Kim et al., 2006), and relating complexity of optimal planning to induced width (Kumar & Zilberstein, 2009). Oliehoek et al. (2008b) measures locality of interaction as a function of both agent scope and state factor scope, focusing on the complexity of joint planning stage-by-stage as a series of collaborative graphical Bayesian games. To my knowledge, the last aspect of weak coupling that I consider, *degree of influence*, has received no attention in the literature. However, the theory I have developed might explain the performance of the Coverage Set Algorithm Becker et al. (2004b), which effectively partitions each agent's local policy space (by way of a parametrization that encodes the effects of its policy on other agents' rewards).

Aside from the three aspects {*agent scope*, *state factor scope*, and *degree of influence*} on which my analysis concentrates, researchers have performed similar analyses relating other forms of Dec-POMDP problem structure to problem complexity. For instance, Goldman & Zilberstein (2004) characterize the complexity of various Dec-POMDP subclasses by classifying agents' communication capabilities, whether or not agents share any information during execution, and the agents' objectives (e.g., whether they are maximizing rewards or striving to reach a set of goal states). Shen et al. (2006) characterize complexity of optimal Dec-MDP planning according to the complexity of the minimal encoding of agents' local policies.

Allen (2009) takes a different approach to characterizing problem structure. With the motivation of predicting the amount of computation required by optimal solution methods and the value achieved by approximate solution methods, he develops an information-theoretic metric, *influence gap*, that quantifies roughly the difference in the degree to which each of two agents can affect world state transitions, joint observations, and joint rewards in a Dec-POMDP. The meaning of *influence* in Allen's

work is slightly different from that of my *degree of influence*, which refers to the degree to which agents can impact each other. However, his results express the same general sentiment that varying levels of impact result in varying problem complexity.

Lastly, there is a strong connection between my analysis and that of Brafman & Domshlak (2008). Whereas my analysis quantitatively characterizes weakly-coupled sequential decision making problems (specifically TD-POMDPs), Brafman & Domshlak (2008) quantitatively characterize "loosely-coupled" multiagent classical planning problems. Note that, although the Dec-POMDP is an optimization problem, the multiagent classical planning problem is one of satisfaction, whose solution is a joint plan that satisfies a set of goal conditions. In their analysis of complexity, Brafman & Domshlak (2008) take advantage of this fact to transform the planning problem into a constraint satisfaction problem (CSP). Much like in my analysis of joint policy computation as constraint optimization, they incorporate a parameter ω corresponding to the induced width of the constraint graph. They describe the level of coupling of a problem with one other variable δ that measures the number of potential coordination points (wherein an agent can affect others by adding a "public" action to its plan). Conceptually, this is similar to *degree of influence*, which, in the case that it is minimal, dictates the number of unique impacts that an agent can manifest on another.

3.5.5 Contribution Outside the Scope of the TD-POMDP

Researchers have developed a number of different algorithms for exploiting the kinds weakly-coupled problem structure included in my characterization (Becker et al., 2004b; Kim et al., 2006; Kumar & Zilberstein, 2009; Mostafa & Lesser, 2009; Nair et al., 2003, 2005; Oliehoek et al., 2008b; Witwicki & Durfee, 2010). A broader contribution of my characterization is that it can explain some of the trends observed in the performance of these algorithms that are not easily explained without considering combinations of weak coupling dimensions.

For instance, the successes of a family of ND-POMDP algorithms (Kim et al., 2006; Nair et al., 2005) in scaling to many agents has been attributed to the reduced agent scope associated with ND-POMDP agents' *local neighborhoods* (Kim et al., 2006; Kumar & Zilberstein, 2009; Nair et al., 2005). That is, as long as the agent scope remains small, these algorithms are expected to be practical. However, a generalized version of one of these algorithms (JESP (Nair et al., 2003)) has recently been reported as intractable for a test set of Distributed POMDPs with Coordination Locales (DPCLs) containing *just two agents*, even when generating an approximate solution (Varakantham et al., 2009). A likely explanation for this phenomenon is contained

within Equation 3.15, which suggests that it was not the agent scope of the problems that foiled JESP but instead the cost of JESP's best response calculation. Whereas ND-POMDP problems have an inherently restricted state factor scope due to the strict separation of agents' local states and transition and observation independence, DPCL problems involve transition-dependent agents that need to reason about each others' state variables in order to compute optimal best responses (which JESP employs in computing approximate solutions), making the DPCL more strongly coupled even in its two-agent incarnation.

3.6 Summary

The main contribution of this chapter is a model for multiagent coordination that emphasizes exploitable problem structure. While past work has defined a variety of other structured models, the TD-POMDP expresses structure *without* imposing overly-restrictive assumptions. In particular, the TD-POMDP accommodates rich *transition-dependent* agent interactions and partial observability, yet it is able to articulate aspects of structure rarely exploited outside *transition and observation independent* problems. The TD-POMDP's structure is significant because it decouples the joint model into a set of interdependent local POMDP models that are tied to one another by their transition influences. As a consequence, the TD-POMDP is a natural candidate for the application of decoupled solution algorithms that decompose the computation of joint behavior into a series of simpler computations of local behavior.

Despite the TD-POMDP model's inherently-decoupled representation, the generally-intractable computational complexity of the class of TD-POMDP problems brings into question the efficiency of decoupled solution formulation. Certainly, some TD-POMDP problems are intractable to solve, while others can be decomposed and solved efficiently. I call this latter group of problems *weakly-couped*. Fortunately, the problem structure expressed in the TD-POMDP's description provides clues as to the efficiency of solving any given problem. This insight has driven me to develop a characterization of weakly-coupled problems, and to derive refined bounds on worst-case computational complexity that accounts for three different aspects of weakly-coupled problem structure: *agent scope*, *state factor scope*, and *degree of influence*.

Whereas both *agent scope* and *state factor scope* have been analyzed in some shape or form in prior work (though in more restricted problem contexts), I am the first to formalize the *degree of influence* in any context. Unlike the other two aspects of weak coupling, the *degree of influence* is not immediately discernible from the problem

description. When exploited, however, my weak coupling theory suggests that a low degree of influence translates to significant improvements in computational efficiency. The promise of efficient solutions, along with the elusiveness of evaluating the degree of influences, motivates the development and evaluation of a methodology for exploiting influence structure. Such is the focus of the remainder of this dissertation.

CHAPTER 4

Influence-Based Policy Abstraction

In the last chapter, I claimed that the TD-POMDP's explicit representation of problem structure makes it a natural candidate for modeling and solving weakly-coupled problems efficiently. Guided by my characterization of *degree of influence* as well as *state factor scope*, I now begin to address these claims with the development of a methodology for exploiting the TD-POMDP's weakly-coupled problem structure. The primary insight of this chapter is that, when most agent decisions are independent of peers' decisions, the agents can avoid the complexity of coordinating their *full* policies. They can optimize their joint behavior by instead coordinating policy abstractions that convey only the essential influences. In connection with the theory presented in Section 3.5.2, influences summarize classes of *impact-equivalent* policies (Def. 3.45). Here, I examine *what* these influences are, *how* they can be represented compactly without loss of optimality, and *why* their coordination has potentially significant computational benefits over conventional policy search.

In answering these questions, this chapter contributes a formal characterization of transition-dependent influence. We find that, in the context of the TD-POMDP model, agent interaction can be conveniently modeled using probability distributions over present and past values of shared feature values. Further, this representation of influence suffices for formulating *optimal* joint policies. The influence derivation and proof of sufficiency presented herein serve as the theoretical foundation for the influence-based solution methods developed in subsequent chapters. Practically, and in connection with the remainder of the dissertation, this chapter also develops an important piece of the influence-based solution methodology: the model that allows each agent to efficiently compute its optimal local policy in response to promised influences of its peers.

Despite the fact that the influence formalism I develop here has no *direct* application outside of the TD-POMDP model, I believe that there is potential for farther-reaching

impact. For instance, the formalism could be extended to represent concurrent transition dependencies in the more general class of Dec-POMDPs. Moreover, the idea of reducing nonlocal policies to probability distributions over local effects *without* loss of information (sufficient for *optimal* reasoning) is itself a conceptual contribution. This insight could inspire the development of similar approaches to reasoning about uncertainty in multi-agent contexts other than Dec-POMDP planning.

4.1 Overview

In contrast to the general Dec-POMDP where each agent's behavior may be arbitrarily intertwined with all others', TD-POMDP agents are coupled to one another through structured feature dependencies between select individuals of the team. In particular, for weakly-coupled agents whose decisions are largely independent of one another (as in the interaction graph shown in Figure 3.7), the TD-POMDP provides a succinct representation for their interactions. We can, in turn, exploit this representation using a decoupled solution methodology (reviewed in Section 2.3.3) that decomposes the joint policy formulation into a series of local policy formulations. While substantial computational leverage has been obtained in using the decoupled approach to solve problems where agents are transition and observation independent (Becker et al., 2004b; Nair et al., 2005; Varakantham et al., 2007), less progress has been made in applying the same techniques to problems where agents interact through the transition model.[1]

Much of the difficulty in decomposing transition-dependent agents' policy formulations is due to the complexity of formulating and solving best-response models. As discussed in Section 4.2, computing a best response entails converting the joint problem into a single-agent POMDP, wherein the agent uses a *belief-state* representation to keep track of its knowledge about the system's trajectory as it takes actions and receives observations. In contrast to the single-agent POMDP belief state (Smallwood & Sondik, 1973), a Dec-POMDP agent's belief-state needs to include information that it gains about other agents' possible beliefs in addition t-o the information that it gains about the Dec-POMDP world state. For a general Dec-POMDP agent whose interactions are unrestricted, this entails maintaining a probability distribution over the possible observations of interacting peer agents (as derived by Nair et al., 2003) which is necessarily exponential in the number of peers. However, thanks to the

[1] Oliehoek (2010) has made recent progress in applying heuristic decomposition of value functions in factored Dec-POMDP, effectively decoupling the planning problem at each decisions step. However, as of yet, this approach produces only approximate solutions.

structure introduced in Chapter 3, TD-POMDP agents can make use of an alternative belief-state representation, as I describe in Section 4.2.2 and derive in Section 4.2.3. This novel belief-state representation consists of a vector whose size depends not on the number of peer agents, but instead on the state factor scope (described more precisely in Section 3.5.1.3), making it advantageous for weakly-coupled agents with sparse peer interactions.

What we find from the derivation of a TD-POMDP agent's belief state is that information about peers' behavior can be represented quite compactly in the form of a probability distribution over nonlocal feature values. Since the agent is influenced by peers only through nonlocal features, the transition dynamics of all nonlocal features constitutes a model of *influence*. In order to make optimal decisions, the agent does not need to know the details of peers' planned behavior as long as it knows the resulting influences. Before formally characterizing interagent influence in Section 4.3, I illustrate the high-level concepts with an example.

Example 4.1. Figure 4.1 portrays a simple, concrete example problem involving two rover agents. The rovers are each equipped with different hardware, so it is necessary for rover 5, upon visiting site C, to prepare the site in order for rover 6 to gain any value from visiting the site. Apart from this interaction, the two agents' problems are completely independent. Neither of them interacts with any other agents, nor do they share any observations except for the occurrence of site C's preparation and the current time. In a TD-POMDP, this simple interaction corresponds to the assignment of a single boolean nonlocal feature *site-C-prepared* that is locally-controlled by rover 5, but that influences (and is nonlocal to) rover 6. Thus, in planning its own actions, rover 6 needs to be able to make predictions about *site-C-prepared*'s value (influenced by rover 5) over the course of execution.

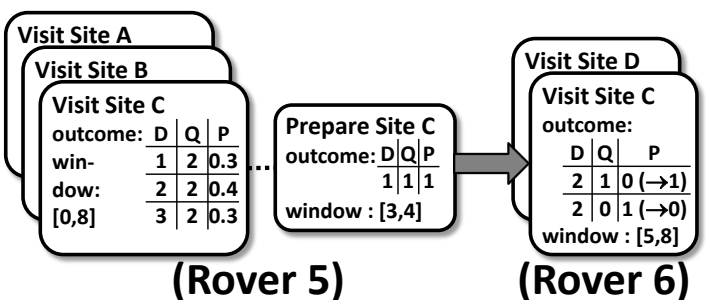


Figure 4.1: Example of limited influence.

Due to window constraints present in Example 4.1, the only information relating rover 6's behavior that is relevant to rover 5 is the probability with which *site-C-prepared* will become *true* at $time = 4$. At the start of execution, *site-C-prepared* will take on value $false$ and remain $false$ until rover 5 completes its "Prepare Site C" task (constrained to finish only at time 4, if at all, given the task window in Figure 4.1). After the site is prepared, the feature will remain *true* thereafter until the end of execution. With these constraints, there is no uncertainty about when *site-C-prepared* will become *true*, but only if it will become *true*. Hence, the influence of rover 5's policy can be summarized with just a single probability value, $Pr(\textit{site-C-prepared} = true | time = 4)$.

Aside from providing an elegant, compact[2] representation of nonlocal policy information, the *influence* abstraction (exemplified by $Pr(\textit{site-C-prepared} = true | time = 4)$) engenders a potentially-significant reduction in the size of the search space for optimal joint policies. As described more formally in Section 4.5, the *influence space* clusters together those individual policies of each agent that exert identical influences on the agent's peers. In Example 4.1, notice that any two policies that differ only in the decisions made after time 3 will yield the same value for $Pr(\textit{site-C-prepared} = true | time = 4)$. By considering only those influence values achievable by some feasible policy, agents avoid jointly reasoning about the multitude of local policies with equivalent influences. In Section 4.6, I corroborate this claim with an empirical evaluation of influence space size over a systematically-explored space of random problems.

4.2 Belief State and Influence

To decouple the joint policy computation into local policy computations, agents require local decision models that incorporate the influences of their peers' candidate policies. The purpose of the local model is to allow an agent to reason about the implications of its individual action choices given that all peers' choices are assumed to be determined. Doing so allows the agent to compute a best-response policy relative to its peers' promised policies. From the perspective of this decision-making agent, once its peers fix their policies, they cease to be decision makers and instead become processes of the stochastic environment. As such, the best-response model is really a local POMDP whose construction (as discussed in Section 4.2.1) is derived from the Dec-POMDP model, but whose observation signal is local (instead of joint) and

[2]Throughout this chapter, I use the word *compact* to refer to the fact that the encoding exploits weakly-coupled problem structure to express the necessary information in fewer parameters (than conventional representations).

whose action selection is local (instead of joint).

Due to the partial observability of POMDPs, the agent cannot track its current system state precisely. Instead, as is common practice when building and solving local POMDP models, the agent maintains a *belief state* that summarizes the knowledge it gains as it acts and observes the environment (Smallwood & Sondik, 1973). The belief state encodes information sufficient for the agent to make predictions and to select choices that are just as good as those that it could by remembering its complete action-observation history. Figure 4.2 depicts the use of a belief state in place of action-observation history.

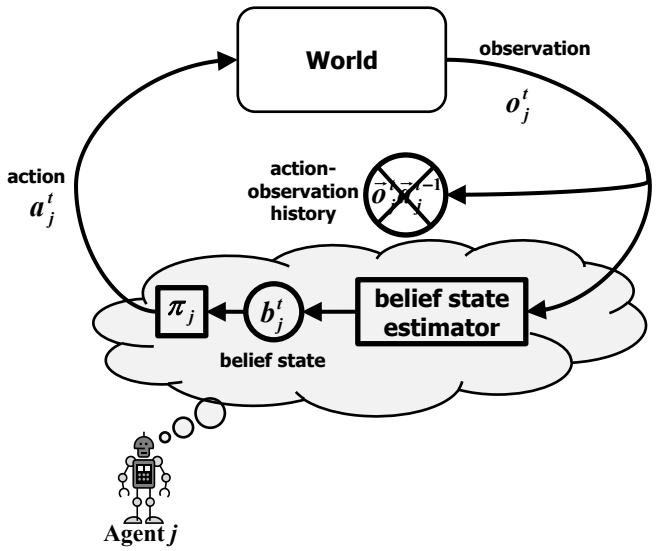


Figure 4.2: Usage of belief state for POMDP agent reasoning.

For the purposes of a best-response POMDP conditioned on fixed peer behavior, representation and maintenance of belief state are both nontrivial to operationalize and computationally complex. With the goal of reducing the complexity of best-response reasoning for TD-POMDP agents, this section develops a more efficient best-response model that exploits weakly-coupled problem structure. I begin in Section 4.2.1 by examining prior work (Nair et al., 2003) on belief state representations for agents whose peers' policies have been fixed. I refer to this general representation henceforth as the *General Best-Response Belief-State*. Next, in Sections 4.2.2–4.2.3, I derive a condensed version that takes advantage of the structure articulated by the TD-POMDP model, thereby improving computational efficiency of local (best-response) reasoning. Through this derivation, I reveal nonlocal policy information in Section 4.2.5 that forms the basis for my influence-based abstraction methodology (to which the remainder of

this chapter is devoted).

4.2.1 General Best-Response Belief State

Let us begin by considering an existing formulation of best-response belief state that Nair derived for the general class of Dec-POMDPs (which, at the time, he referred to as MTDPs). Given the fixed, deterministic policies of an agent's peers, the problem of finding an optimal local best-response policy may be represented using a complex but normal single-agent POMDP (Nair et al., 2003). Recall, from my review in Section 2.2.2.2, that the single-agent POMDP belief state (which I will denote $\boldsymbol{b}$) summarizes the agent's action-observation history with a probability distribution over possible world states: $b_j^t(s^t) = Pr\left(s^t | \vec{a}_j^{t-1}, \vec{o}_j^t\right), \forall s^t \in S$, where j is the agent, s^t is a possible current world state, and $\left\{\vec{a}_j^{t-1}, \vec{o}_j^t\right\}$ is the action-observation history. This particular belief state vector is a sufficient statistic for predicting future action-observation consequences in single-agent problems (Smallwood & Sondik, 1973). However, in the context of a best-response calculation, where peer agents are assumed to be executing fixed policies conditioned on their own partial observations, a distribution over (Dec-POMDP) world states is insufficient. Given that the agent's local observations (and actions) may be correlated with peers' observations, information gained to inform inference about peers' beliefs may be lost with the translation of local observation history to world state distribution.

Example 4.2. Consider two rovers (1 and 2) that receive a joint observation of wind at their base station, which provides partial information about the weather conditions at the various sites that they might choose to visit. To rover 1, observing *wind-felt-at-base* serves as an indication of the possibility that state feature *dust-storm-at-site-A=true*. Assume that rover 1 has planned a policy that is particularly sensitive to this observation, dictating that if it ever observes any wind, it will not travel to visit site A for the rest of the day. More precisely, it will not perform action *begin-trip-to-site-A* for any observation history containing observation *wind-felt-at-base*. In planning its behavior in response to rover 1's policy, rover 2 considers the scenario where it begins its day observing wind at base at 9:00 and then travels to site A, observing directly that *dust-storm-at-site-A=false* at 12:00. For this scenario, rover 1's policy dictates that, due to the existence of wind over base in the morning, it will not make a trip out to site A

at noon. However, rover 2 will not be able to perform this reasoning based solely on world state (*dust-storm-at-site-A*=false). Rover 2 will need to reason instead that, given its observations of wind three hours ago, rover 1 also observed wind and therefore should not be expected to visit (regardless of whether or not there is a dust storm at site A). The is because rover 1's beliefs about the world state differ from rover 2's beliefs.

Nair ensures that information about peer beliefs is not lost by augmenting the classical belief state with a probability distribution over peer observation histories (Nair et al., 2003):

$$b_j^t\left(s^t, \vec{o}_{\neq j}\right) = Pr\left(s^t, \vec{o}_{\neq j}^t | \vec{a}_j^{t-1}, \vec{o}_j^t\right), \forall s^t \in S, \forall \vec{o}_{\neq j}^t \in \Omega_1 \times ... \times \Omega_{j-1} \times \Omega_{j+1} \times ...\Omega_{j+1} \tag{4.1}$$

Equation 4.1 shows the *multiagent belief state* vector $\boldsymbol{b}_j^t$ that agent j associates with a given action-observation history ending at time t, where each component represents the probability of a unique world state (s^t) and combination of unique peer observation histories ($\vec{o}_{\neq j}^t = \{\vec{o}_i^t, \forall i \neq j\}$). By maintaining a joint distribution over world state and peer observation histories, agent j is able to keep track of its belief about the world state as well as the likelihoods of other agents' possible beliefs.

Use of any belief state representation requires the agent's ability to compute and update its belief state as it performs actions and receives observations. Using Nair's belief state update function (Nair et al., 2003), $BSU()$, agent j can compute the individual components of its belief state as shown in Equation 4.2. The initial belief state $\boldsymbol{b}_j^0$, conditioned on agent j's as-of-yet empty observation history, is simply the probability distribution of world start states dictated by the Dec-POMDP problem description. Subsequent belief states $\boldsymbol{b}_j^{t+1}$ are a function of previous belief state, local action, and local observation.

$$\begin{aligned}
\boldsymbol{b}_j^0 &= BSU\left(\emptyset\right) \quad = \langle Pr\left(s^0\right)\rangle \\
\boldsymbol{b}_j^{t+1} &= BSU\left(\boldsymbol{b}_j^t, a_j^t, o_j^{t+1}\right) = \left\langle Pr\left(s^{t+1}, \vec{o}_{\neq j}^{t+1} | \boldsymbol{b}_j^t, a_j^t, o_j^{t+1}\right)\right\rangle \\
&= \left\langle \frac{\sum_{s^t \in S}\left[b_j^t\left(s^t, \vec{o}_{\neq j}^t\right) Pr\left(s^{t+1} | s^t, \left\langle a_j^t, \pi_{\neq j}(\vec{o}_{\neq j}^t)\right\rangle\right) Pr\left(o_j^{t+1}, o_{\neq j}^{t+1} | s^t, \left\langle a_j^t, \pi_{\neq j}(\vec{o}_{\neq j}^t)\right\rangle, s^{t+1}\right)\right]}{\text{a normalization factor}} \right\rangle
\end{aligned} \tag{4.2}$$

With every new action agent j takes and observation it receives, it can calculate each belief state component as in Equation 4.2 by looping over all possible last world states s^t, for each adding the product of the three terms in the numerator, and

normalizing (since the component probabilities should all sum to 1). Calculation of these three terms is straightforward using the Dec-POMDP model and the fixed peer policies. The first term is the probability that the world was in state s^t at the previous time step and that the other agents had observed the subsequence of observations $\vec{o}^t_{\neq j}$ from times 0 to t equal to those in the respective belief state component (as dictated by the previous belief state). The second term is the probability that the current world state s^{t+1} is equal to that of the respective belief state component, conditioned on previous state and previous joint action composed of the action taken by j and the set of actions dictated by the other agents' fixed policies applied to their respective observation histories[3] (as computed using the Dec-POMDP transition function P). The third term is the joint probability of agent j's new observation o_j^{t+1} and the new set of observations for the other agents associated with the respective belief state component conditioned on the previous world state and joint action (as in the second term) and new world state. Through successive applications of its belief state update function (Equation 4.2), agent j can compute a belief state $\boldsymbol{b}_j^t$ given any sequence of actions and observations $\left\{a_j^0, ..., a_j^{t-1}, o_j^0, ..., o_j^t\right\}$. A simplified example of one such belief state trajectory is pictured in Figure 4.3.

Computing the best-response policy boils down to solving the fully-observable MDP defined over the space of belief states. Figure 4.3 shows just one path through this MDP. In general, there will be a branch for each combination of action that agent j can take and observation that agent j might receive. The reward signal R' (below) of the belief-state MDP is equal to the expected immediate rewards that the team would receive (given uncertainty of the true system state).

[3]This dissertation, as with Nair's work, considers and computes policies that are deterministic. (For any finite-horizon Dec-POMDP, there exists a deterministic joint policy that achieves the same value as the optimal randomized joint policy.) Under the assumption that each peer agent's policy is deterministic, it need not be conditioned on action-observation history, but only on observation history. This is because action history is uniquely determined from each observation history by stepping through the observations and selecting the deterministic action choice. Furthermore, computation of belief state relies on the property that peer policies map their observations histories to actions ($\pi_i : \vec{O} \mapsto A$), as opposed to mapping belief states to actions. Even though agents are planning their policies using the belief state representation, the dynamic programming algorithm Nair uses to compute best response policies enumerates all reachable observation sequences, recording those sequences which resulted in each reachable belief state, and thereby computing a policy that is a function of observation histories. Were peer policies defined over belief states and not observation histories, computation of this term would become much more complicated, ultimately requiring recursive invocation of peers' belief state update functions, which in turn would require invocation of their peers' belief state update functions.

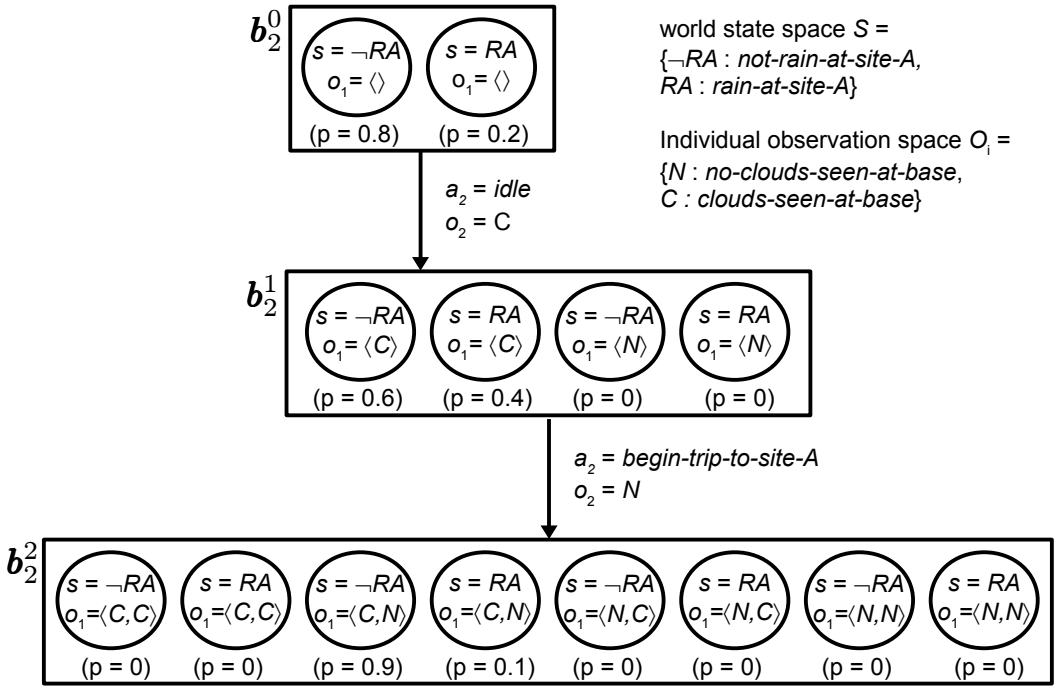


Figure 4.3: One possible belief state trajectory of rover 2 from Example 4.2.

$$
\begin{aligned}
R'\left(\boldsymbol{b}_j^t, a_j^t\right) &= \sum_{\left\langle s^t, \vec{o}_{\neq j}^t \right\rangle} \Bigg[\boldsymbol{b}_j^t\left(s^t, \vec{o}_{\neq j}^t\right) \\
&\qquad \cdot \textstyle\sum_{s^{t+1} \in S} Pr\left(s^{t+1} | s^t, \left\langle a_j^t, \pi_{\neq j}(\vec{o}_{\neq j}^t) \right\rangle\right) R\left(s^t, \left\langle a_j^t, \pi_{\neq j}(\vec{o}_{\neq j}^t) \right\rangle, s^{t+1}\right) \Bigg] \\
U^*\left(\boldsymbol{b}_j^t, a_j^t\right) &= R'\left(\boldsymbol{b}_j^t, a_j^t\right) \\
&\quad + \sum_{o_j^{t+1} \in \Omega_j} \left[Pr\left(o_j^{t+1} | \boldsymbol{b}_j^t, a_j^t\right) \cdot \max_{a_j^{t+1} \in A_j} U\left(BSU\left(\boldsymbol{b}_j^t, a_j^t, o_j^{t+1}\right), a_j^{t+1}\right) \right]
\end{aligned}
\tag{4.3}
$$

Equation 4.3, as derived by Nair et al. (2003), shows the calculation of immediate reward $R'()$, which agent j can calculate by invoking the Dec-POMDP model's reward function $R()$. The value U^* associated with each belief state and action pair is defined recursively as the sum of the immediate rewards and future rewards obtained by taking the optimal action in every subsequent belief state.

The belief state space is continuous, containing an infinite number of possible distributions over world states. However, given that the Dec-POMDP has a finite

horizon and finite state, action, and observation spaces (as is the case in the class of problems this thesis considers), there are a finite number of possible state transitions and observations, and hence a finite number of reachable belief states. Nair's best-response solution algorithm takes advantage of this fact by expanding only those belief states that are reachable.

Although there are a finite number of belief states, the belief state vector itself becomes computationally expensive to maintain as the problem size increases. A belief state encountered by agent j at time t contains $|S| \cdot \prod_{i \neq j} |\Omega_i|^t$ components. Under the assumption that all agents' individual observation spaces are bounded by $|\Omega_i|$, the worst-case space complexity is $O\left(|S| \cdot (|\Omega_i|^t)^{n-1}\right)$: exponential in the number of agents (as well as the problem time horizon). This exponential dependence carries over to the time complexity of any solution algorithm that performs component-wise belief updates (as in Equation 4.2, and depicted in Figure 4.3). Given that this computation is all directed towards computing a single agent's (best-response) policy, and the joint policy space is exponential in the number of agents, the cost of finding optimal joint policies in this manner is potentially doubly-exponential in the number of agents (in the worst case).

4.2.2 Condensed Belief State for TD-POMDP Agents

The *general best-response belief state* representation discussed in the previous section, while tractable for small problems involving two agents (as was demonstrated by Nair et al. (2003)), does not scale well to teams of three or more agents (as was shown empirically by Varakantham et al. (2009)). Though complete, its representation contains a significant amount of belief information that may be irrelevant for an agent in a weakly-coupled system. The intuition is that if most peer decisions have no bearing on the agent's local decision problem, then the agent need not distinguish most peer observation histories, nor distinguish most state information relating to peers' activities. The structured agent coupling of the TD-POMDP model leads us to define a representation of belief state that is more compact for such weakly-coupled cases, and whose compactness depends upon on the scope of interaction. In the extreme case of independent agents, the new belief state representation is equivalent to the traditional single-agent POMDP belief state (Smallwood & Sondik, 1973). This is accomplished by representing only that information which is necessary to make optimal decisions.

First, consider the observational information in Equation 4.1: $Pr\left(\vec{o}^{\,t}_{\neq j} | \vec{o}^{\,t}_j, \vec{a}^{\,t-1}_j\right)$. For the general Dec-POMDP agents, belief state includes a distribution over peer

observation histories because of potential correlation between local observations and peer observations (as was the case for the rover agents in Example 4.2). Although the general Dec-POMDP allows for arbitrary correlation of agent and peer observations, the TD-POMDP makes explicit the structure with which observations can be correlated. By Definition 3.5, each individual TD-POMDP agent j's observation is a function of its local state variables and local action. By Equation 3.3 in Definition 3.5, the only way that j's observation may be correlated with a peer i's observation is if there are state features common to both i's and j's local states. Moreover, the values of these mutually-modeled features are the only information that links the two agents' observations o_i and o_j. Instead of maintaining a distribution over all peer observation histories, a TD-POMDP can instead maintain a distribution over just those mutually-modeled state feature values. This is all that a TD-POMDP agent needs in order to make distinctions between different peer observations (based on its own observations).

The other information represented by the general best-response belief state (Equation 4.1) is the world state distribution: $Pr\left(s^t|\vec{o}_j^t, \vec{a}_j^{t-1}\right)$. This conveys the information necessary to predict how the system will progress from one time step to the next. For a TD-POMDP agent j, local state s_j models all relevant features; the consequences of j's actions are both determined solely by local state feature values and applied solely to (changes in) local state feature values. In planning its actions, maintaining a distribution over the subset of world state $s_j \in s$ (in conjunction with the distribution of mutually-observed state feature histories) should be enough to optimize its behavior (with respect to its local value function) given the fixed policies of its peers. This intuition leads to the following representation of belief state, which I prove in Section 4.2.3 sufficiently summarizes a TD-POMDP agent's action-observation history.

Definition 4.3. The **TD-POMDP belief state** for agent j, denoted $\boldsymbol{b}_j$, represents a joint probability distribution over current local state s_j and histories of mutually-modeled (Def. 3.13) features $\bar{m}_j$:

$$b_j^t\left(s_j^t, \vec{m}_j^{t-1}\right) = Pr\left(s_j^t, \vec{m}_j^{t-1}|\vec{a}_j^{t-1}, \vec{o}_j^t\right) \tag{4.4}$$

The mutually-modeled features $\bar{m}_j$ are the only world state features that may be mutually observable (Def. 3.2) because each is modeled in some other agent's local state, and the other agents' observations do not depend on state features outside of their local states, respectively. However, one should note that these features may be only partially observable, or equivalently, indirectly observable (from observations of dependent locally-controllable state features), or (in the degenerate case) completely

unobservable. Each feature f in tuple $\bar{m}_j$ fits into one of the following categories:

1. f is *locally-controlled* by agent j and thus modeled as a nonlocally-controllable feature by some other agent i.

2. From j's perspective f is *nonlocally-controlled*, and thus modeled by exactly one other agent i as a locally-controllable feature.

3. f is an *unaffectable* feature that impacts both i's and j's local state transitions.

The novelty of the TD-POMDP belief state representation is its exploitation of weakly-coupled problem structure. Unlike Nair's belief state representation, which is exponential in the number of agents, the length of the vector in Equation 4.4 is exponential in the number of mutually-modeled state features irrespective of the number of agents. For weakly-coupled problems where several agents interact through a (proportionally) small number of world features, this new belief state representation will be much more manageable than the general belief state.

The TD-POMDP belief state is updated in the same fashion as was the general belief state (described in Equation 4.2). The new belief state update function is as follows:

$$\begin{aligned} \boldsymbol{b}_j^{t+1} &= BSU\left(\boldsymbol{b}_j^t, a_j^t, o_j^{t+1}\right) = \left\langle Pr\left(s_j^{t+1}, \vec{m}_j^t | \boldsymbol{b}_j^t, a_j^t, o_j^{t+1}\right)\right\rangle \\ &= \left\langle \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t - \bar{m}_j^t} P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right) b_j^t\left(s_j^t, \vec{m}_j^{t-1}\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) : \text{a normalization factor}} \right\rangle \end{aligned} \tag{4.5}$$

I present a detailed derivation of Equation 4.5 in Section 4.2.3, and describe here the individual terms, contrasting them with those of the general belief state update function (Equation 4.2). The first term, $O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right)$, which is the probability of the new observation given the action taken and the world state encoded in the respective component of the belief state vector, roughly corresponds to the third term in Equation 4.2. Due to the factored TD-POMDP observations, local observation is not correlated with peer observations *except* in the values of the latest shared state features, so need not depend on other agents' actions nor observations, nor on world features outside of the local state.

The second, third, and fourth terms, $P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$, appear inside of a summation over possible last values of unshared features $(s_j^t - \bar{m}_j^t)$. The product of these terms constitutes the probability of new local state given last local state, mutually-modeled history, and action: $Pr\left(s_j^{t+1}|s_j^t - m_j^t, \vec{m}_j^t, a_j^t\right)$, but have been

factored into individual transition probability components according to Equation 3.10. The product roughly corresponds to the second term in Equation 4.2, but here need only represent the probability of new local state, and is consequently conditioned on a different set of past state and action information.

The last term, $b_j^t\left(s_j^t, \vec{m}_j^{t-1}\right)$, which too appears inside the summation, represents prior probability information encoded in the previous belief state, serving the same purpose as the first term in Equation 4.2, but invoking a different set of belief state information. This prior gets multiplied by the product of the previous three terms to compute $Pr\left(s_j^{t+1}|\boldsymbol{b}_j^t, a_j^t\right)$. Just as in Equation 4.2, the denominator of Equation 4.5 can be treated as a constant factor for normalization since the variables that it is conditioned on take on the same value for all nonzero components of the $\boldsymbol{b}_j^{t+1}$.

Most of the new $BSU()$ terms are straightforward to compute from the TD-POMDP problem description (via application of $O_j()$, $P_j^L()$, and $P_j^U()$). The only exception is $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$. This is also the only term that that depends upon peers' fixed-policy behavior. I will discuss this term further in Section 4.2.5 and Chapter 6. For the moment, assume this term is efficiently computable given the TD-POMDP model and the other agents' fixed policies.

Using the new belief state update function, agents maintain a different belief state representation, but the transition structure of the underlying MDP is congruent to that of Nair's belief-state MDP. That is, each possible action-observation pair maps to a belief-state-action pair in the belief-state MDP. The difference between the two representations is that Nair's belief state update function might group different action-observation sequences (together in one belief state) than would the TD-POMDP belief state update function. Let us rewrite the belief-state MDP's reward function $R'()$ (from Equation 4.3) using our new belief state representation:

$$R'_j\left(\boldsymbol{b}_j^t, a_j^t\right) = \sum_{\left\langle s_j^t, \vec{m}_j^{t-1}\right\rangle} \left[b_j^t\left(s_j^t, \vec{m}_j^{t-1}\right) \sum_{s_j^{t+1} \in S_j} \left[P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right) \cdot R_j\left(a_j^t, s_j^{t+1} = \left\langle \bar{l}_j^{t+1}, \bar{u}_j^{t+1}, \bar{n}_j^{t+1}\right\rangle\right) \right] \right] \tag{4.6}$$

Although the new belief-state MDP reward function $R'()$ has roughly the same form as that described in Equation 4.3, its output differs in one important dimension. Instead of associating *joint* rewards with the belief states and actions, the new reward function takes advantage of the TD-POMDP's reward decomposition to assign

immediate local values that are independent of the other agents' behavior (as given by $R_j()$). The repercussion of using this local reward valuation instead of a global reward valuation is that agent j's best-response policy maximizes its expected local utility instead of the expected joint utility. As proven in Chapter 6, this is satisfactory given that the encompassing search process entails all agents computing local best responses and combining their local utilities to evaluate each viably-optimal point in the joint policy space.

While it was straightforward to reason about peers' behavior and to see how their fixed policies were used in the general best-response belief state update, transition, and valuation from Section 4.2.1, it is less evident using the new belief state representation. However, we have isolated a single term common to the reward function (Equation 4.6) and belief state update function (Equation 4.5) that is dependent on nonlocal behavior: $Pr\left(n_j^{t+1}|\vec{m}_j^t\right)$. As developed in the next section, this term expresses the *influence* that is exerted on agent j by the other agents as they execute their policies. All other pieces of agent j's best-response model are independent of its peers' policies. Moreover, all other terms can be computed using only the local portions of the TD-POMDP model, thereby maintaining a separation of the individual agents' (potentially-private) information.

The primary benefit of this TD-POMDP belief state representation is its compactness and scalability. By taking advantage of the TD-POMDP's useful properties such as factoring of state observations and rewards that express agents' independence, and structured transitions that express their weakly-coupled dependence on their peers, we are able to derive a best-response model that is more efficient to maintain. Whereas the general best-response belief state representation grows exponentially with the number of agents, the worst-case space complexity of this new representation is $O\left(|S_j| \cdot |M_j|^t\right)$ (where M_j represents the domain of agent j's shared feature values) irrespective of the number of agents. Although a distribution of histories of shared features is maintained, this is expected to be much more compact than representing a joint distribution of observation histories for several other agents (which was maintained by the general best-response model). Together with the reduction in space complexity of the TD-POMDP best reponse belief state, the same reduction in time complexity ensues for any policy formulation method that performs component-wise belief state updates.

Furthermore, if the problem exhibits *local full observability* (Definition 2.8), the belief state representation need not represent a probability distribution. That is, if each agent's current observation fully dictates the current local state, all that must

be maintained is a unique history and not a probability distribution over all possible histories (since only one history will have positive probability).

4.2.3 TD-POMDP Belief State Sufficiency

Here, I prove the claim that the TD-POMDP agent belief state representation presented in Definition 4.3 sufficiently summarizes a TD-POMDP agent j's action-observation history $a^0....o^t$. Before descending into the proof, I begin with a supporting definition and lemma.

Definition 4.4. A belief state vector $\boldsymbol{b}_j^t$ is a **sufficient statistic** if it encodes all of the information gained by agent j as it executes from time 0 to time t required for making predictions about future information that will be gained after time t.

Lemma 4.5. *If, by maintaining a belief state vector $\boldsymbol{b}_j^t$ and forgetting its past actions and observations $\langle \vec{a}_j^{t-1}, \vec{o}_j^t \rangle$, agent j can accurately evaluate all future action-observation probabilities, then $\boldsymbol{b}_j^t$ is a* sufficient statistic*:*

$$\boldsymbol{b}_j^t \text{ sufficient}$$
$$\Longleftrightarrow$$
$$\forall \left\{ t \leq T, k \leq (T - t + 1), \vec{a}_j^{t+k-1}, \vec{o}_j^{t+k} \right\},$$
$$Pr\left(o_j^{t+k+1} | \vec{a}_j^{t+k-1}, \left\langle \vec{o}_j^{t+k}, a_j^{t+k} \right\rangle\right) = Pr\left(o_j^{t+k+1} | \boldsymbol{b}_j^t, \left\langle a_j^t, o_j^{t+1}, ..., a_j^{t+k-1}, o_j^{t+k} \right\rangle, a_j^{t+k}\right)$$

Proof. I prove this lemma by analyzing how information is gained by agent j. Prior to execution, j has its decision model (the TD-POMDP, in this case) and promised peer policies. The information that j obtains during execution from times 0 to t is that it performed a series of actions $\vec{a}_j^{t-1}$ and received a series of observations $\vec{o}_j^t$. Subsequently, from time t to time $t+1$, the only additional information gained is that action a_j^t resulted in observation o_j^{t+1}. A complete *information state* would therefore be a record of all actions taken and observations received.

If, as the premise of the lemma states, j can accurately evaluate future action-observation probabilities, then j can accurately evaluate the probabilities of all future information states. Any prediction that j might want to make must depend only on information state (and the prior information contained in the decision model and peer policies). Therefore j can make every prediction about future information (as accurately as it could have by recording its information state $\langle \vec{a}_j^{t-1}, \vec{o}_j^t \rangle$ exactly). By definition, $\boldsymbol{b}_j^t$ is a *sufficient statistic*. □

The result of Lemma 4.5 can be stated simply as follows. Because the agent interacts with the system only by performing actions and receiving observations,

(probabilistically) predicting future action-observations allows prediction of anything else that the agent could dream of predicting. In other words, agent j's belief-state MDP constitutes a generative model of future action-observation consequences. As such, this model suffices for the agent to plan optimal decisions given fixed policies of its peers. In proving sufficiency of the TD-POMDP belief state, we will also have proven that the TD-POMDP belief-state methodology enables computation of optimal local best-response policies.

Theorem 4.6. *The TD-POMDP belief state (Def. 4.3),* $\boldsymbol{b}_j^t = \left\langle Pr\left(s_j^t, \vec{m}_j^{t-1} | \vec{a}_j^{t-1}, \vec{o}_j^t\right)\right\rangle$, *is a sufficient statistic.*

Proof. By Lemma 4.5, to prove that the belief state representation $\boldsymbol{b}_j^t$ is a sufficient statistic, it suffices to prove that for any action-observation history $\left\langle \vec{a}_j^{t-1}, \vec{o}_j^t \right\rangle$, the probabilities of all future observations obtained by taking any future actions (given action-observation history) can be determined directly (and exactly) from the belief state vector. I prove this by reverse induction over history length t:

Base Case ($t = T$):

At the problem horizon (time T), agent j has taken all of the actions and received all of the observations already. Hence, there are no future predictions to be made. Trivially, $\boldsymbol{b}_j^T$ is a sufficient statistic for predicting the empty set of probabilities of future action-observation consequences.

Inductive Step:

Next, we derive that if $\boldsymbol{b}_j^{t+1}$ is sufficient for computing all future action-observation probabilities given $\left\langle \vec{a}_j^t, \vec{o}_j^{t+1} \right\rangle$, this implies that $\boldsymbol{b}_j^t$ must also be sufficient (given $\left\langle \vec{a}_j^{t-1}, \vec{o}_j^t \right\rangle$). The following equation expresses the belief state vector at time $t+1$.

$$b_j^{t+1}\left(s_j^t, \vec{m}_j^{t+1}\right) = Pr\left(s_j^{t+1}, \vec{m}_j^t | \vec{a}_j^t, \vec{o}_j^{t+1}\right) = Pr\left(s_j^{t+1}, \vec{m}_j^t | \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t, o_j^{t+1}\right)$$

by Definition 4.3, and expansion of the action-observation history vectors

$$= \frac{Pr\left(o_j^{t+1}, s_j^{t+1}, \vec{m}_j^t, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}{Pr\left(o_j^{t+1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}$$

by definition of conditional probability

$$= \frac{Pr\left(o_j^{t+1} | s_j^{t+1}, \vec{m}_j^t, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(s_j^{t+1}, \vec{m}_j^t, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}{Pr\left(o_j^{t+1} | \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}$$

by two applications of the definition of conditional probability

$$= \frac{O_j\left(o_j^{t+1} | a_j^t, s_j^{t+1}\right) Pr\left(s_j^{t+1}, \vec{m}_j^t, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}{Pr\left(o_j^{t+1} | \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}$$

by definition of the TD-POMDP local observation function O_j (Def. 3.5)

$$= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t \in S_j} Pr\left(s_j^{t+1}, s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}$$

by the law of total probability

$$= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t} Pr\left(s_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(s_j^t, \vec{m}_j^{t-1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}$$

by applications of the definition of conditional probability, and cancellation

$$= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t} Pr\left(s_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(s_j^t, \vec{m}_j^{t-1}|\vec{a}_j^{t-1}, \vec{o}_j^t, \cancel{a_j^t}\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)}$$

because current state is independent of future action

$$= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t} Pr\left(s_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) b_j^t\left(s_j^t, \vec{m}_j^{t-1}\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)} \tag{4.7}$$

by substitution of the appropriate belief state component (Definition 4.3)

We can further simplify Equation 4.7 by targeting the second term in the numerator, which specifies the conditional probability of the local state at time $t+1$ dependent on actions, observations, and various feature values at previous time steps. This term may be expanded as follows by taking into account the TD-POMDP's factorization of local state and local transition developed in Section 3.2.2. Recall that local state s_j is composed of locally-controllable features $\bar{l}_j$, unaffectable features $\bar{u}_j$, and nonlocally-controllable features $\bar{n}_j$.

$$\begin{aligned}
Pr\left(s_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) &= Pr\left(\bar{l}_j^{t+1}, \bar{u}_j^{t+1}, \bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) \\
&= Pr\left(\bar{l}_j^{t+1}|\bar{u}_j^{t+1}, \bar{n}_j^{t+1}, s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(\bar{u}_j^{t+1}, \bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) \\
&\qquad \text{by application of Bayes' rule} \\
&= P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) Pr\left(\bar{u}_j^{t+1}, \bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) \\
&\quad \text{by substitution of the factored local feature transition function } P_j^L \text{ (Eq. 3.10)} \\
&= P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) Pr\left(\bar{u}_j^{t+1}|\bar{n}_j^{t+1}, s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) Pr\left(\bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) \\
&\qquad \text{by application of Bayes' rule} \\
&= P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) \\
&\quad \text{by substitution of the unaffectable feature transition function } P_j^U \text{ (Eq. 3.10)} \\
&= P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right) \qquad (4.8)
\end{aligned}$$

The last step in the derivation of Equation 4.8 relies on the property that, given a history over mutually-modeled feature values $\vec{m}_j^t$, new nonlocally-controlled feature values $\bar{n}_j^{t+1}$ are conditionally independent of all remaining state feature values, local observation history, and local action history. It is straightforward to reason that $\bar{n}_j^{t+1}$ is independent of the latest local action. Since each feature in $\bar{n}_j^{t+1}$ is controlled by some other agent i, its value would depend only on i's latest action a_i^t (dictated by a fixed policy over i's past observations $\vec{o}_i^t$) and i's latest values of state features s_i^t, none of which can be affected by a_j^t until the next time step $t+1$ at the earliest. Justification of the other conditional independencies requires an intimate look at the factored structure of TD-POMDP feature transitions (described formally in Section 3.2.2).

Relationships among TD-POMDP variables may be represented graphically by 2-stage DBN in Figure 4.4, which divides all of the world state features into five distinct sets. The lower-most state variable, $s_{\subseteq j}$, represents those (unshared) features from agent j's local state which do not appear in any other agent's local state. Working our way upwards, agent j's mutually-modeled features $\bar{m}_j$ appear within the grey box, and are further decomposed into shared locally-controlled features $\bar{l}_j \subseteq \bar{m}_j$, shared unaffectable features $\bar{u}_j \subseteq \bar{m}_j$, and shared nonlocally-controlled features $\bar{n}_j \subseteq \bar{m}_j$. The state features that remain are features that appear in other agents' local states but not j's local state and are represented above the grey box as variable $s_{\neq j}$. The observations are also divided into agent j's observations o_j and those of the other agents $o_{\neq j}$. Actions are similarly divided, with all other agents' policies assumed to be fixed (because agent j is the one computing a best response). The connecting arrows follow from the definitions of the TD-POMDP state transitions and local observation function developed in Sections 3.2.1–3.2.2.

Captured within the DBN in Figure 4.4 are a number of different conditional independence relationships (Russell et al., 1996). The relationship that we will take advantage of is one of *direction-dependent separation* (Pearl, 1988), or *d-separation* for short. In review, a directed path from a node x to a node y is *blocked* given a set of *evidence* nodes E if the path contains a node in E. A set of *evidence* nodes E d-separates a node x from another node y if all paths between x and y are *blocked*. If E d-separates x from y, y is conditionally independent of x given E. From the shading of Figure 4.4, it is plain to see that every path leading from any node in $\{s_{\subseteq j}t, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\}$ to n_j^{t+1} passes through evidence set $\vec{m}_j^t$ (highlighted in grey). Hence, every node in the set $\{s_{\subseteq j}t, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\}$ is d-separated from n_j^{t+1} by evidence set $\vec{m}_j^t$ (highlighted in grey). This implies the conditional independence relationship: $Pr\left(\bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right) = Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$. Therefore, the last step in the derivation of Equation 4.8 holds.

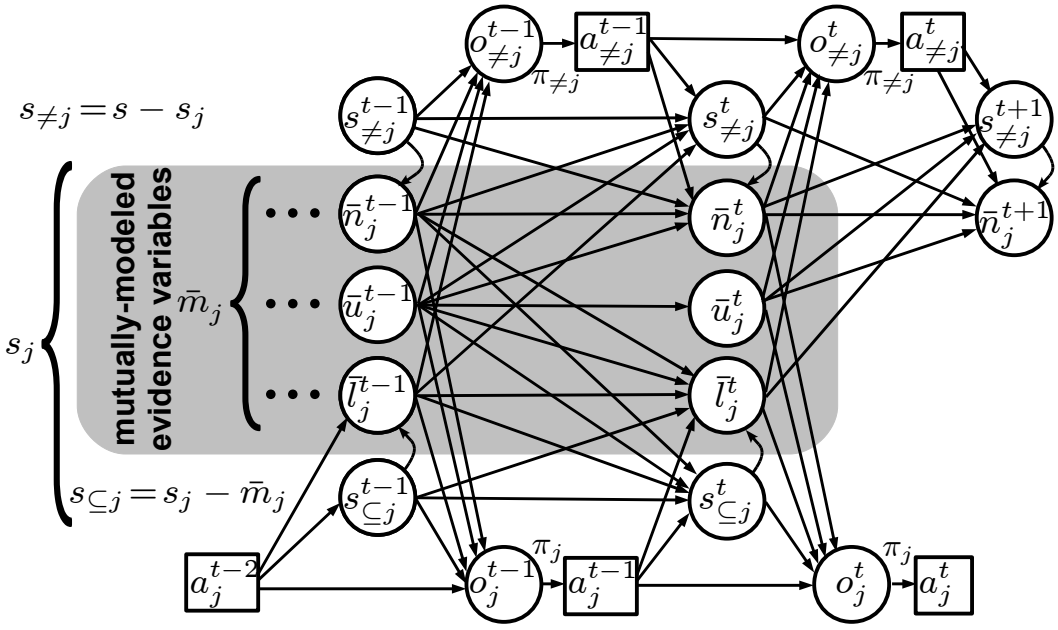


Figure 4.4: A DBN expressing CI relationships among TD-POMDP variables.

Intuitively, this conditional independence relationship is made possible by the nonconcurrency of agent influence. There is no path leading from action a_j^t to n_j^{t+1} because agent j cannot influence the transition probabilities of other agents' locally-controlled features until the next time step. This is reflected in the path leading from a_j^{t-2} to l_j^{t-1}, and continuing on to $s_{\neq j}^t$. Similarly, agent j cannot influence other agents' actions until first effecting a change in its local feature values.

Plugging Equation 4.8 back into Equation 4.7 results in the following simplified expression for belief state b_j^{t+1}:

$$b_j^{t+1}\left(s_j^{t+1}, \vec{m}_j^t\right) = Pr\left(s_j^{t+1}, \vec{m}_j^t | \vec{a}_j^t, \vec{o}_j^{t+1}\right)$$
$$= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t - \bar{m}_j^t} P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right) b_j^t\left(s_j^t, \vec{m}_j^{t-1}\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{t-1}, \vec{o}_j^t, a_j^t\right)} \tag{4.9}$$

The first thing to note is that the denominator in Equation 4.9 is equal for all components of the vector (because all are conditioned on the same action-observation history). Thus, it can be treated as a normalizing constant which is equal to the sum,

over all components, of their respective numerators:

$$Pr\left(o_j^{t+1}|\vec{a}_j^{t-1},\vec{o}_j^{t},a_j^t\right) = \sum_{\left\langle s_j^{t+1},\vec{m}_j^t\right\rangle}\left[O_j\left(o_j^{t+1}|a_j^t,s_j^{t+1}\right)\sum_{s_j^t-\bar{m}_j^t}P_j^L\left(\bar{l}_j^{t+1}|s_j^t,a_j^t\right)P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right)Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)b_j^t\left(s_j^t,\vec{m}_j^{t-1}\right)\right] \tag{4.10}$$

Turning back to Equation 4.9, determining belief state at time $t+1$ given action and observation history only involves computations of the five terms in the numerator. The first three terms are simply applications of the TD-POMDP agents' local observation function and local transition functions (contained in the TD-POMDP model). The fourth term is not so straightforward to calculate, but it does not depend on knowledge of the action-observation history. Nor does the fifth term. In fact, the numerator can be computed using only knowledge of the previous belief state b_j^t and without keeping track of the action-observation history. That is, next belief state is a function of current belief state and next action-observation pair. Equation 4.9 thereby serves as a belief state update function for TD-POMDP agents. Further, the derivation of Equation 4.9 implies that the process as defined over belief states is Markovian.

Recall that, for the purposes of this proof, we assumed that $\boldsymbol{b}_j^{t+1}$ was sufficient for determining probabilities of action-observation pairs from time $t+1$ onward. Under this assumption, $\boldsymbol{b}_j^t$ must be sufficient for determining probabilities of action-observation pairs from time t onward. Such predictions at time t (whereby an action a_j^t induces an observation o_j^{t+1}) are performed by Equation 4.10, which is based solely on belief state $\boldsymbol{b}_j^t$ without the need to remember past actions and observations. Predictions at time $t+1$ and beyond may be made by applying Equation 4.9 to determine the next belief state, which can in turn (given our inductive assumption) can be used to determine all action-observation probabilities from times $t+1$ to the end of horizon T. Thus, our inductive step holds. This completes the proof that for all values of t, $\boldsymbol{b}_j^t$ is a sufficient statistic for the action-observation history $\left\langle \vec{a}_j^{t-1},\vec{o}_j^{t}\right\rangle$. □

The implication is that agent j can make optimal decisions by basing its action choices solely on the probability distribution expressed by Equation 4.4. As it takes actions and receives observations, it can safely forget past actions and observations as long as it updates its new belief state from its previous belief state with every new observation (using Equation 4.9). Although every action-observation history maps to a single belief state, not every representable belief state (of which there are infinitely many) corresponds to an action-observation history. Just as with the general best-response belief state representation (Sec. 4.2.1), several histories may map to

the same TD-POMDP belief state. In other words, the reachable belief state space is potentially significantly smaller than the number of possible action-observation histories.

4.2.4 Complexity of Best Response Computation

In Section 4.2.2, my comparison against the general best-response belief state focused on the efficiency of updating the condensed TD-POMDP belief state representation, a result based solely on the relationship between the number of components in the belief state vector and the number of shared state features. There is yet another distinct, and arguably more significant computational advantage to using the TD-POMDP belief state relating to the overall complexity of planning best response policies. Recall that the purpose of instantiating the *belief state* formalism is to facilitate the solving of a single-agent POMDP model (as portrayed in Figure 4.2). That is, once an agent's peers' policies have been fixed, the peers become anonymous facets of a single-agent environment, and $\boldsymbol{b}_j^t$ encodes the agent's belief about the state of the single-agent POMDP that represents that environment. Using the condensed representation I developed, the best-response POMDP need only model a subset of those features from the original TD-POMDP world state.

To see this, let us rewrite the TD-POMDP belief state update equation (Eq. 4.5), replacing the belief state component index with a variable $x_j^t = \langle s_j^t, \vec{m}_j^{t-1} \rangle$:

$$
\begin{aligned}
b_j^{t+1}\left(x_j^{t+1}\right) &= BSU\left(\boldsymbol{b}_j^t, a_j^t, o_j^{t+1}\right) \\
&= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum\limits_{s_j^t - \bar{m}_j^t} P_j^L\left(\bar{l}_j^{t+1}|s_j^t, a_j^t\right) P_j^U\left(\bar{u}_j^{t+1}|s_j^t\right) Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}} \\
&\qquad \textit{by substitution of } x_j^t = \langle s_j^t, \vec{m}_j^{t-1} \rangle \textit{ into Eq. 4.5} \\
&= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t - \bar{m}_j^t} Pr\left(s_j^{t+1}|s_j^t, a_j^t, \vec{m}_j^{t-1}\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}} \\
&\qquad \textit{by collection of transition terms, given Eq. 3.10} \\
&= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t - \bar{m}_j^t} Pr\left(s_j^{t+1}, \vec{m}_j^t|s_j^t, a_j^t, \vec{m}_j^{t-1}\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}} \\
&\qquad \textit{because } \vec{m}_j^t \textit{ is included in the conditional information } \left\{s_j^t, \vec{m}_j^{t-1}\right\} \\
&= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{s_j^t - \bar{m}_j^t} Pr\left(x_j^{t+1}|s_j^t, a_j^t, \vec{m}_j^{t-1}\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}} \\
&\qquad \textit{by substitution of } x_j^{t+1} = \langle s_j^{t+1}, \vec{m}_j^t \rangle
\end{aligned}
$$

$$= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right) \sum_{\langle s_j^t, \vec{m}_j^{t-1}\rangle} Pr\left(x_j^{t+1}|s_j^t, a_j^t, \vec{m}_j^{t-1}\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}}$$

because, for the additional combinations of values of $\langle s_j^t, \vec{m}_j^{t-1}\rangle$ *considered by the summation,* $Pr(x_j^{t+1}|s_j^t, a_j^t, \vec{m}_j^{t-1})=0$

$$= \frac{Pr\left(o_j^{t+1}|a_j^t, s_j^{t+1}, \vec{m}_j^t\right) \sum_{\langle s_j^t, \vec{m}_j^{t-1}\rangle} Pr\left(x_j^{t+1}|s_j^t, a_j^t, \vec{m}_j^{t-1}\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}}$$

by Def. 3.5 and the conditional independence of observation o_j^{t+1} *on past state information given* $\langle a_j^t, s_j^{t+1}\rangle$

$$= \frac{Pr\left(o_j^{t+1}|a_j^t, x_j^{t+1}\right) \sum_{x_j^t} Pr\left(x_j^{t+1}|x_j^t, a_j^t\right) b_j^t\left(x_j^t\right)}{\text{a normalizing factor}}$$

by substitutions of x_j^t

(4.11)

Comparing the simplification in Equation 4.11 with the single-agent POMDP belief state (reviewed in Section 2.2.2.2), we see that the TD-POMDP best-response belief state is identical to that of a single-agent POMDP with state $x_j^t = \left\langle s_j^t, \vec{m}_j^{t-1}\right\rangle$. Moreover, the TD-POMDP best-response model is itself a POMDP with state $\left\langle s_j^t, \vec{m}_j^{t-1}\right\rangle$, a subset of the world state representation s_t of the joint decision model (Def. 3.15).

Observation 4.7. *The state of the single-agent POMDP used for the TD-POMDP best response need only represent features* $\left\{s_j^t, \vec{m}_j^{t-1}\right\}$.

Combining POMDP complexity theory from Section 2.2.3 with Observation 4.7, I deduce the following result.

Observation 4.8. *In the worst case, planning a best response for a TD-POMDP agent* j *requires time exponential in* $\|S_j\| \cdot \|M_j\|^{T-1}$, *denoted* $\text{EXP}\left(\|S_j\|\|M_j\|^{T-1}\right)$.

Simply put, the size of the state space of the best response POMDP is at most $\|S_j\| \cdot \|M_j\|^{T-1}$, and all known (general POMDP) solution algorithms have a worst-case time complexity exponential in the size of state space.

A stronger result holds for TD-POMDP problems with *local full observability* (Definition 2.8), wherein the belief state encodes the exact value of $\left\langle s_j^t, \vec{m}_j^{t-1}\right\rangle$ (instead of a probabilistic distribution over values). For such problems, the best-response model is an MDP, for which the complexity is known to be polynomial in the size of the state space.

Observation 4.9. *For a* locally-fully observable *TD-POMDP, the worst-case time to plan agent* j*'s best response is polynomial in* $\|S_j\| \cdot \|M_j\|^{T-1}$.

Observation 4.9 follows directly from the MDP's *polynomial* complexity (Papadimitriou & Tsitsiklis, 1987), a result reviewed in Section 2.2.3.

Just like the size of a TD-POMDP best-response belief state, the complexity of a TD-POMDP agent's best-response computation depends upon the number of shared state features $\|\bar{m}_j\|$ and the time horizon T but not necessarily on the amount of state information for the entire team of agents. For problems in which the world state space grows exponentially with the number of agents, I expect the computational savings afforded by the TD-POMDP best response model to be substantial, enabling scaling of the best response computation to larger problems with more agents than was possible with the general best response representation.

4.2.5 Influence Information

Aside from the complexity of local planning reasoning, another benefit of the TD-POMDP belief state is that it distinguishes nonlocal information dependent on other agents' policies from local information independent of other agents' policies. The nonlocal information serves as an abstraction of peers' policies and, as I develop in Section 4.5, facilitates an efficient partitioning of the nonlocal policy space into impact equivalence classes (Def. 3.45).

The only component of a TD-POMDP agent's best-response model (Section 4.2.2) that is dependent on the policies of the agent's peers is one term, $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$, found in both the transition probabilities (Eq. 4.5) and the reward function (Eq. 4.6) of the best-response POMDP. As such, the probability distribution $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$ represents exactly the information that j needs (prior to execution) in order to model (and compute best response policies to) planned behavior of its peers. All of the other information required for best-response decision making is contained within j's *local model* (Def. 3.16) and is independent of other agents' decisions.

The consequences of a peer agent i's decisions with respect to j manifest themselves exclusively in the values of $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$. As such, we call $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$ the *influence* of agent i on agent j.[4] By altering its planned policy, i might change its influence, in turn changing j's best response.

[4]If more than of agent j's peers jointly control the set of j's nonlocal features $\bar{n}_j$, then $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$ is the combination of influences of those peers on j, which I also refer to as the (*joint*) *influence*.

4.3 Characterization of Transition Influences

In the last section, I derived a condensed representation of belief state for TD-POMDP agents, deriving a local best response model and contrasting the size of its representation and the computational complexity of its employment (for formulating best responses) with those of the general best-response belief state representation. Another important distinction is that the TD-POMDP best response model is not seeded with fixed peer policies, but instead with fixed *influences*. That is, in order to compute a best response to agent i's policy π_i, agent j may not need to know all the details of π_i (which were necessary when using the general belief-state representation in Section 4.2.1). Instead it only needs to know the *influence* of π_i.

Definition 4.10. The **influence** of agent i's policy π_i on agent j, denoted $\Gamma^j_{\pi_i}$, is information summarizing π_i that is sufficient for agent j to plan a best response $\pi^*_j(\pi_i, \bar{\pi}_K)$ to π_i (and the policies $\bar{\pi}_K$ of j's other peers $K = \mathcal{N} - \{i, j\}$):

$$\forall j, \forall \bar{\pi}_K \in \left(\times_{k \in (\mathcal{N}-\{i,j\})} \Pi_k\right), \left[\pi^*_j\left(\Gamma^j_{\pi_i}, \bar{\pi}_K\right) = \pi^*_j(\pi_i, \bar{\pi}_K)\right].$$

The objective of influence-based abstraction is to reduce the amount of information that agents need to exchange and coordinate over (during planning). By abstracting away inessential details of agent i's policy π_i, $\Gamma^j_{\pi_i}$ should compactly encode the consequences of i's behavior as it relates agent j's decisions. With such an abstraction, agent i need not broadcast its full policy containing a multitude of decisions (exponential in the number of possible sequences of observations), nor disclose intimate details of plans that have no bearing on j's decisions.

Figure 4.5 depicts the usage of the influence-based abstraction, wherein agent j's best response calculation takes as input the influence $\Gamma^j_{\pi_i}$ abstracted from peer policy π_i and returns agent j's consequent optimal local policy $\pi^*_j\left(\Gamma^j_{\pi_i}\right)$.

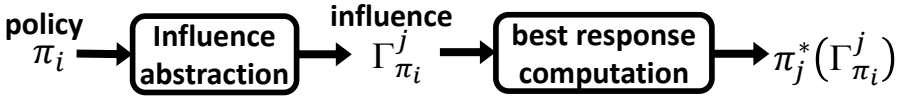


Figure 4.5: Abstracting *influences* from policies.

In the context of TD-POMDP agent coordination, the "best response" block refers to the solving of the POMDP model described in Sections 4.2.2–4.2.4, whereas details of the "influence abstraction" block are presented in Chapter 5. The focus of this section is on the identification of the content and structure of influence information.

The concept of influence-based abstraction is quite general, deriving inspiration from work in multiagent classical planning (Durfee & Lesser, 1991; Smith, 1980; Tambe, 1997; Xuan & Lesser, 1999) as well as methodologies for solving other specialized classes of Dec-POMDPs (Becker et al., 2004a; Musliner et al., 2006). Although the discussion here will remain centered around TD-POMDP agents' influences, the development of characteristics and formulations of influence models that follow may be more broadly applicable. This dissertation constitutes the first endeavor at a general characterization of influence in the context of sequential decision making, which subsumes several related influence models (noted and cited where appropriate).

In the subsections that follow, I systematically categorize influences, revealing a language through which agents can convey the policy information essential to coordination. My development of influence terminology culminates, in Section 4.3.5, with a formal characterization of a complete influence model for TD-POMDP agents.

4.3.1 Transition Influences

In the TD-POMDP, the only way that agent i can impact j is through the manipulation of nonlocal features. As such, information about the expected transitions of nonlocal features sufficiently summarizes π_i. Let us call this particular type of influence a *transition influence*.

Definition 4.11. The **transition influence** of TD-POMDP agent i's policy π_i on TD-POMDP agent j's nonlocal feature n_{jx} is a probability distribution $\Gamma^j_{\pi_i}(n_{jx}) = \left\langle Pr(n^{t+1}_{jx}|...) \right\rangle$ that serves as a sufficient summary of π_i for j to predict the (probability distribution over) values of n^{t+1}_{jx} for any action-observation history $\left\langle \vec{a}^{t-1}_j, \vec{o}^t_j, a^t_j \right\rangle$ that j may encounter.

By representing influences of peers' policies using probability distributions, agents can straightforwardly construct transition models for each of their nonlocal features. In general, modeling the transitions of a Dec-POMDP state feature would require a transition probability for every value of the feature conditioned on every feature of the world state and every joint action. However, given the factorization of TD-POMDP state, modeling the transitions of a TD-POMDP agent's nonlocal feature often requires substantially less information.

Example 4.1 (continued). Turning back to the 2-agent problem shown in Figure 4.1, consider the influence of rover 5's policy on rover 6, which may be modeled using a *transition influence* $\Gamma^{6}_{\pi_5}(\textit{site-C-prepared}) = \left\langle Pr(\textit{site-C-prepared}^{t+1}|...)\right\rangle$. In this particular problem, rover 6 does not need a complete probability distribution that is conditioned on all features. In fact, the only features that rover 6 can use to predict the value of *site-C-prepared* are *time* and *site-C-prepared* itself. Although *site-C-prepared* is dependent on other features from rover 5's local state, rover 6 cannot observe any evidence of these features except through its observations of *site-C-prepared* and *time*. Thus, all other features can be marginalized out of the distribution $Pr(\textit{site-C-prepared}|...)$.

Furthermore, the only influence information that is relevant to rover 6 is the probability with which *site-C-prepared* will become *true* conditioned on $\textit{time} = 4$. At the start of execution, *site-C-prepared* will take on value $false$ and remain $false$ until rover 5 completes its "Prepare Site C" task (constrained to finish only at time 4, if at all, given the task window in Figure 4.1). After the site is prepared, the feature will remain *true* thereafter until the end of execution. With these constraints, there is no uncertainty about <u>when</u> *site-C-prepared* will become *true*, but only <u>if</u> it will become *true* (at $\textit{time} = 4$). Hence, the influence of rover 5's policy can be summarized with just a single probability value, $Pr(\textit{site-C-prepared} = \textit{true}|\textit{time} = 4)$, from which rover 6 can infer all transition probabilities of *site-C-prepared*.

4.3.2 State-Dependent Influences

The influence in Example 4.1 (Figure 4.1) has a very simple structure due to the highly-constrained transitions of the nonlocal feature. By removing constraints, we can more generally categorize the influence between rover 5 and rover 6.

Example 4.1 (continued). Let the window of execution of "Prepare Site C" be unconstrained: $[0, 8]$. With this change, there is the possibility of rover 5 preparing site C at any time during execution. The consequence is that a single probability is no longer sufficient to characterize rover 5's influence. Instead of representing a single probability value, rover 6 needs to represent a probability for each time *site-C-prepared* could be set to true. In this case, a set of probabilities $\left\langle Pr(\textit{site-C-prepared}^{t+1} = \textit{true}|\textit{site-C-prepared}^{t} = false, \textit{time}^{t} = t), \forall t\right\rangle$ is required, each of which is conditioned on features *site-C-prepared* and *time*.

Definition 4.12. A transition influence $\Gamma_{\pi_i}(n_{jx})$ is **state-dependent** with respect to a subset of features $\bar{f} \subseteq s$ if its summarizing distribution need be conditioned only on $\bar{f}$'s latest value: $\Gamma_{\pi_i}(n_{jx}) = Pr\left(n_{jx}^{t+1} | \bar{f}^t\right)$.

The set of probabilities $\left\langle Pr(\textit{site-C-prepared}^{t+1} | \textit{site-C-prepared}^t, \textit{time}^t) \right\rangle$ in Example 4.1 is an abstraction of rover 5's policy that conveys both the probability of the interaction taking place and its potential timing. Definition 4.12 extends past development of more restrictive forms of state-dependent influences called *commitments*, which accounted for time but not probability (Musliner et al., 2006) or probability but not time (Witwicki & Durfee, 2007).[5] More generally, state-dependent influences may be conditioned on features other than *time*. For instance, as I describe in Example 4.13, an influence might need to be conditioned on other jointly-observable features such as *weather*.

4.3.3 History-Dependent Influences

Generalizing further, the probability of an interaction may differ based on both present and past values of state features.

Example 4.13. Consider the satellite and rover from Figure 3.1, and consider that they jointly observe a feature *weather* that is unaffectable, but that may affect their interaction. For instance, if it is cloudy in the morning, this prohibits the satellite from taking pictures, and consequently lowers the probability that it builds a path for the rover in the afternoon. Thus, by monitoring the history of the weather, the rover could anticipate the lower likelihood of help from thesatellite, and might change some decisions accordingly. Using Definition 4.14, we say that influence $\Gamma_{\pi_i}(\textit{path-A-build})$ is history-dependent with respect to feature *weather*.

Definition 4.14. A transition influence $\Gamma_{\pi_i}(n_{jx})$ is **history-dependent** w.r.t. features $\bar{f} \subseteq s$ if its summarizing distribution need be conditioned on the history of values of $\bar{f}$: $\Gamma_{\pi_i}(n_{jx}) = Pr\left(n_{jx}^{t+1} | \vec{f}^t\right)$.

Becker et al. (2004a) employ a special case of history-dependent influences in their Event-driven Dec-MDP solution algorithm, wherein agents augment their local decision models with event histories, thereby representing the probability of future nonlocal events conditioned on the histories of past events.

[5] I explore an extension of this special form of state-dependent influence in Section 7.2.

4.3.4 Influence-Dependent Influences

Transition influences may also be interdependent.

Example 4.15. For instance, in the interaction digraph in Figure 3.7, agent R4 has two arcs (labeled n_{4b} and n_{4c}) coming in from agent SAT3, indicating that agent 3 is exerting two influences, such as if agent SAT3 could build two different paths for agent R4. In the case that agent 3's time spent building one path leaves too little time to plan the other path, the nonlocal features n_{4b} and n_{4c} are highly correlated, requiring that their joint distribution be represented.

Definition 4.16. Two transition influences $\Gamma_{\pi_i}(n_{jx})$ and $\Gamma_{\pi_i}(n_{jy})$ are **influence-dependent** if $\Gamma_{\pi_i}(n_{jx}) = Pr\left(n_{jx}^{t+1}|...\right)$ need be conditioned on concurrent values of n_{jy} or *vice versa*, thereby necessitating a joint distribution $\Gamma_{\pi_i}(n_{jx}, n_{jy}) = Pr\left(n_{jx}^{t+1}, n_{jy}^{t+1}|...\right)$.

4.3.5 Comprehensive Influence DBN

With the preceding terminology, I have systematically introduced an increasingly comprehensive characterization of transition influences. A given TD-POMDP influence might be history-dependent with respect to one feature and state-dependent with respect to another. There may also exist chains of influence-dependent influences.

Example 4.17. In Figure 3.7, agent R7 models two nonlocal features, one (n_{7a}) influenced by agent SAT1 and the other (n_{7b}) influenced by agent R6. The additional arc between agents SAT1 and R6 forms an undirected cycle that implies a possible dependence between n_{7a} and n_{7b} by way of n_{6b}. The only way to ensure a complete influence model is to incorporate all three influences into a joint distribution.

In general, for any team of TD-POMDP agents, their influences altogether constitute a Dynamic Bayesian Network (DBN) whose variables consist of the nonlocal features as well as their respective dependent state features and dependent history features. Figure 4.6 illustrates the *influence DBNs* for the four examples presented in this chapter along with their implied conditional probability tables (CPTs). Once all of the influences associated with an agent i's nonlocal features have been decided, i

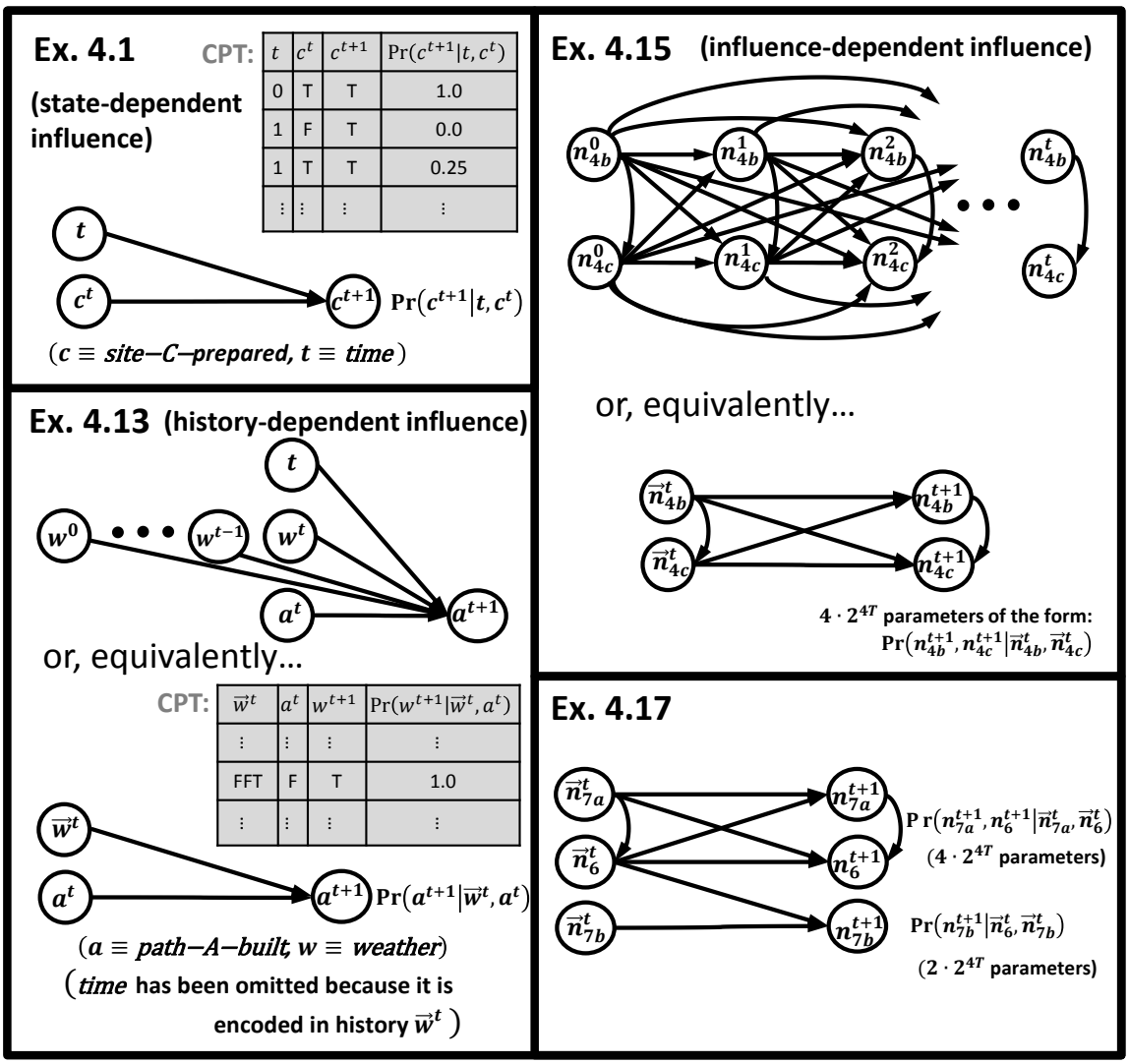


Figure 4.6: The influence DBN for each previously-presented example.

can extract the corresponding conditional probabilities and inject them into its local best-response model (replacing the term $Pr\left(\bar{n}_i^{t+1}|...\right)$ identified in Section 4.2.5).

The connections between the variables in the *influence DBN* are dictated by my characterization of state-dependence, history-dependence, and influence-dependence. That is, the scope of a variable n_{ix}^{t+1} in the influence DBN (which refers to the subset of variables $\bar{f}$ for which an arrow is drawn from f to n_{ix}^{t+1}) is such that the DBN encodes sufficient information for agent i to model the probabilities of n_{ix}'s transitions *given* that i's peers hold their policies still.

Note that the *influence DBN* is very different from the DBN that I presented in Section 4.2.3 (Figure 4.4) to describe the conditional independencies in the joint

model. I will refer to the previously-described DBN as the *TD-POMDP DBN* because it represents all variables in the TD-POMDP model. In contrast, the influence DBN has a smaller width than the TD-POMDP DBN, modeling only the transitions of nonlocal features. However, in the case of history-dependent influences, the influence DBN has a greater depth of connectivity than does the TD-POMDP DBN, connecting variables indexed with time step $t+1$ to those indexed with t, with $t-1$, with $t-2$, an so on.[6]

Given history- and influence-dependence, the influence DBN could potentially grow to be more complex than the TD-POMDP DBN, encoding a larger number of probability parameters to encode than there are elements in the TD-POMDP transition matrix. Indeed, as TD-POMDP agents' interactions become more complicated, more and more parameters involving more and more variables are needed to encode their effects. However, due to the TD-POMDP's decomposable transition structure, the DBN need contain only those critical variables that link the agents' POMDPs together.

Theorem 4.18. *For any given TD-POMDP, the influence $\Gamma^j_{\pi_i}(n_{jx})$ of agent i's policy π_i on agent j's nonlocal feature n_{jx} need only be conditioned on histories of mutually-modeled features $\vec{m}_j$ (Def. 3.13).*

Proof. This follows directly from the proof of belief state sufficiency presented in Section 4.2.3. In review, the belief state $\boldsymbol{b}^t_j = \langle Pr\left(s^t_j, \vec{m}^{t-1}_j | \vec{a}^{t-1}_j, \vec{o}^t_j\right), \forall s^t_j, \vec{m}^{t-1}_j \rangle$ was proven to be sufficient for computing agent j's best response using the update rule: $\boldsymbol{b}^{t+1}_j = \left\langle \frac{O_j\left(o^{t+1}_j | a^t_j, s^{t+1}_j\right) \sum_{s^t_j - \bar{m}^t_j} P^L_j\left(\bar{l}^{t+1}_j | s^t_j, a^t_j\right) P^U_j\left(\bar{u}^{t+1}_j | s^t_j\right) Pr\left(\bar{n}^{t+1}_j | \vec{m}^t_j\right) b^t_j\left(s^t_j, \vec{m}^{t-1}_j\right)}{\text{a normalization factor}} \right\rangle$. Here, the first term is j's local observation function (Def. 3.5), the second and third terms are locally-dependent components of j's local transition function (Def. 3.14), and the last term is j's previous belief state. The remaining term, $Pr\left(\bar{n}^{t+1}_j | \vec{m}^t_j\right)$, is the only one that depends upon peers' policies. Thus, $Pr\left(\bar{n}^{t+1}_j | \vec{m}^t_j\right)$, serves as a sufficient summary of i's policy for computing j's best response. □

Corollary 4.19. *The influence DBN grows with the number of mutually-modeled state features irrespective of the number of local state features and irrespective of the number of agents.*

By Theorem 4.18, it is sufficient (though not always necessary) for agents' influences to encode the histories of state features that are shared among agents. Moreover,

[6] Essentially, the influence DBN is the result of variable elimination performed on the TD-POMDP DBN. In particular, a variable x_i (which may refer to a feature in agent i's local state, agent i's action, or agent i's observation) that is eliminated is marginalized out due to the fact that is it *unobservable* to all other agents in the system except through the observations of mutually-modeled features. The inclusion of history features in the influence DBN is the direct result of such elimination.

the complexity with which an agent models its peers is controlled by the tightness of coupling with respect to state factor scope (Def. 3.39), and *not* by the complexity of the peer behavior, nor by the number of peer agents.

Despite this result, the influence DBN could still become too complex for TD-POMDP agents to use effectively. Example 4.20 describes an extreme case, wherein the conditional probability table grows unwieldy. The size of the conditional probability tables associated with the influences from Examples 4.15 and 4.17 are characterized in Figure 4.6.

Example 4.20. Consider an example problem for which 10 out of 11 state features in agent i's local state are nonlocal features in agent j's state, and hence mutually-modeled by agent j. In this case, in order to capture the possible effects of the 11th unobservable feature, agent j's sufficient encoding of i's influence would include the histories of all 10 mutually-modeled features. Given a time horizon of length T, and under the assumption that all features are boolean, there are $2^{(10T)}$ possible combinations of mutually-modeled histories and 2^{10} combinations of joint mutually-modeled feature values. In this case, the specification of the influence DBN would require on the order of $2^{(10T+10)}$ parameters.

Aside from the space complexity of storing the CPT, the ramifications of a large influence encoding are as follows. First, as per Observation 4.8, in the worst case, the computation required to compute a best response grows with the amount of information on which the influence is conditioned. Second, as I describe in Chapter 5, using my mixed-integer linear programming (MILP) methodology, the number of MILPs that must be solved in order enumerate feasible influence settings grows linearly with the number of parameters that encode the influence. Lastly, as my empirical results presented Section 4.6.2.4 suggest, the number of feasible influence settings tends to grow with the size of the influence encoding (regardless how each influence setting is found).

Given the various forms of growth in computational complexity associated with large influence encodings, it is important to identify conditions under which influence encodings remain compact. Along these lines, I describe one set of conditions under which we can avoid history dependence in the next section.

4.4 A Special Case: Influences on Event-Driven Features

The characterization of TD-POMDP transition influences that I developed in Section 4.3 was extremely general, encompassing all interactions that one TD-POMDP agent might have with another. I now focus on one particular class of interactions– those that can be represented with *event-driven*[7] features. After defining this class of interactions, I derive conditions under which agents can encode their influences on event-driven features compactly. In particular, I prove that when the interaction digraphs contain no cycles (undirected or directed), influences on event driven features are state-dependent and not history-dependent.

Definition 4.21. An **event-driven feature** f is a Boolean state feature that encodes the occurrence of an event, such that $Pr(f^{t+1} = false | f^t = true) = 0$.

The condition in Definition 4.21 means that the transition of an *event-driven feature* f is restricted such that the f can change from $false$ to $true$ (from one state to the next) but never from $true$ to $false$. Intuitively, if the corresponding *event* has not occurred, $f = false$. Once the event occurs, f changes to $true$ and can never thereafter return to $false$. Features of this type have appeared in several of the example problems that I have presented thus far (e.g., Examples 3.1, 3.31, and 4.1).

Definition 4.22. An **event-driven interaction** in a TD-POMDP refers to one or more **event-driven** nonlocal features through which one agent affects another.

Example 4.23. In the problem from Example 4.1 and Figure 4.1, rover 5 interacts with rover 6 by completing a task "Prepare Site C" and thereby altering future outcomes of rover 6's own tasks. This interaction is *event-driven* because it can be represented with a nonlocal feature *site-C-prepared*$\in \{true, false\}$ that encodes the rover 5's completion of site C preparations. Inherently, once task "Prepare Site C" is completed, the task (and underlying feature value change) cannot be undone.

[7] I adopt the term *event-driven* from the work of Becker et al. (2004a), who defined a Dec-POMDP subclass called the *Dec-MDP with event-driven interactions* (or the EDI-Dec-MDP). In my definitions, I present a slight generalization of Becker et al.'s semantics so as to accurately define *event-driven* interactions in my more general TD-POMDP problem class.

Theorem 4.24. *For a TD-POMDP problem whose nonlocal features are all event-driven and whose interaction digraph (Def. 3.27) contains no directed or undirected cycles, each influence $\Gamma(n_{jx})$ on a nonlocal features n_{jx} has the following properties:*

1. *For any nonlocal feature $n_y \not\equiv n_{jx}$, $\Gamma(n_{jx})$ need not be conditioned on n_y.*
2. *$\Gamma(n_{jx})$ is state-dependent (but <u>not</u> history-dependent) with respect to n_{jx} (such that encoding $Pr(n_{jx}^{t+1}|n_{jx}^{t})$ suffices).*

Proof. I address properties (1) and (2) separately.

1. To prove that property 1 holds, let us consider three disjunctive cases:
   <u>case a</u>: $n_y \notin \bar{m}_j$. By Theorem 4.18, $\Gamma(n_{jx})$ need only be conditioned on features in agent j's mutually modeled feature set $\bar{m}_j$ (Def. 3.13). Thus, $\Gamma(n_{jx})$ need not be conditioned on n_y.
   <u>case b</u>: $n_y \in \bar{m}_j \wedge n_y \in \bar{n}_j$. By Definition 3.12, $n_y \in \bar{n}_j$ refers to the fact that n_y is controlled by another agent, which we will call agent i, and affects agent j. Feature n_{jx} is also controlled by another agent, which we will call agent k, and affects agent j. We can deduce that $i \neq k$ from the acyclicity of the interaction digraph. If i and k were the same agent, this would mean two edges leading from node i to node j, constituting an undirected cycle. Further, we can deduce that $i \notin \Lambda_k$ (agent i is not a digraph ancestor of agent k, using Definition 3.28), because this would indicate an undirected cycle containing nodes i, j, and k (i.e. $i \in \Lambda_k$, $i \in \Lambda_j$, and $k \in \Lambda_j$). Hence, by Theorem 3.32, agent i cannot affect the value of n_{jx}, through its control of n_y or otherwise. Thus, n_{jx}^{t+1} is independent of $\vec{n}_y^t$, and $Pr(n_{jx}^{t+1}|\vec{n}_y^t, ...) = Pr(n_{jx}^{t+1}|...)$. Therefore, $\Gamma(n_{jx})$ need not be conditioned on n_y.
   <u>case c</u>: $n_y \in \bar{m}_j \wedge n_y \notin \bar{n}_j$. By Definition 3.12, $n_y \notin \bar{n}_j$ refers to the fact that feature n_y is controlled agent j (and is hence a nonlocal feature to some other agent). From the interaction digraph acyclicity, we can deduce that agent j is not an ancestor of agent k (who controls n_{jx}). Hence, using the same line of reasoning as in *case b*, by Theorem 3.32, $Pr(n_{jx}^{t+1}|\vec{n}_y^t, ...) = Pr(n_{jx}^{t+1}|...)$, and therefore, $\Gamma(n_{jx})$ need not be conditioned on n_y.

2. To prove that property 2 holds, let us consider two cases:
   <u>case a</u>: $n_{jx}^t = true$. By Definition 4.21, $Pr(n_{jx}^{t+1} = true|n_{jx}^t = true) = 1$, yielding a deterministic transition that is independent of previous values $\vec{n}_{jx}^{t-1}$. Thus, $Pr(n_{jx}^{t+1}|n_{jx}^t = true, \vec{n}_{jx}^{t-1}) = Pr(n_{jx}^{t+1}|n_{jx}^t = true)$.
   <u>case b</u>: $n_{jx}^t = false$. By Definition 4.21, $\vec{n}_{jx}^{t-1} = \langle false, false, false, ..., false \rangle$,

indicating that when n^t_{jx} takes on value $false$, its history $\vec{n}^{t-1}_{jx}$ is fully determined. Thus, $Pr(n^{t+1}_{jx}|n^t_{jx} = false, \vec{n}^{t-1}_{jx}) = Pr(n^{t+1}_{jx}|n^t_{jx} = false)$.
Combining *case a* and *case b*, $Pr(n^{t+1}_{jx}|n^t_{jx}, \vec{n}^{t-1}_{jx}) = Pr(n^{t+1}_{jx}|n^t_{jx})$, and hence $\Gamma(n_{jx})$ need not be conditioned on $\vec{n}^{t-1}_{jx}$. Therefore, by Definition 4.14, $\Gamma(n_{jx})$ is state-dependent but not history-dependent with respect to n_{jx}.

□

Example 4.23 (continued). Returning to the problem shown in Figure 4.1, since there is only a single interaction, the interaction digraph is degenerately acyclic. By Theorem 4.24, rover 5's influence on nonlocal feature *site-C-prepared* (modeled by rover 6), which we will abbreviate $\Gamma(c)$, may be encoded with probability distribution $\Gamma(c) = Pr(c^{t+1}|c^t, t)$, making $\Gamma(c)$ state dependent with respect to nonlocal feature $c \equiv$*site-C-prepared* and unaffectable feature $t \equiv$*time*.

The significance of Theorem 4.24 is that, for a commonly-studied class of problems with event-driven interactions (Becker et al., 2004a; Marecki & Tambe, 2009; Mostafa & Lesser, 2009), when the interaction digraph topology contains no cycles, agents influence encodings need not be conditioned on event histories. Avoiding history means that the size of the influence encodings will, at worst, grow linearly with the time horizon. Furthermore, by property 1 in Theorem 4.24, for problems with event-driven interactions and acyclic interaction digraphs, influences need not encode joint distributions over nonlocal feature transitions. In this case, the size of the influence DBN grows linearly with the number of event-based interactions (as long as no cycles are created). In my empirical results in Section 4.6.2.4, I show these traits to yield significant reduction (compared to history-dependent event-driven influences) in the overall computation required by influence-based policy abstraction.

However, the presence of just one undirected cycle violates the conditions of Theorem 4.24, and necessitates dependence on history (as we see in Example 4.25).

Example 4.25. Consider a slight variation of Example 4.23 in which there is additional interaction involving rover 5 preparing site D for rover 6. In this case, there are two event-based nonlocal features (*site-C-prepared* and *site-D-prepared*), and two corresponding edges in the interaction digraph both of which lead out of vertex 5 and into vertex 6. Consequently, the digraph contains an undirected cycle, thereby violating the conditions of Theorem 4.24. With this variation, the influence $\Gamma(c)$ becomes history-dependent and influence-dependent. The influence DBN must model the joint distribution $Pr(c^{t+1}, d^{t+1} | \vec{c}^{\,t}, \vec{d}^{\,t})$. Intuitively, since they are controlled by the same agent, the transitions of *site-C-prepared* and *site-D-prepared* are no longer independent. For instance, rover 5 cannot finish preparing both sites at the same time. Similarly, the probability with which rover 5 finishes preparing site D at any given time depends upon how long ago rover 5 finished preparing site C.

Fortunately, for cyclic cases, agents can encode the histories of *event-driven* features compactly. By Definition 4.21, the transitions of an event-driven feature f are structured such that f can only change from $false$ to $true$ but never from $true$ to $false$. Thus, the complete history $\vec{f}^{\,t}$ may be captured by a single variable f^t_{hist} with domain $\{0, 1, ..., t, false\}$ whose value is set to the time index that f changed from $false$ to $true$, or set to $false$ if f has not changed from $false$ to $true$.

4.5 Influence Space

Theorems 4.18 and 4.24 describe the size of the influence encoding, but they say nothing about the number of possible influence assignments. Each combination of local policies corresponds to a summarizing *influence DBN* whose probability values have been assigned accordingly. The **influence space** is the domain of feasible assignments to the probability values encoded by the influence DBN, where each feasible assignment is the result of at least one combination of local policies. I refer to a feasible assignment to the influence DBN as an **influence point** in the influence space.

An important hypothesis of this dissertation is that, in formulating optimal joint policies, agents can gain potentially significant computational advantages by searching through the influence space instead of searching through the joint policy space directly. The intuition is that, although every *influence point* maps to at least one joint policy, there may be many joint policies that all map to the same influence point.

Example 4.26. Returning to the example problem shown in Figure 4.1, rover 5 has several sites it can visit, each with uncertain durations. In general, different policies that it adopts may achieve different interaction probabilities. However, due to the constraints in Figure 4.1, many of rover 5's policies will map to the same influence point. For instance, any two policies that differ only in the decisions made after time 3 will yield the same assignment to $\Gamma^{6}_{\pi_5}(\textit{site-C-prepared}) = Pr(\textit{site-C-prepared} = \textit{true} | \textit{time} = 4)$. For this example, the influence space is strictly smaller than the policy space.

By considering only the feasible influence values, agents avoid redundant joint reasoning about the local policies with identical influences. Figure 4.7 illustrates the potential reduction from influence space to policy space.

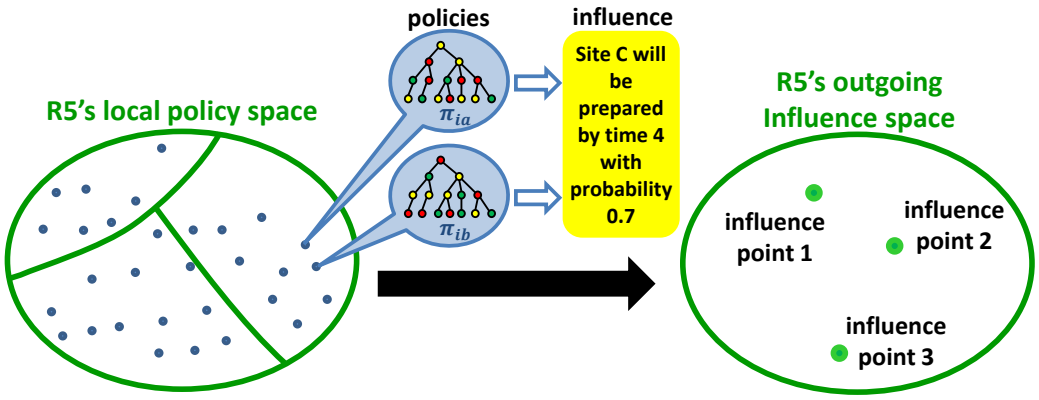


Figure 4.7: An agent's local policy space and resultant influence space.

In relation to the *weak coupling* theory developed in Section 3.5.2, influence-based abstraction can be viewed as framework for partitioning a TD-POMDP agent's policy space into *impact-equivalence classes* (Def. 3.45). For any two policies $\{\pi_i^x, \pi_i^y\}$ whose influences are identical ($\Gamma^j_{\pi_i^x} = \Gamma^j_{\pi_i^y}$), they necessarily provoke the same best-response from agent j. By Definition 3.44, π_i^x and π_i^y are *impact equivalent* and thus may be grouped into the same *impact equivalence class*. The converse is not true, however. Two local policies of agent i that result in the same best response from agent j may not map to the same influence point. As such, influence abstraction will not necessarily yield the most coarse-grained partitioning of the policy space.

Definition 3.49 in Section 3.5.2 related an agent's local policy space size to the

number of partitions achieved by an impact-equivalent partitioning scheme, calling the maximal ratio of these two quantities the **degree of influence**. For influence-based policy abstraction, the number of partitions is equal to the size of the influence space. Hence, the *degree of influence* afforded by influence-based policy abstraction is the influence space size divided by the policy space size.

4.6 Empirical Analysis of Influence Space Size

In the preceding section, I have provided intuition and anecdotal evidence to support my claim that influence-based policy abstraction framework can be employed to reduce the size of the search space wherein TD-POMDP agents seek to optimize their joint behavior. I have also contended that, beyond toy examples, there is a large space of TD-POMDP problems for which agents have far fewer unique influences than they do policies. I now defend this claim with a rigorous empirical analysis.

Further, I investigate the circumstances under which influence-based abstraction yields the greatest reduction. The theoretical treatment of *weak coupling* presented in Section 3.5.2 has proven that the worst-case complexity of computing optimal TD-POMDP policies is dependent upon the *degree of influence*. However, the theoretical results do not say anything about what problems might have a low degree of influence, nor do they provide a method for determining a problem's degree of influence before solving it. By itself, the theory cannot be applied practically. The empirical results that I present here pick up where the theory has left off, striving to expose identifiable attributes that determine a problem's degree of influence (in the context of influence-based policy abstraction), and thereby illuminating a part of the TD-POMDP space that is truly weakly coupled (along the *degree of influence* dimension).

My high-level strategy for performing this empirical analysis is as follows. Using the testbed detailed below, I generate a large sampling of random problems. The space of random problems is parameterized by a variety of attributes (detailed in Section 4.6.1), each of which I connect to identifiable aspects of the TD-POMDP problem description. According to this parameterization, I sample the problem space evenly across all parameter settings. For a given problem, I focus on an individual agent's influences in isolation from the rest of the team, directly measuring the size of the agent's influence space and the size of its local policy space.

By varying each parameter and observing its effect on the influence space size and the policy space size, I am able to characterize the relationships between a problem's identifiable attributes and its (less discernible) degree of influence. The results in

Section 4.6.2, which are the culmination of an iterative process of parameter definition and parameter testing, highlight those attributes that appear to be the strongest empirical predictors of a problem's influence-space size and its degree of influence. In Chapter 6, I empirically evaluate the overall computation performed by an optimal TD-POMDP algorithm that employs influence-based abstraction.

4.6.1 Experimental Setup

I perform my analysis on a testbed of task-based problems specified using a simplified version of the TÆMS modeling language (Decker, 1996). Problems of this flavor frequently arise in the Dec-POMDP literature (Becker et al., 2004a; Marecki & Tambe, 2009; Mostafa & Lesser, 2009; Musliner et al., 2006). Moreover, the multiagent planning community in general has demonstrated an interest in problems formulated using TÆMS, due to its domain-independent, naturally-distributed specification of interdependent agent activities with uncertain outcomes, its quantitative representation of team goals, and its emphasis on structured agent interactions (Atlas, 2009; Horling et al., 2006; Lesser et al., 2004; Smith et al., 2007; Wagner et al., 2003; Wu & Durfee, 2007; Xuan & Lesser, 1999). Though others have performed experiments on various hand-coded TÆMS problems, no universally-accepted problem suite has emerged. As such, I have created my own TÆMS-based testbed so as to systematically generate problems with a desired set of controlled parameters. I describe these parameters along with the details of my problem generator in Section 4.6.1.3, following a description of my problem specification in Section 4.6.1.1 and the corresponding models of influence in Section 4.6.1.2. In Section 4.6.1.4, I describe my method of evaluating each problem's degree of influence.

4.6.1.1 Task-based Problem Specification

The problem representation I am about to describe is the same as that introduced in Example 3.1 and used in several other examples in Chapters 3 and 4. For the sake of reproducibility of my experiments, I now provide a more detailed description of the task-based problem specification.

Each problem contains n agents, where each agent i has a set of tasks, $\mathbb{T}_i = \{task_{i1}, ..., task_{i||\mathbb{T}_i||}\}$, that it may execute with *outcomes* $\mathbb{D}_i = \{d_{i1}, ..., d_{i||\mathbb{T}_i||}\}$ and *window* constraints $\mathbb{W}_i = \{w_{i1}, ..., w_{i||\mathbb{T}_i||}\}$. For $task_{ix}$, $d_{ix} = \{\langle dur_{ixk}, qual_{ixk}, prob_{ixk}\rangle\}$ specifies the probabilities of outcome *durations* and associated *qualities*. Each window constraint is a pair $w_{ix} = \langle est_{ix}, lft_{ix}\rangle$ denoting the *earliest start time* and *latest finish*

time of the task. An agent can only perform one of its tasks at a time, with idling allowed between task executions. As such, the TD-POMDP's local action set A_i contains an action for each $task_{ix} \in \mathbb{T}_i$ and a *NOOP* action that causes the agent to idle for one time step. Once an agent starts a task, it cannot interrupt the task, so its only available action is to continue the task until the task ends. All task executions must occur between time steps 0 and a finite horizon T. However, an agent cannot start a task before the task's *earliest start time* or after its *latest finish time*. If a task does not achieve one of its prescribed outcomes by its *latest finish time*, the task instead terminates immediately in a failure outcome (with quality 0).

There are also task interrelationships called *effects*, each of the form form $e_{ix,jy} = \langle task_{ix}, task_{jy}, d'_{iy} \rangle$, indicating that the completion of $task_{ix}$ with positive outcome quality changes the subsequent outcome distribution of $task_{jy}$ from d_{iy} to d'_{iy} (as long as agent j performs $task_{jy}$ after $task_{ix}$ finishes).[8] Examples of *effects* appear in Figure 3.8 as arrows connecting tasks. Here, Task A effects a change in the outcome of Task D, *enabling* a nonzero quality outcome to be attained with positive probability. The same sort of effect links Task C and Task F, as well as Task B and Task C.

Agent i may have *local effects* $\mathbb{L}_i = \{..., e_{ix,iy}, ...\}$ that link its own tasks. Additionally, agent i may have *incoming nonlocal effects* $\mathbb{N}_i^{\text{in}} = \{..., e_{jx,iy}, ...\}$ and *outgoing nonlocal effects* $\mathbb{N}_i^{\text{out}} = \{..., e_{ix,jy}, ...\}$, indicating interactions with other agents. Collectively, the set of all of the team's nonlocal effects is denoted $\mathbb{N} = \bigcup_{\forall i} \left[\mathbb{N}_i^{\text{in}} \cup \mathbb{N}_i^{\text{out}}\right]$. In the presence of nonlocal effects, each agent's tasks $\mathbb{T}_i$ are categorized into two disjoint sets $\mathbb{T}_i = \mathbb{T}_i^{\text{local}} \cup \mathbb{T}_i^{\text{nle}}$: *local* tasks $\mathbb{T}_i^{\text{local}}$ and *nonlocally-affecting* tasks $\mathbb{T}_i^{\text{nle}}$, such that $task_{ix} \in \mathbb{T}_i^{\text{nle}}$ if and only if $task_{ix}$ is referenced in $\mathbb{N}_i^{\text{out}}$ (which indicates that it affects another agent's task).

The TD-POMDP local state S_i includes a *task status* features for each $task_{ix} \in \mathbb{T}_i$ with domain {*not-started*, *started-at-time-t* (for each $t < T$), *completed-with-positive-quality*, *failed*}. The mutually-modeled feature set $\bar{m}_i$ consists of *time* (the current time index) as well as the statuses of agent i's *nonlocal effects* $\mathbb{N}_i^{\text{in}} \cup \mathbb{N}_i^{\text{out}}$, each with domain {*true*, *false*} indicating whether or not the respective effecting task has completed.[9] As such, each nonlocal feature $n_{ix,jy}$ is a boolean variable indicating the status of an incoming nonlocal effect $e_{ix,jy} \in \mathbb{N}_i^{\text{in}}$. The TD-POMDP is *locally fully observable*, such

[8] The definition of *effect* here deviates slightly from that of the TÆMS language (Decker, 1996), but suffices to model several common TÆMS effects such as task *enablement* and *disablement* and special cases of *facilitation* and *hindering*.

[9] Additional information about an *effect* need not be encoded in the TD-POMDP state because the effect only depends upon whether or not the affecting task has completed with positive quality. How long ago the task completed or which outcome it attained (as long as the outcome had positive quality) is of no consequence to the affected task's dynamics.

that agents observe the features in their local states directly (including the statuses of their incoming nonlocal effects). An agent's local rewards are zero for all state-action pairs prior to the horizon T, and otherwise equal to the sum of its completed task qualities. The objective is to plan coordinated policies for agents' task executions that maximize the summation of qualities attained from all agents' completed tasks. A detailed example of the transition and reward structure for these problems appears in Figure 2.3 (in Chapter 2), which shows a single-agent MDP that has been constructed as described here.

Although my influence-based abstraction methodology (and the solution methods that I present in subsequent chapters) are fully general to the TD-POMDP problem class, my task-based problem specification just described has several limitations that restrict consideration in my empirical work to a subset of TD-POMDP problems. For instance, each task has just one contiguous window and agents receive full observations of their individual task statuses. These are both restrictions that are inherent to the TÆMS modeling language (Decker, 1996). Additionally, agents' interactions consist solely of event-driven nonlocal effects relating to task completion events, and the value of a joint policy is assumed to be the *summation* of completed task quality values. All of these restrictions together are common to other empirical studies of related Dec-POMDP subclasses (Becker et al., 2004a; Beynier & Mouaddib, 2005; Marecki & Tambe, 2009; Mostafa & Lesser, 2009).

4.6.1.2 Anatomy of an Influencing Agent

In this particular set of experiments, I study individual agents' outgoing influences, directly comparing local policy space size with outgoing influence space size. As such, each problem consists of a single agent i that is not affected by others[10], but who models zero or more locally-controlled nonlocally-affecting features, each of which can be thought of as affecting some other phantom agent, and hence each of which agent i considers to be *mutually-modeled* (Def. 3.13). Figure 4.8 shows a snapshot of agent i as the root vertex in an interaction digraph, where each outgoing edge represents a nonlocally-affecting feature. Each such feature n_x is a boolean feature that, when *true*, denotes the successful completion of agent i's nonlocally-affecting task $task_{ix} \in \mathbb{T}_i^{\text{nle}}$.

[10]For simplicity, I limit consideration in this analysis to an agent that may influence its peers but is not influenced by its peers. Given an acyclic interaction digraph, this limitation does not restrict the generality of my results: an uninfluenced agents' local decision model is equivalent to that of an agent whose incoming influence has been fixed. In the case of a cycle, where agent i's outgoing influence settings affect others that in turn influence i, I have not yet determined whether or not the size of the overall influence space will be affected.

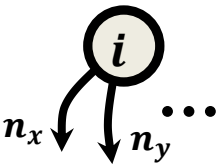


Figure 4.8: A digraph vertex representing an influencing agent.

Aside from agent i's nonlocally-affecting task features, the only other feature that is *mutually-modeled* is $time \in \{0, 1, ..., T\}$. Agent i models the corresponding influence on each nonlocal feature as *state-dependent* with respect to feature *time*.[11] In this set of experiments, we consider two different variations: (a) one modeling each influence $\Gamma_{\pi_i}(n_x)$ as state-dependent with respect to n_x, and (b) another modeling all of the influences with a single joint distribution $\Gamma_{\pi_i}(n_x, n_y, ...)$ that is history-dependent with respect to all nonlocal features $\{n_x, n_y, ...\}$. Figure 4.9 illustrates each variation.

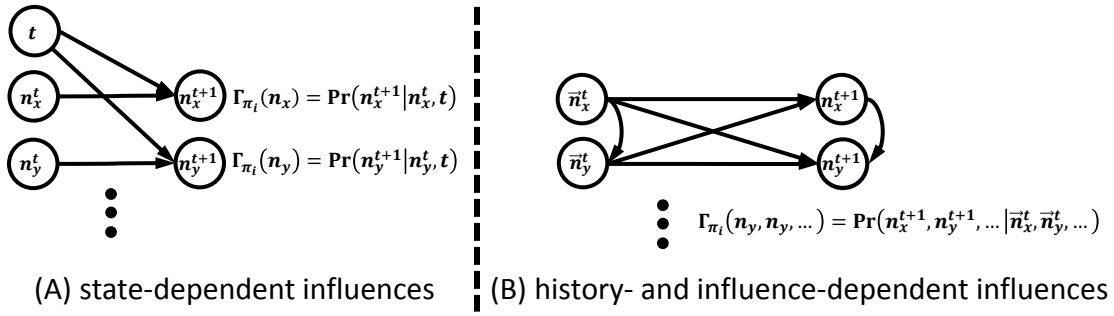


Figure 4.9: Two variations of the agent i's influences.

The analysis of these two variations was motivated by my theoretical treatment of *event-driven* interactions in Section 4.4. Case (a), in which the influences are represented as separate state-dependent distributions, corresponds to a problem whose interaction digraph contains no cycles. Here, the phantom agents that i is assumed to be influencing are necessarily unique, meaning each nonlocal feature $\{n_x\}$ affects a different agent. Case (b) corresponds to problems wherein all of the nonlocal features $\{n_x, n_y, ...\}$ correspond to edges situated in an undirected cycle in the interaction digraph. For instance, all of the nonlocal features might affect the same phantom agent.

[11] The influence $\Gamma_{\pi_i}(n_x)$ is state-dependent (and not history-dependent) with respect *time* because the feature *time* changes deterministically and predictably.

4.6.1.3 Problem Attributes and Testbed Parameters

I have implemented a random problem generator to systematically create random problems of the form described in Section 4.6.1.1. For each problem, I generate a local TD-POMDP model $\mathcal{M}_i$ (Def. 3.16) for an influencing agent i, who is described in Section 4.6.1.2. My testbed is comprised of sets of random problems, wherein each set is seeded with a particular setting of parameters. The parameters, detailed below, serve as control knobs to adjust various high-level attributes of the TD-POMDP problems that I generate. I now describe each high-level attribute and the associated testbed parameter(s).

Number of Decision Steps. A parameter T controls the global time horizon. All task executions must occur in the interval $[0, T]$. As such, the TD-POMDP is *unrolled* (referring to the process of creating states, actions, and transitions indexed with $time \in \{0, \ldots, T\}$) such that there are T decision steps $\{0, 1, \ldots, T-1\}$. Likewise, observation histories are of maximal length T.

Branching Due to Decisions. From each state s^t (with $time = t$), there is at least one branch[12] for each available action by which the agent i transitions into a state s^{t+1} (with $time = t+1$). For task-based TD-POMDP problems, the branching due to agents' decisions is controlled by a parameter *tasks per agent* $= \|\mathbb{T}_i\|, \forall i$. Because there are only branches for available actions, the branching factor is also controlled by a parameter *local window size*$\in (0, 1)$, by which the each task's earliest start time and latest finish time are set. The length of every task's window is $\lfloor$*local windows size*$\cdot (T-1) \rfloor + 1$. Using this quantity, each task's earliest start time is selected such that the task's window is placed uniformly randomly within the interval $[0, T]$.

Branching Due to Uncertainty. The branching factor is also dependent on the uncertainty inherent in the tasks that agents perform. A parameter *uncertainty* $\in [0, 1]$ controls the number of outcomes of each task. All outcomes of a given task have equal quality[13] (selected uniformly randomly $\in \{1, ..., 10\}$) but different durations. Each task's duration distribution is set to a randomly-assigned probability mass function,

[12] Note that for task-based problems, the TD-POMDP state space is a directed acyclic graph and *not* a tree.

[13] Since TD-POMDP agents' transition influences model feature transition probabilities and *not* rewards, task outcome qualities will not affect the size of the influence space nor the degree of influence.

wherein $\lfloor uncertainty \cdot (local\ windows\ size - 1) \rfloor + 1$ different durations are selected at random in the interval $(1, local\ windows\ size)$, each of which are assigned a random probability such that their probabilities together sum to 1.

I have generated 50 random problems[14] for each setting of the above parameters, whose domains are given in Table 4.1. As specified thus far, these problems contain no agent interaction, and will be henceforth referred to as *baseline* problems, and the above parameters as *baseline* parameters.

Next, I consider several additional parameters used to exact control over the agent i's interactions. For this second set of parameters, which I refer to as *interaction parameters*, rather than generating problems at random for each parameter setting, I use the 50 problems per baseline parameter setting, each of which contains no agent interactions, as baseline problems. For each baseline problem, I generated modified versions by systematically varying each interaction parameter. In the end, I have a set of 50 test problems per setting of baseline parameters per setting of interaction parameters. The domains of each parameter are summarized in Table 4.1. I describe the interaction parameters below, indexed by the high-level TD-POMDP attributes they are intended to embody.

Number of Nonlocal Features. The number of nonlocal features is controlled by a parameter $NLATs = \|\mathbb{N}_i^{\text{out}}\|$, which is the number of agent i's nonlocally-affecting tasks, constrained to be less than or equal to *tasks per agent*.

State-dependent Influences vs. History and Influence-dependent Influences. As described in Section 4.6.1.2, I analyze two different variations of influence models. In the first variation, i's influence on each nonlocal feature is modeled using a separate state-dependent distribution. In the second variation, i models a single joint

[14] I selected the number 50 based on the following observations. The random variations in example problems generated by my testbed differ most significantly in the earliest start time of each task, which can take on $T - \lfloor localWindowSize(T-1) \rfloor$ different values (each uniformly at random), and in the positioning of random task durations, which can take on $\binom{\lfloor (localwindowSize*(T-1)\rfloor+1}{\lfloor uncertainty \lfloor (localwindowSize*(T-1)\rfloor\rfloor+1}$ different values. The number of combinations of these two types of random variations, across all tasks, is thus $tasksPerAgent\,(T - \lfloor localWindowSize(T-1)\rfloor)\binom{\lfloor (localwindowSize*(T-1)\rfloor+1}{\lfloor uncertainty \lfloor (localwindowSize*(T-1)\rfloor\rfloor+1}$, a term which is maximized by the baseline parameter setting: $\langle T = 5, localwindowSize = 1.0, uncertainty = 0.5,\ tasksPerAgent = 3\rangle$. For this particular parameter setting I generated an extra 100 random problems for each setting of the interaction parameters. Using this additional test set, I performed each comparison presented Section 4.6.2 and compared the results with the same comparison using a set of 50 problems. In all cases, the trends observed with the 100-problem test set were qualitatively identical to those observed with the 50-problem test set. From this, I concluded that, for the other parameters settings whose problem variance was theoretically smaller, that 50 random problem was adequate to sample the space for all other parameter settings.

distribution that is history-dependent with respect to all nonlocal features. Parameter *influence type*$\in \{state, history\}$ controls which variation is used.

Window of Nonlocal Feature Manipulation. For problems with a single nonlocal feature, I introduce two parameters that constrain when the nonlocal feature is allowed to be manipulated. The size of the window of agent i's nonlocally-affecting task is set to *NLATWindow*, and the beginning of the window is set to *NLAT_est*. In essence, $NLATWindow$ and $NLAT\_est$ control the timing of agent i's interactions. Empirical results that I present in the next section demonstrate that both of these features have a significant impact on the size of the influence space.

Problem Attributes	**Testbed Parameter**	**Domain**
— *Baseline Parameters* —		
Number of Decision Step	T	$\{1, 2, 3, 4, 5\}$
Branching Due To Decisions	*tasks per agent*	$\{1, 2, 3\}$
	local window size	$\{0.0, 0.5, 1.0\}$
Branching Due To Uncertainty	*uncertainty*	$\{0.0, 0.5, 1.0\}$
— *Interaction Parameters* —		
Number of Nonlocal Features	*NLATs*	$\{1, \ldots, tpa\}$
State-Dependent Influences vs. History-Dependent Influence-Dependent Influences	*influence type*	$\{state, history\}$
Window of Nonlocal Feature Manipulation[15]	*NLATWindow*	$\{0, 1, \ldots, T\}$
	NLAT_est	$\{0, \ldots, T-1\}$

Table 4.1: Testbed parameterization.

The parameters and their respective domain values shown in Table 4.1 have been chosen so as to generate spaces of problems that are sufficiently rich to include a variety of different scenarios, and to demonstrate general trends and relationships between a problem's high-level characteristics and its degree of influence, yet define a space that is small enough to be explored systematically and thoroughly. Strictly speaking, my results are only directly applicable to problems in my testbed. The degree to which other TD-POMDP problems exhibit the same quantitative values of influence space sizes and degree of influence will depend upon their similarity to those from my testbed. However, I have no reason to believe that the qualitative trends observed here will not generalize beyond the space considered here.

[15]The domains of *NLATWindow* and *NLAT_est* are systematically explored only in the case of a single nonlocally-affecting task ($NLAT$=1).

4.6.1.4 Evaluation Scheme

To evaluate *influence space size*, I employ an algorithm presented in the next chapter (Section 5.6) that explores the influence space exhaustively, counting each unique setting of probabilities in the *influence DBN* that is achieved by a deterministic policy of agent i. For each problem, in addition to recording the number of influences in agent i's influence space, I also record the number of deterministic policies in agent i's policy space, computed as the product of the number of available actions in every state of the agent i's best response model (detailed in Section 4.2). For any given problem, I calculate the *degree of influence* by dividing the *influence space size* by the *policy space size*.

4.6.2 Results

Using the testbed described in Section 4.6.1, I have performed a series of experiments that illuminate the relationships between the degree of influence and the problem attributes described in Section 4.6.1.3. I now present the results, organized by problem attribute (each of which was introduced in Section 4.6.1.3). I provide a summary of all findings at the end of this section.

Note that each of the comparisons described below has been performed across all combinations of parameter settings. For each result that I present, instead of overwhelming the reader with page after page of plots for each and every setting, I select just a few cases (which I label {A, B, C, ... } followed by the corresponding parameter setting) whose qualitative trends are representative of the entire space of parameter settings tested.[16]

4.6.2.1 Number of Decision Steps

The number of agent i's decision steps, specified by the time horizon T, has a significant effect on the size of the problem in general. Since *time* is necessarily a feature of the TD-POMDP state, the state space grows (in general exponentially) with each additional decision step. In turn, each additional state with more than one available action constitutes an additional decision for agent i to make, causing an increase in the policy space (which is in general doubly exponential in the time horizon).

[16]Rest assured that the plots for all other settings have been examined, and are omitted here simply because they do not provide any additional information about the high-level, qualitative trends that I describe.

I posit that the influence space size should also increase with the time horizon. In general, a longer horizon entails more times during which agent i's interactions can take place. In particular, in the problems that I generate, the windows of agents tasks are specified relative to the time horizon T, such that the number of different times that agent i is allowed to start a nonlocally affecting task increases proportionally with T. The greater the number of possible start times, the greater the number of possible finish times, and hence the more unique influences.

Figure 4.10 supports my hypothesis. Here, local state space size, local policy space size, and influence space size are plotted as a function of T, each for three different settings of the baseline parameters[17] (from Table 4.1). Out of all possible combinations of parameters, the three settings, labeled A, B, and C, were chosen as representative snapshots of three different gradations of problem difficulty (as measured by state space size and policy space size). Moreover, the trends portrayed by these plots are representative of the trends observed across all of the other possible settings.

In each plot, T is varied from 1 to 5 along the x-axis, and the y axes are given a logarithmic scale. As expected, a near-exponential increase in state space size is observed along with an exponential (or asymptotically greater) increase in policy space size for all parameter settings. Similarly, the influence space size increases exponentially with T. Although the steepness of the exponential increase depends upon the particular parameter setting, the steepness of increase of policy space size generally[18] exceeds that of the influence space size. The result of this difference in growth is that the *degree of influence* decreases as the number of decision steps grows. The degree of influence for settings A, B, and C, is plotted, again on a logarithmic scale, in Figure 4.11.

4.6.2.2 Branching Due To Decisions

Next, I examine the impact of branching in the local state space caused by decisions (i.e. action choices) that agent i faces . For the task-based problem in my testbed, branching from decisions is controlled by two parameters: the number of *tasks per agent* and the *local window size*. Each of these two parameter, when increased, yields a consequent increase in the number of available actions (averaged across the entire

[17]For each problem, one or two (prescribed by "NLATs=1" or "NLATs=2") of agent i's tasks have been selected at random and turned into nonlocally-affecting tasks, and modeled as state-dependent or history-dependent (as prescribed by "influenceType=state" or "influenceType=history") influences.

[18]This trend was observed across all parameter settings with the exception of one degenerate case, involving a single task per agent with window size of 1, in which both the influence space size and policy space size remained constant. In this case, regardless of the time horizon or the placement of the task's window, agent i only ever has 2 possible policies and 2 possible influences.

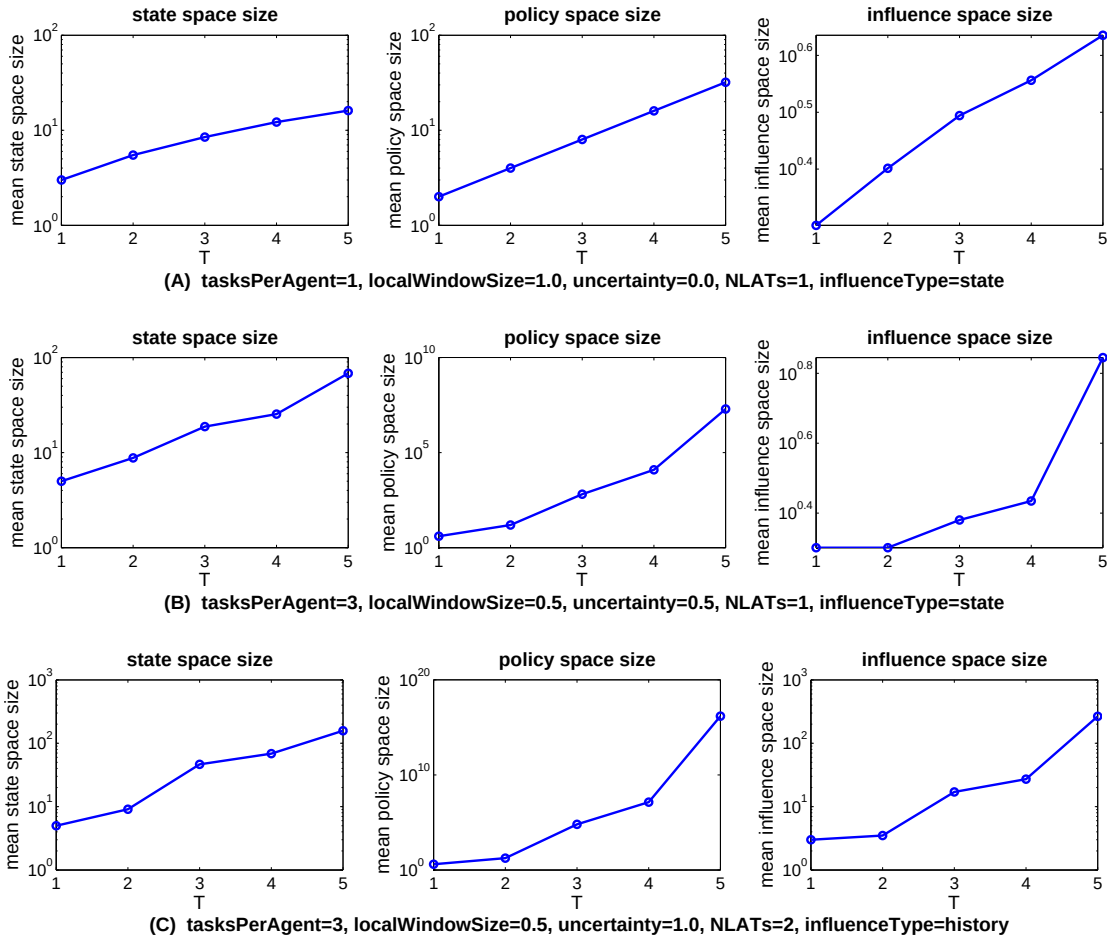


Figure 4.10: State, policy, and influence space sizes as a function of *time horizon* T.

state space). In any given state, an agent has at most *tasks per agent*+1 available actions (where the additional action causes the agent to *idle*). However, not all of these actions will be available in every state. The *local window size* controls the proportion of decision steps during which each action can be taken.

For this experiment, I systematically vary each of these two parameters and examine the effect on *policy space size* and *influence space size* as before. I also measure the *average branching factor* (across all nonterminal states) of the agents' local decision model that results from each setting of the two parameters, so as to solidify the connection between testbed parameters and branching factor (which is a more general attribute that is easily computed for any TD-POMDP problem, task-based or otherwise).

The results are shown in Figure 4.12, where *local window size* is varied along the x-axis and the three values of *tasks per agent* appear as lines superimposed on each

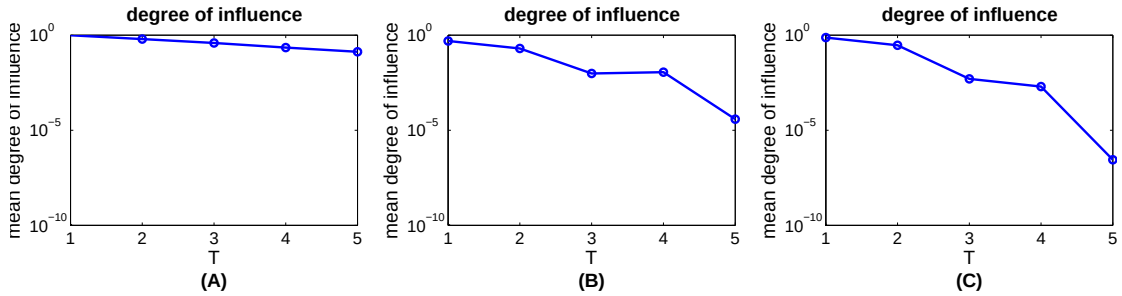


Figure 4.11: *Degree of influence* as a function of *time horizon* T.

plot. Again, out of all the combinations of parameter settings, I present three settings (A, B, and C) whose qualitative trends are representative of those observed in all of the remaining settings.

Figure 4.12 generally confirms that more tasks and wider task windows yield a larger branching factor. However, the increase in branching factor due to local window size is extremely small for setting A. This is because the *uncertainty* parameter is set such that each task has a single duration that is uniformly randomly selected from interval $(0, \lfloor \textit{local windows size} \cdot (T-1) \rfloor + 1)$. Effectively, as the local window size increases, tasks tend to take deterministically longer to complete, causing a larger portion of states to have just a single available *continue* action, and thereby counteracting the rise in branching factor from increased local window size.

For all settings, we observe that the policy space size increases both with *local window size* and with *tasks per agent*, and that the influence space size increases with the local window size. In the majority of parameter settings, we observe the same qualitative relationship between influence space size and *tasks per agent*, though this trend is faint for setting B and indiscernible for setting A. The reason for this is that, due to the combination of small values of *uncertainty* and small values of *local window size*, the number of outcomes of each task is limited to just 1, for setting A and all of the points in setting B except for *local window size*=1.0 (wherein the number of task outcomes is 2). With just a single duration per task, problems become entirely deterministic, as do agent i's influences. That is, each influence either conveys (1) that the nonlocally-affecting task will complete with certainty at a particular time or (2) that the nonlocally-affecting task will never complete. As such, the number of influences simply relates to the number of different times agent i is allowed to start the nonlocally affecting task regardless of the other tasks that the agent might execute.

In all three cases, we observe a decrease in the degree of influence as the branching factor increases (shown in Figure 4.13). The decrease in degree of influence is minimal

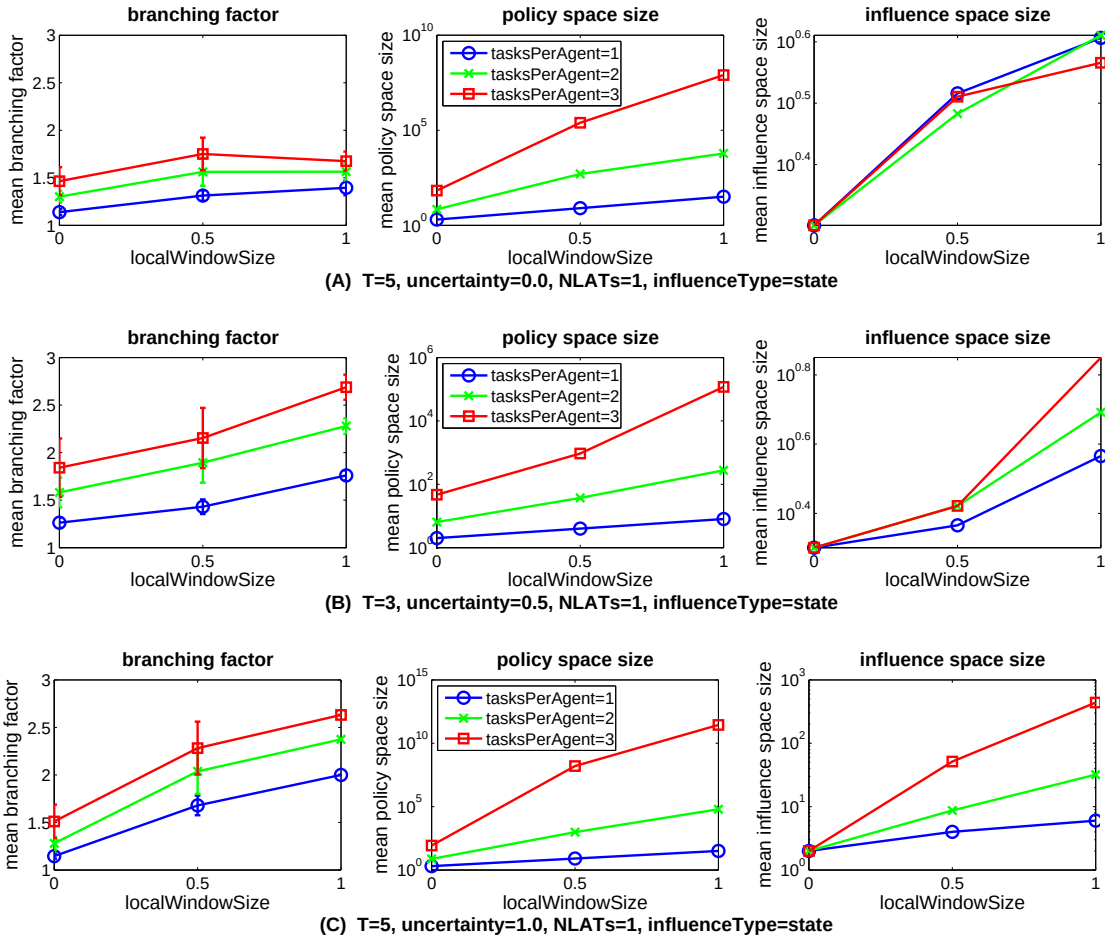


Figure 4.12: Branching factor, policy space size, and influence space size.

when there is only a single task per agent because, in this case, all of the agents decisions involve deciding whether not to start the single nonlocally-affecting task. For this scenario, one might expect the influence space size to be equal to the policy space size. This expectation is valid when the *local window size* is 0 (meaning that the length of the window is 1 time unit) because there are just 2 influences and 2 policies. However, as the nonlocally-affecting task window grows, a growing number of policies dictate that agent i start executing the task too late for it to succeed before its latest end time, and each of these policies all map to the same influence, thereby effecting a decrease in the degree of influence.

4.6.2.3 Branching Due To Uncertainty

Another component that affects the branching factor of agent i's state space is the uncertainty in the outcomes of its actions. Given more uncertainty, there is a wider

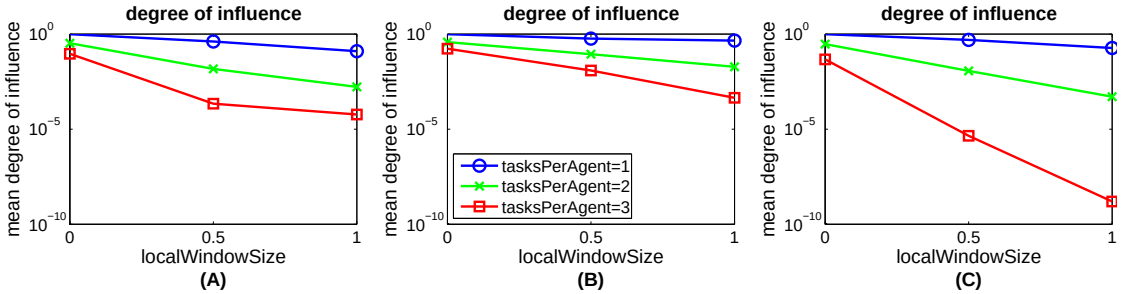


Figure 4.13: *Degree of influence* vs. *tasks per agent* and *local window size*.

array of future states reachable from each action, and hence a larger branching factor. Again, I hypothesize that the increase in branching factor will yield a general increase in the size of the policy space. Although there are no more actions available in each state, I expect that there will be a greater number of states, thereby yielding a greater number of policy decisions. Uncertainty should also have a significant effect on the size of the influence space. Since influences encode probabilities of nonlocal feature transition outcomes, and more uncertainty yields a greater number of probabilistic outcomes, increasing the uncertainty can only increase the number of feasible influence points.

I test these hypotheses by varying a testbed parameter, *uncertainty*, which controls the number of outcomes of each of agent i's tasks, such that the number of outcomes per task is set to $uncertainty \cdot \lfloor local\ windows\ size - 1 \rfloor + 1$ (as detailed in Section 4.6.1.3). The results are shown in Figures 4.14 and 4.15, which plot the branching factor, policy space size, influence space size, and *degree of influence* as a function of *uncertainty* for three different parameter settings. As before, the trends shown for parameters settings A, B, and C are qualitatively characteristic of all remaining combinations of parameter settings (from Table 4.1) not shown.

As predicted, all cases exhibit an increase in both branching factor (plotted on a linear scale) and influence space size (plotted on a logarithmic scale) as *uncertainty* is varied from 0.0 to 1.0. In almost all cases (including those not shown), we also observe an increase in the policy space size. Moreover, the increase in the policy space size usually overwhelms that of the influence space size. This trend is clearly illustrated by case A, where both the policy space and influence space appear to grow exponentially but the policy space grows more steeply, yielding an overall decrease in the degree of influence (Figure 4.15A). In case B, we again observe an exponential increase in the influence space, but a less pronounced increase in policy space, and consequently

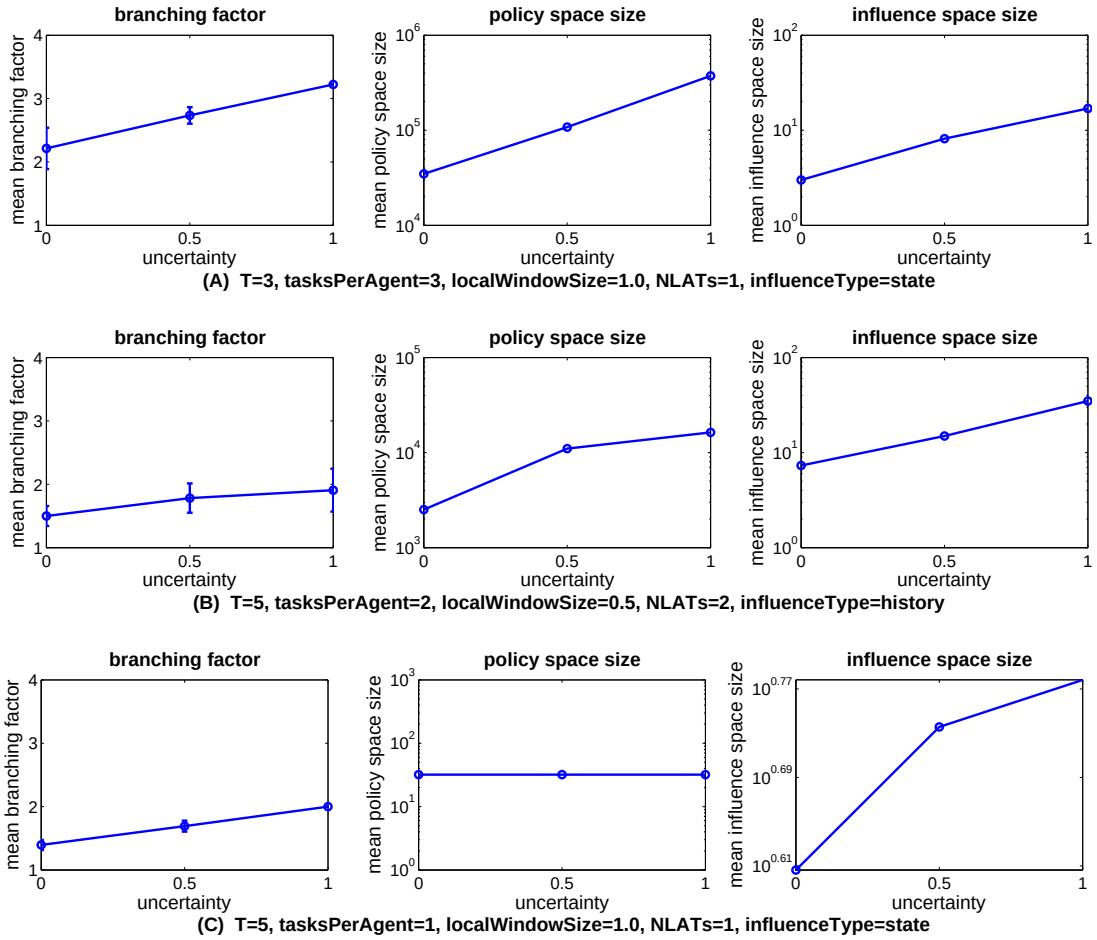


Figure 4.14: State, policy, and influence space sizes as a function of *uncertainty*.

only a very slight decrease in the degree of influence. This suggests that for smaller problems with fewer tasks (such as in case B), influence-based abstraction may have less computational benefit (yielding a smaller reduction in the search space even when there is a large amount of uncertainty).

Case C illustrates an extreme wherein there is just a single task. Here, since the only actions agent i has are to begin the task or to *idle*, the only choice the agent faces is whether or not to begin its task. Varying the outcomes of the lone task have no effect on the policy space whatsoever, and hence we observe a flat policy space size curve (in Figure 4.14C). In contrast, the influence space grows as the uncertainty is varied from its minimum to maximum value, yielding a slight increase in the degree of influence (Figure 4.15C). Note that, out of all parameter settings including those not shown, case C exhibited the largest growth in the degree of influence.

The results in the past three sections are promising. In general, Figures 4.10–4.15

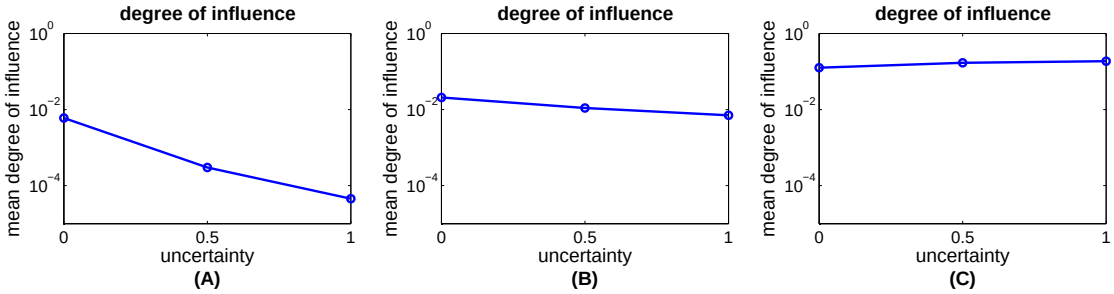


Figure 4.15: *Degree of influence* as a function of *uncertainty*.

show an increase in influence space size but a decrease in the degree of influence. This suggests that, by and large, as agents' local problems become more complex, the benefits of abstracting influences will be magnified. However, this empirical trend is conditioned on the agents' nonlocal effects remaining constant as the local problem size grows. In the next set of experiments, I explore what happens when the nonlocal effects are varied.

4.6.2.4 Number of Nonlocal Features and Influence Type

The number of nonlocal features (controlled by agent i and affecting other agents) has a direct effect on the size of agent i's influence encoding, whose parameters are the probabilistic transitions of agent i's nonlocal features. As described in Section 4.6.1.2, if each nonlocal feature is encoded with a separate state-dependent influence (which is denoted by parameter setting *influenceType=state*), the size of the influence encoding grows linearly with the number of nonlocal features. If, on the other hand, agent i models its influences as history-dependent and influence-dependent with respect to each other (*influenceType=history*), the size of i's influence encoding grows exponentially with the number of nonlocal features.

As the size of the encoding increases, the influence is capable of representing more details of i's policy. Further, by describing agent i's behavior with greater specificity, thereby allowing each influence point a refined scope of possible policies, one would expect that this richer encoding would also engender a larger number of influence points. In other words, a larger encoding should result in a larger influence space.[19]

[19]Although this assertion is intuitive, there do exist counterexamples for which the influence space size is the same for a larger, more informative encoding than it is for a smaller, less informative

As such, I offer the general hypotheses that (1) as the number of nonlocal features increases, the size of the influence space increases; and (2) when influences are history- and influence-dependent, the size of the influence space will be, on average, larger than when influences are state-dependent. I test these hypotheses by systematically generating variations of the baseline problems used in the previous sections (and described in Table 4.1), where in each variation, some number (specified by the value of parameter $NLATs$) of agent i's tasks are turned into nonlocally-affecting tasks. Further, for each problem, I explore the feasible space of influences for both the state-dependent encoding (denoted *influenceType=state*) and the history-dependent influence-dependent encoding (denoted *influenceType=history*), both of which are described in Section 4.6.1.2.

Figure 4.16 shows the results of varying the number of nonlocally-affecting tasks for three different settings of baseline parameters (A, B, and C). Across all settings (including those not shown), we observe an increase in the average influence space size as the number of nonlocally-affecting tasks increases. We also observe an increase in the average degree of influence for all settings due to the fact that agent i's policy space remains constant as $NLATs$ is varied (given that $tasksPerAgent$ is fixed). For state-dependent influence encodings (which are indicated by the black line with circular markers in Figure 4.16), the average increase in influence space size was near linear in all cases. The history-dependent influence encodings exhibited similar average influence space growth in some cases, and super-linear growth in other cases.

The three cases A, B, and C that I present here illustrate a dominant trend that I noticed across the space of baseline parameters settings. For cases with low uncertainty (e.g., case A), the influence space size for state-dependent influence encodings was always equal to that of the history-dependent influence encoding. Intuitively, the two influence space sizes must be the same for this case because $uncertainty = 0.0$ classifies problems as entirely deterministic.[20] When $uncertainty > 0$, and as the number of nonlocal features increases, the growth of the influence space size for history-dependent encodings overtakes that of state-dependent encodings on average. Moreover, as illustrated by cases B and C, as the uncertainty becomes larger, the gap between average influence space sizes widens. The last point in Figure 4.16C is missing because, for some of

encoding.

[20]When all effects are deterministic, any given policy will result in nonlocal features changing values at predictable times, and there is only one possible history. In this case, any influence point encoded with a history-dependent influence can be reduced to one encoded with a state-dependent influence, and thus cannot differentiate between any two policies that the state-dependent influence could not have differentiated between, and therefore cannot accommodate a larger space of feasible influences.

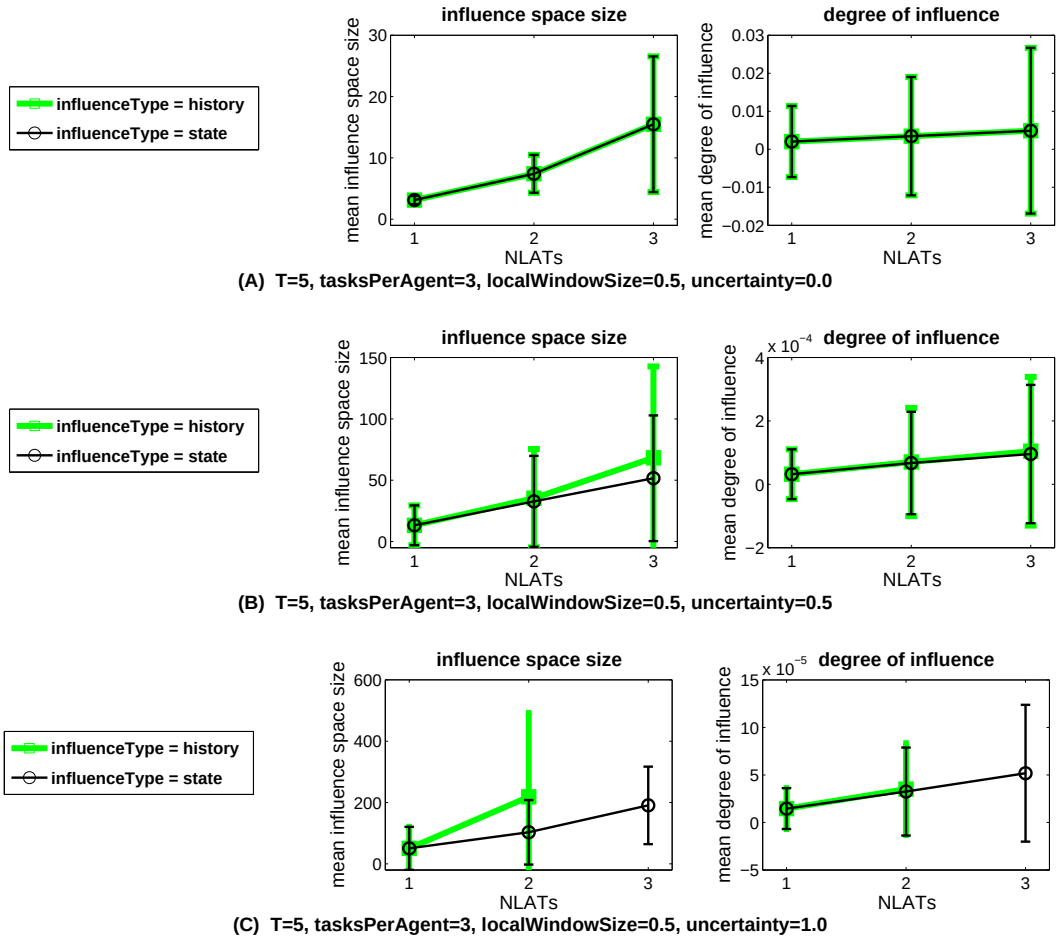


Figure 4.16: Varying the number of nonlocally-affecting tasks and influence type.

the problems with setting $\langle uncertainy = 1.0, NLATs = 3, influenceType = history \rangle$, the influence spaces were not able to be explored exhaustively within the 3 hours of computation time allotted to each problem. Based on those runs that did complete, the average influence space size for this missing data point is greater than 600.

For all cases, notice that the variance in both the influence space size and the degree of influence is extremely large. Figure 4.17 presents a more detailed view of influence space sizes for case B above. Here, a histogram shows the distribution of influence space sizes for 100 randomly-generated problems per value of $NLATs$, and for both the state-dependent influence encoding (top) and the history-dependent encoding (bottom). With histograms for the three values of $NLATs$ superimposed (in black, grey, and white), we observe that distributions take on a similar shape. For all values of $NLATs$ and $influenceType$, there is a large mass within the range of 1-50 influences and a tail leading outward. As $NLATs$ increases, the mass tends to

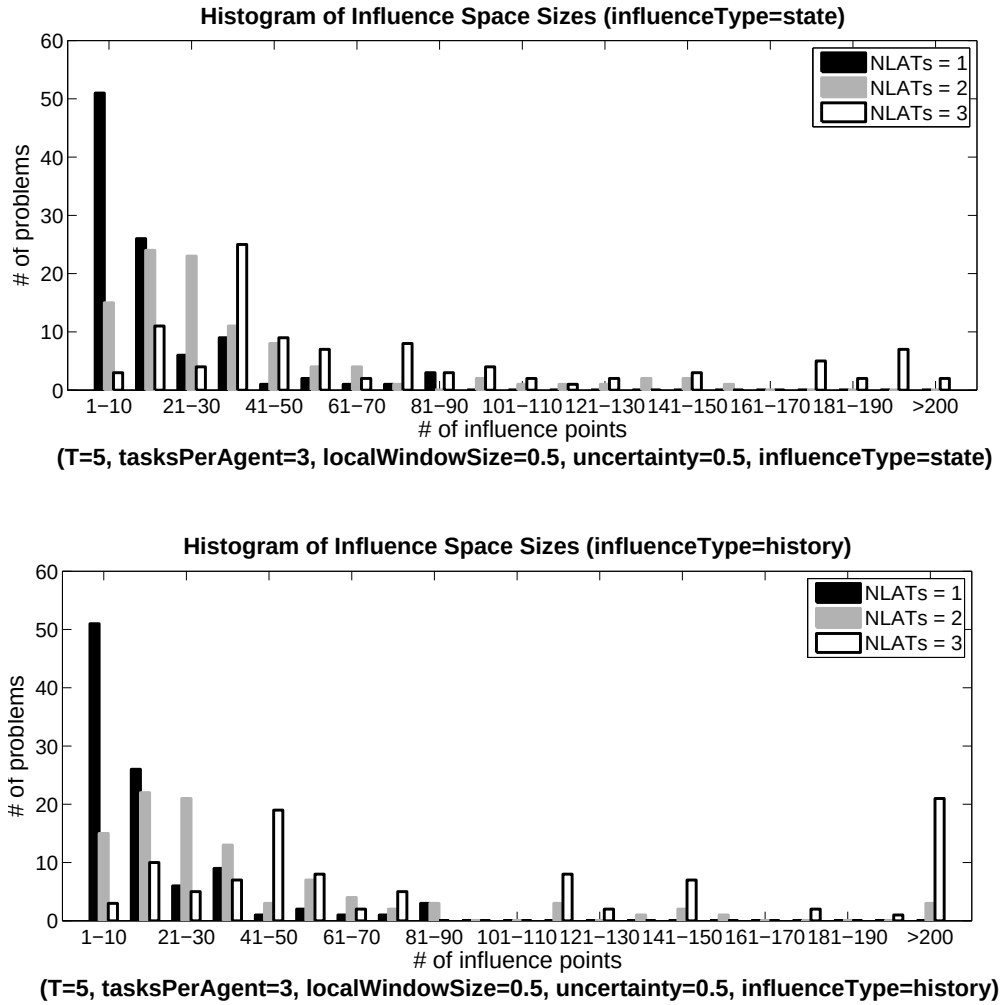


Figure 4.17: Distribution of influence space sizes for 100 problems (per setting).

become more spread out, and the tail of the distribution heavier. For this particular case, there is not a large difference between the state-dependent influence encoding and the history-dependent encoding except for $NLATs = 3$, where we observe that the distribution is spread well beyond 200 for the latter encoding.

One cause of the variance in influence space size is due to the variance in the complexities of the local models created by the random problem generator (caused by random window placement and random duration selection). This is evident from the large range of policy space sizes, which ranged from $5,184$ all the way to $382,205,952$ for the problems plotted in Figure 4.17. With this wide range of policy space sizes, it is not surprising that the influence space sizes varied from 2 to 613. Figure 4.18 shows a scatter plot of policy space sizes and corresponding influence space sizes for

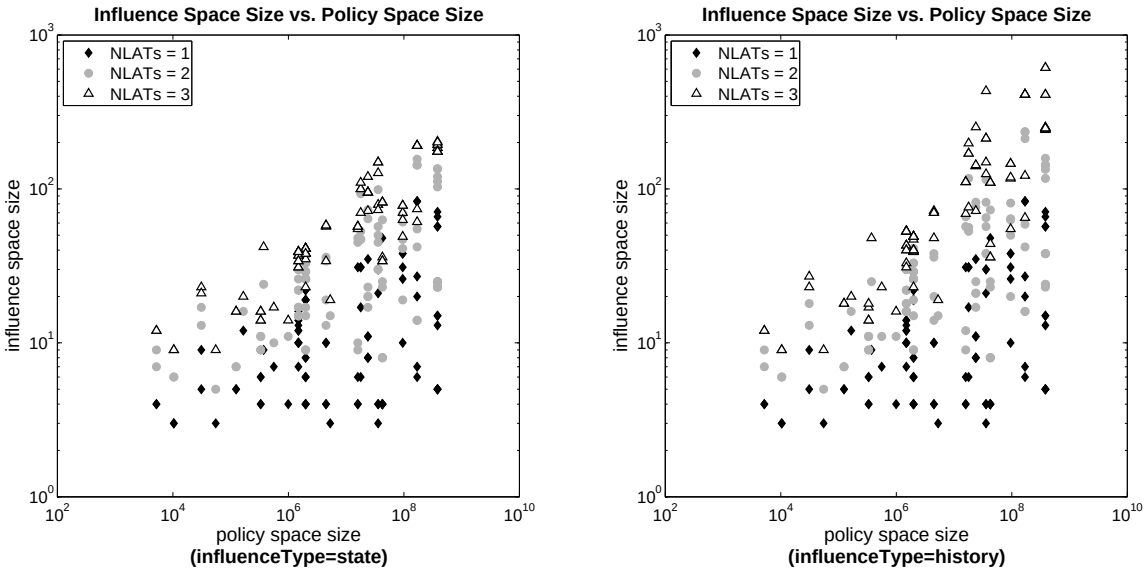


Figure 4.18: Scatter plot of policy space sizes respective influence space sizes.

the same sets of problems, again classified by the number of nonlocally affecting tasks.

Here, we observe a slight correlation between policy space size and influence space size. The correlation appears to be strongest when the number of nonlocally affecting tasks is greatest. This is somewhat intuitive since a greater value of $NLATs$ corresponds to a larger percentage of agent i's tasks that are nonlocally affecting, and thus a larger portion of policy decisions that impact the transitions of nonlocal features. The correlation is weakest for $NLATs = 1$ (plotted with black diamonds). Clearly, there are other factors at play that are causing the large variance in the influence space size. In the next subsection, I explore some of these other factors.

4.6.2.5 Window of Nonlocal Feature Manipulation

In Section 4.6.2.2, I analyzed the effects of manipulating tasks' execution windows via a parameter $localWindowSize$, which set the size of all of agent i's task windows. Here, I exert finer control, targeting only those nonlocally-affecting tasks' windows, using an analogous[21] parameter $NLATWindow$. By manipulating the windows of the nonlocally-affecting tasks, I am able to control the proportion of decision steps during which each nonlocal feature may be affected. I now test the initial hypothesis that, all else being equal, a larger window of nonlocal feature manipulation will cause an increase in the degree of influence.

[21]Whereas $localWindowSize$ specifies the size *relative* to the time horizon T, $NLATWindow$ specifies the size of the nonlocally-affecting tasks window in *absolute* terms.

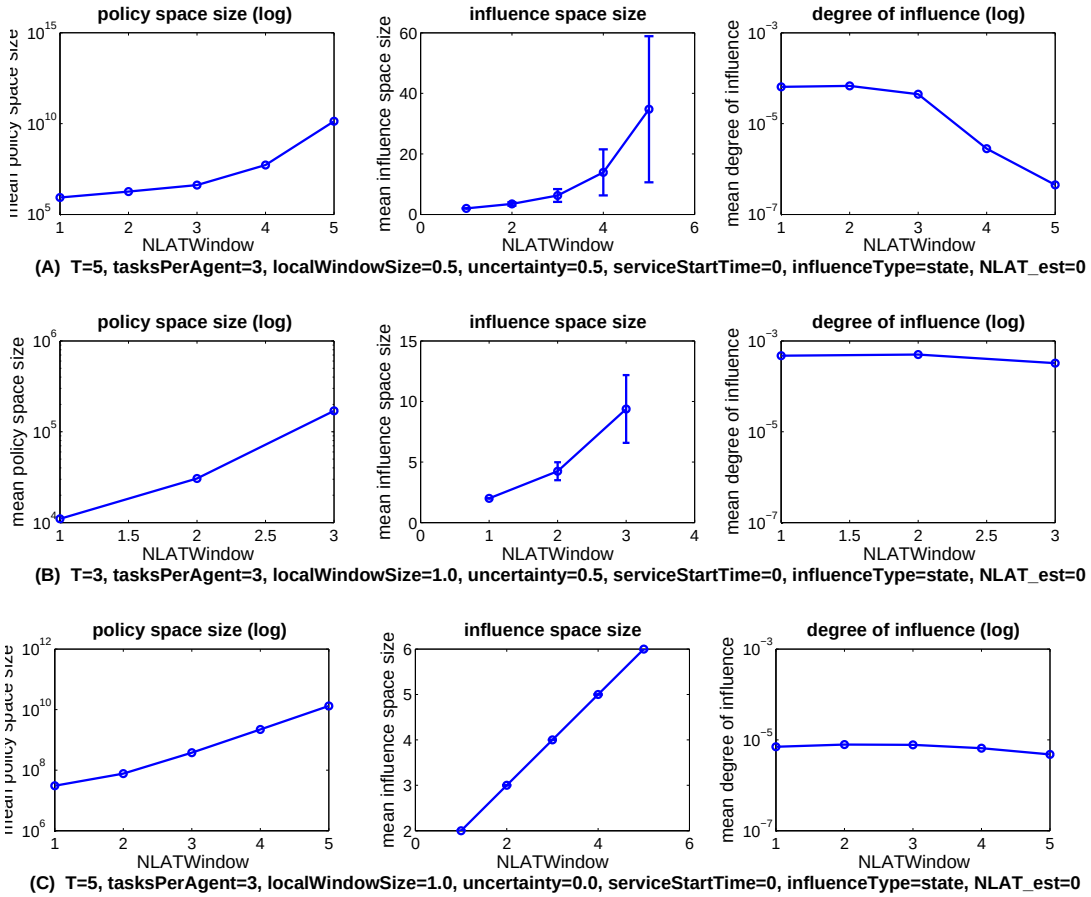


Figure 4.19: Varying the size of the nonlocal feature manipulation window.

Contrary to my hypothesis, empirical results showed an inverse relationship between the degree of influence and the size of the nonlocally-affecting task's window across all baseline parameter settings, 3 of which are shown in Figure 4.19. Due to the sensitivity of the policy space size, we observe an (often super-) exponential increase when even just a single task's window is expanded. In case A, the policy space growth overwhelms the growth of the influence space, resulting in a significant decrease in the degree of influence as the nonlocally-affecting tasks window was increased from 1 time unit to the length of the problem horizon T (a trend which was most common across all parameter settings). For small problems, such as that shown in case B, the growth of influence space nearly matches that of the policy space, yielding a degree of influence that is relatively flat. For case C, $uncertainty = 0$ indicates that all tasks have deterministic durations. In this case, determinism forces a linear increase in the number of influences (as was previously described in Section 4.6.2.2). The policy space growth is similarly stunted, which results in a degree of influences which is relatively

unaffected by the size of the nonlocally-affecting task's window.

Although my initial hypothesis regarding the increasing degree of influence turned out to be false, we can distill from these results a strong trend in the increase of the influence space. For all nondeterministic cases (including those not shown), the influence space size grew exponentially with the increasing nonlocally affecting task window. Just as we observed in Section 4.6.2.4, however, the increase in the number of influences is accompanied by an exponential increase in the variance. Next, I examine one other factor contributing to this variance.

I hypothesize that it is not only the size of the window of nonlocal feature manipulation that affects the influence space size, but also the window's temporal placement. Intuitively, the later the window of nonlocal feature manipulation, the more decisions that agent i has made before interacting with others, and hence the more that could have transpired locally before an interaction takes place, corresponding to an increasing number of possible trajectories of agent i before setting the nonlocal feature. I hypothesize that, as a consequence, the later the window of nonlocal feature manipulation, the more feasible influence points there will be in general. I test this hypothesis by simultaneously varying the size of the nonlocally-affecting task's window and location of its window (as controlled by the earliest start time $NLAT\_est$).

The results, plotted in Figure 4.20, confirm that this hypothesis holds true for the problems in my testbed. The only exceptions are settings for which there are no other tasks besides the single nonlocally-affecting task or for which the other tasks are all deterministic[22], as in case A. For these exceptions, the influence space size remains constant because for each time t that agent i can start its nonlocally affecting task, regardless of the value of t, there is necessarily a single influence point due to the certainty that agent i will indeed start the task at time t.[23]

In both cases B and C, the influence space size increases exponentially as the nonlocally-affecting task window is shifted forwarded in time, as predicted. However, the degree of influence exhibits more complicated behavior. At a high level, as the local window size is increased, progressing from 0.0 in case A to 1.0 in case C, the degree goes from strictly decreasing (in case A) to wavering (in case B), to strictly increasing in case C. Given the inverse relationship between policy space size and degree of influence, the trend that we observe in the degree of influence is a result of the opposite trend in the size of the policy space.

[22]Determinism in case A is due to the setting $localWindowSize = 0.0$, dictating that each local task has just a single outcome with duration 1 and probability 1.

[23]Although there is uncertainty as to when the nonlocally-affecting task will finish, this uncertainty is encoded in a single influence.

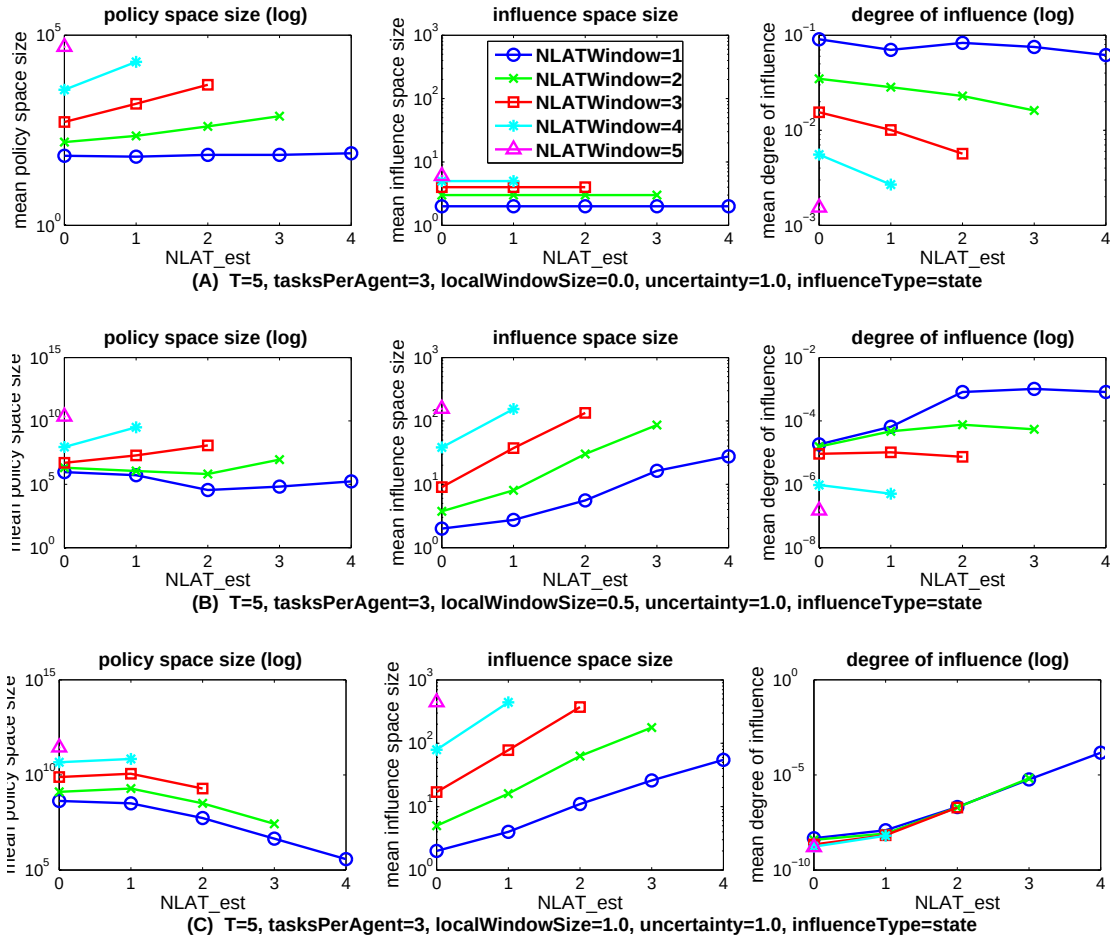


Figure 4.20: Varying the nonlocally-affecting task's start time *and* window size.

The explanation behind this trend is as follows. As the nonlocally-affecting task window is shifted forward in time, there are two opposing forces being exerted on the policy space. First, there is policy space growth due to the fact that there are more states at later times than at earlier times, and hence more opportunities for increasing the number of policy decisions when the nonlocally affecting task is allowed to be started later on. In essence, the longer agent i delays a decision about its interaction, the more that could have transpired, and the more circumstances that it will need to consider. This force is dominant in case A as well as in cases B and C given that $NLATWindow$ is large.

Second, the later agent i may begin its nonlocally-affecting task, the later the branching due to the uncertain task outcomes occurs, and the smaller the subsequent increase in states at later times and the smaller the corresponding rise in the number of decisions due to these additional states. In essence, the longer that agent i delays

its interactions, the shorter-lasting the consequences of its interactions will be, and the less it will need to reason about thereafter. This second force is dominant when local tasks' windows are large and also when the nonlocally affecting task window is small. This makes sense because this second force is magnified when the branching factor (due to additional actions) is increased, which occurs when local task windows are enlarged (as we observed in Section 4.6.2.2).

4.6.3 Summary of Findings

The empirical results that I have presented in Sections 4.6.2.1–4.6.2.5 provide the following insights:

- In general, as an agent's local decision problem becomes more complex (involving more states and actions, and more uncertainty), there is an increasing number of influences that the agent can exert on its peers. However, in almost all cases, the rate of policy space growth exceeds that of influence space growth. Consequently, agents with more complex local behavior tend to have a smaller *degree of influence*. To validate that this result was not restricted to the space of relatively small problems considered above, I ran two additional tests on a set of larger problems. The results, shown in Figure 4.21, anecdotally corroborate that the trends observed in my experiments can be extrapolated to problems with time horizons of 10 or greater.[24] These trends suggests that the potential advantages of coordinating abstract influence (over coordinating full policies) are magnified as agents' local behavior becomes more complex.

- As the size of an agent's influence encoding increases, the number of points in its feasible influence space tends to increase. Specifically, we observed an increase in the average influence space size when the encoded distribution included more nonlocal features and also when its probabilities were conditioned on history instead of on state. We also observed that the growth in the influence space due to the more verbose history-dependent encoding was greater when the problem uncertainty increased. With respect to the degree of influence, agents whose interactions can be encoded more compactly tend to be more weakly-coupled. As a consequence, influence-based abstraction is (on average) more effective at reducing the size of the search space when the influence encoding is small.

[24]Figure 4.21A extends my analysis of growing number of decision steps (from Section 4.6.2.1) and Figure 4.21B performs the same comparison of varying numbers of tasks and task window sizes from Section 4.6.2.2, but for larger problems with a time horizon of 10.

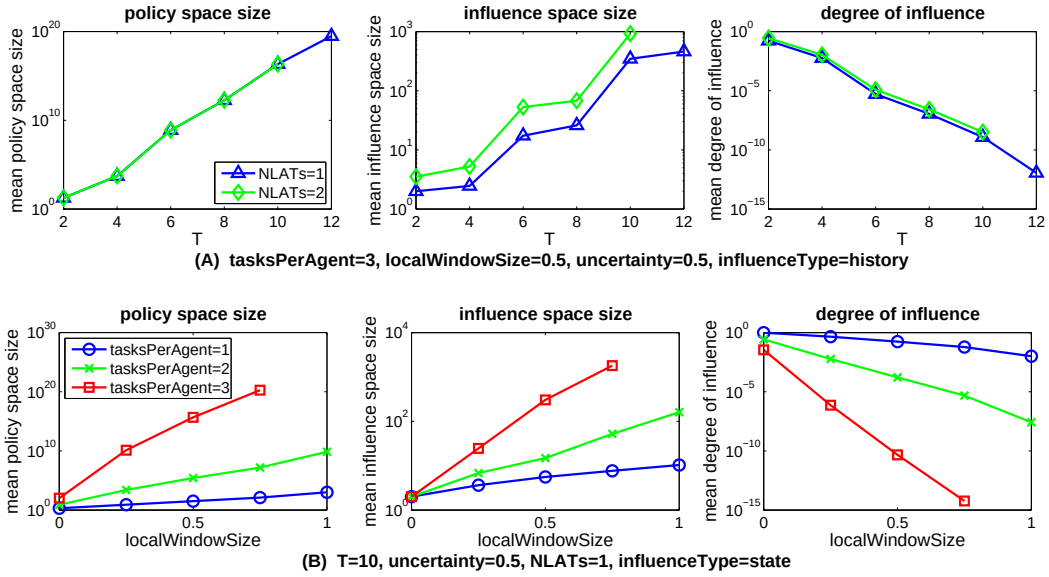


Figure 4.21: Increasing the size of the local decision problem.

- In general, along with increasing influence space size, we also observed a significant increase in the variance of the influence space size. This indicates that the computation required to search the influence space (exhaustively) is increasingly unpredictable for problems with more complex influence encodings. To combat this unpredictability, I have identified several characteristics that can be used to gauge a problem's influence space size and its degree of influence:

  1. The size of the policy space appears to be weakly correlated with the size of the influence space.

  2. Not surprisingly, increasing the size of the window during which agents are allowed to interact tends to increase the number of feasible influence points. However, the additional number of policies that results from the agents' greater interaction flexibility tends to overwhelm the growth of the influence space. This suggests that giving agents a broader array of choices about when to interact actually makes them *more* weakly coupled.[25]

  3. For nondeterministic problems, moving the window of interaction forward in time tends to increase the number of feasible influence points. Intuitively, the later an agent's interaction may occur, the more that can transpire before the interaction, and so the greater the uncertainty in if and when

[25]This result indicates that there is some discrepancy between the intuitive definition of weak coupling and the semantics that I have presented in Section 3.5.

the interaction will take place. This also tends to increase the degree of influence. However, for cases in which the interaction window is small, or in which there is little uncertainty, moving the window of interaction forward in time can cause a decrease in the degree of influence (and thus an increase in the effectiveness of influence-based abstraction at reducing the size of the policy space).

4.7 Summary

This chapter makes several key contributions. First, I developed a novel best-response model whose computational complexity is dependent on the number of shared state features and otherwise independent of the number of peer agents. As such, agents' usage of this best response model constitutes inherent exploitation of reduced *state factor scope* (as formalized in Section 3.5.1.3). Although restricted in its application to TD-POMDP problems, this best response model is the first to exploit such structure in transition-dependent agents (while still providing optimal solutions).

Within the larger scheme of this dissertation, in this chapter I have developed a general framework for abstracting agents' transition influences. More significantly, I have proven that the influence abstractions suffice for *optimal* local reasoning about peers' behavior. To begin to evaluate the efficacy of my influence-based abstraction methodology, I have performed an empirical analysis, and presented evidence that influence-based abstraction enables a significant reduction the overall search space. Further, by identifying problem characteristics that impact the size of the influence space and the size of the policy space, my analysis takes steps towards characterizing the circumstances under which influence-based policy abstraction is most advantageous. Although the results presented in this chapter capture only the number of influences and not the computation required to find each point in the feasible influence space, my rigorous analysis of influence-space size and degree of influence lead me, in Chapter 6, to characterize the overall computational advantages and disadvantages of influence-based abstraction (after developing the remaining components of my influence-based solution methodology in Chapter 5).

CHAPTER 5

Constrained Local Policy Formulation

In the last chapter, I identified an alternative search space, the *influence space*, proposing that agents coordinate *influences* instead of policies. However, they cannot do away with policies altogether. Whereas influences convey expectations about select portions of joint behavior, agents' policies provide complete specifications of local behavior, without which a solution to the planning problem would be incomplete. As such, there is an inherent duality of agent reasoning associated with influence-based policy abstraction: individually, agents reason about policies, and jointly, agents reason about influences. The *constrained local policy formulation* methodology that I present in this chapter provides agents with a mapping between the two representations.

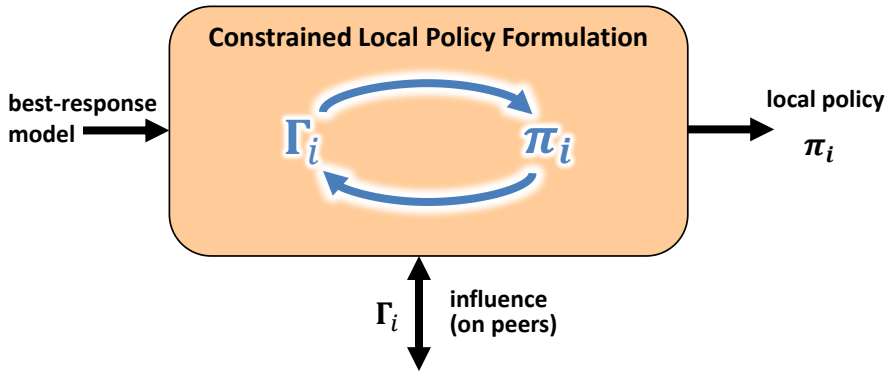


Figure 5.1: Functional diagram of *constrained local policy formulation*.

As indicated by Figure 5.1, which isolates the "constrained local policy formulation" component from the other contents of my approach shown in Figure 1.2, I address both directions of translation. Translating from policy to influence, my methodology allows each agent to extract from any one of its policies the implied influence (on the

agent's peers). Further, given a proposed influence, it allows agents to compute a policy that adheres to the influence, thereby translating from influence to policy.

At the heart of my approach lies a conceptual connection between the *occupation measures* of the MDP dual linear program (LP) formulation (D'Epenoux, 1963; Kallenberg, 1983) and the probabilistic effects that influences encode. In formalizing this relationship, I derive the probability value of each influence component as a function of the occupation measures returned by the MDP LP solution. Further, I develop a novel extension to the MDP LP formulation that incorporates additional constraints so as to guarantee that the solution policy adheres to an agent's proposed influences (if such a policy exists). In contrast to existing alternative approaches, which encourage the enforcement of various forms of influence by biasing the MDP model, my approach strictly enforces agents' influences without the need for parameter tuning or model manipulation, by constraining the policy directly. Further, my approach is guaranteed to produce individual agent policies that are *optimal* with respect to the influence constraints.

5.1 Overview

The contents of this chapter are organized as follows. I begin, in Section 5.2, by formalizing the application of the dual LP to an agent's best response model and incorporating additional mixed-integer constraints for computing and evaluating deterministic policies and for handling partial observations. In Section 5.3, I introduce the relationship between occupation measures and policy effects within the simple, yet restrictive, context of probabilistic goal achievement. I relax this restriction in Section 5.4, extending to *state-dependent* and *history-dependent* influences. Next, in Section 5.5, I contrast my constrained policy formulation methodology with alternative approaches. In Section 5.6, I develop an algorithm that iteratively enumerates all of an agent's outgoing influences and analyze its complexity. I conclude the chapter with a summary of its contributions in Section 5.7.

5.2 Applying the Dual LP Formulation

Among the various single-agent (PO)MDP solution methods reviewed in Section 2.2.1.2, an agent may employ the LP formulation from Equation 5.1 to solve its best-response model (developed in Section 4.2). In review, the best-response model incorporates all peers' influences into its transition function. During the planning

process, peers propose influences, at which point the agent can use its best-response model to reason about its local behavior as if it was alone in the world (since the influences of its peers have been fixed). Throughout this section, I treat the agent, reasoning with its best-response model, in isolation. In general, the TD-POMDP best-response model is partially observable, however, for the moment, let us assume that is a completely-observable single-agent MDP. In Section 5.2.3, I describe an extension for applying LP techniques to partially-observable models. In Sections 5.2.1–5.2.2, I describe extensions for computing and evaluating deterministic policies. But before then, let me formally re-introduce the basic form of MDP dual linear program (D'Epenoux, 1963; Kallenberg, 1983).

The variables $\boldsymbol{x} = \langle x(s,a), \forall s \in S, \forall a \in A \rangle$ of the LP, called *occupation measures*, model the expected (discounted) number of times that action a is taken in state s. For Dec-POMDPs, and consequently for agents' local best response models, the time horizon is finite and the discount factor $\gamma = 1$.[1] Thus, for our purposes, the occupation measures specify the expected *non-discounted* number of times action a is taken in state s from time steps 0 to T (the finite horizon).

$$\max_{\boldsymbol{x}} \sum_{s \in S} \sum_{a \in A} x(s,a) R(s,a)$$
$$\left| \begin{array}{l} \forall s^{t+1} \in S, \sum_{a^{t+1} \in A} x(s^{t+1}, a^{t+1}) - \sum_{s^t \in S} \sum_{a^t \in A} x(s^t, a^t) P\left(s^{t+1} | s^t, a^t\right) = \alpha(s^{t+1}) \\ \forall s \in S, \forall a \in A, x(s,a) \geq 0 \end{array} \right. \tag{5.1}$$

The first constraint of the LP in Equation 5.1 can be thought of as conserving the flow of probability through each state s^{t+1}, requiring that the expected number of times s^{t+1} is exited (the *flow out*) subtracted from the expected number of times s^{t+1} is entered (the *flow in*) be equal the probability of starting in s^{t+1}. The second constraint forces each occupation measure to be no less than zero.

The output returned by an LP solver is a solution, which is a setting of variables (in this case $\boldsymbol{x}$) that maximizes the objective function, in addition to the resulting maximal objective value. In Equation 5.1, the objective function $\max_{\boldsymbol{x}} \sum_{s \in S} \sum_{a \in A} x(s,a) R(s,a)$ ensures that the solution to the LP, which I will refer to as the optimal occupation vector $\boldsymbol{x}^*$, maximizes the expected accumulation of rewards.

Upon computing the optimal occupation vector $\boldsymbol{x}^*$, the agent can recover its

[1] Throughout this chapter, I assume that $\gamma = 1$. The extent to which my formalism can be extended to cases where $\gamma < 1$ is the subject of future work.

corresponding best-response policy as follows:

$$\pi^*(s,a) = \frac{x^*(s,a)}{\sum_{a' \in A} x^*(s,a')} \tag{5.2}$$

Similarly, the agent's (local) value $V(\pi^*)$ of the (local) policy π^* that was computed using Equation 5.2 is simply the value of the objective function:

$$V(\pi^*) = \sum_{s \in S} \sum_{a \in A} x^*(s,a) R(s,a) \tag{5.3}$$

5.2.1 Constraining the LP to Return a Deterministic Policy

In general, the policy π^* returned (in Equation 5.2) by the MDP LP is *stochastic*, prescribing a probability with which the agent shall take each action in each state ($\pi : S \times A \mapsto (0,1)$). There are an infinite number of such policies. In the interest of maintaining a finite search space, my overarching solution methodology for planning coordinated behavior (developed in Section 5.6 and in Chapter 6) restricts itself to *deterministic* policies of the form $\pi : S \mapsto A$. Fortunately, it is straightforward to constrain the LP from Equation 5.1 to return the optimal deterministic policy. However, it entails transforming the LP into a mixed-integer LP (MILP), making policy computation more costly in general. As such, in adopting this extension, the solution algorithms developed in this dissertation inherently trade potential reductions in computational complexity of local policy computation for finiteness of joint policy space and ease of searching the space (as I describe in Section 5.6).

Equation 5.4 computes deterministic policies by extending the standard MDP dual LP (Equation 5.1) with additional variables and constraints. Here , I introduce a vector of boolean variables $\boldsymbol{z} = \langle z(s,a) \in \{0,1\}, \forall s \in S, \forall a \in A \rangle$, whose values indicate whether or not the corresponding occupation measures are greater than zero. Each pair $\{z(s,a), x(s,a)\}$ is thereby connected by a constraint $-1 \leq (x(s,a) - z(s,a)) \leq 0$, requiring that $z(s,a) = 1$ whenever $x(s,a) > 0$ (but not the converse). Subsequently, an additional constraint for each state s of the form $\left(\sum_{a \in A} z(s,a)\right) = 1$, restricts that at most one of $\{z(s,a), z(s,a'), z(s,a''), ...\}$, and thus at most one of $\{x(s,a), x(s,a'), x(s,a''), ...\}$, be nonzero. The solution to the resulting MILP takes the form of an optimal $\langle \boldsymbol{x}^*, \boldsymbol{z}^* \rangle$ pair.

$$\max_{\boldsymbol{x}} \sum_{s \in S} \sum_{a \in A} x(s,a) R(s,a)$$
$$\left| \begin{array}{l} \forall s^{t+1} \in S, \sum\limits_{a^{t+1} \in A} x(s^{t+1}, a^{t+1}) - \sum\limits_{s^t \in S} \sum\limits_{a^t \in A} x(s^t, a^t) P\left(s^{t+1} | s^t, a^t\right) = \alpha(s^{t+1}) \\ \forall s \in S, \forall a \in A,\ -1 \leq (x(s,a) - z(s,a)) \leq 0 \\ \forall s \in S, \left(\sum\limits_{a \in A} z(s,a) \right) = 1 \\ \forall s \in S, \forall a \in A,\ x(s,a) \geq 0 \\ \forall s \in S, \forall a \in A,\ z(s,a) \in \{0,1\} \end{array} \right. \tag{5.4}$$

Upon computing $\langle \boldsymbol{x}^*, \boldsymbol{z}^* \rangle$ subject to the above constraints, the agent recovers its *deterministic* policy as:

$$\pi^*(s) = \arg\max_a z^*(s,a), \tag{5.5}$$

wherein a single action a is assigned to each state s. Note that $z(s,a) \in \{0,1\}$, and $z^*(s,a) = 1$ indicates that a is the only action (if any) with a positive occupation measure in state s. If state s is unreachable via π^*, the deterministic action may be selected arbitrarily by the MILP solver.

In general, enforcing deterministic policies in this manner results in a harder optimization problem. With the addition of integer variables $\boldsymbol{z}$, Equation 5.4 defines a mixed-integer linear program (MILP), whose worst-case complexity is no longer *polynomial*; instead, the best known algorithms take exponential time (in the number of variables) in the worst case.

5.2.2 Evaluating Deterministic Policies

In addition to computing optimal policies, an agent may use another variation of the basic MDP dual LP to evaluate any candidate deterministic policy π (computed by a linear program or otherwise). To do so, it is simply a matter of disallowing all actions other than those specified by the policy π:

$$\forall s \in S, \forall a \in A \text{ s.t. } a \neq \pi(s),\ x(s,a) = 0 \tag{5.6}$$

The constraints given in Equation 5.6, when added to the original LP from Equation 5.1, derives from policy π its implied occupation measures $\boldsymbol{x}$ (consistent with the transition dynamics of the MDP). During the process of solving this LP, its

objective function is evaluated, and π's corresponding value computed.

5.2.3 Handling Partial Observability

Next, I describe an extension for computing optimal POMDP policies. The idea is to model the observation history together with state in an MILP, such that occupation measures $\boldsymbol{x}$ are defined over both state and observation history: $x(s^t, \vec{o}^t, a)$ refers to the probability that the agent observes $\vec{o}^t$ from times $(1, \ldots, t)$, is in state s at time t, and takes action a. Inevitably, the size of the occupation measure vector will be larger than that required for fully-observable problems (growing at worst exponentially in the time horizon). However, the ensuing computational overhead is not unreasonable for problems wherein the observation history can be encoded compactly.

Note that the original semantics of occupation $x(s^t, a)$ can easily be recovered from the POMDP occupation measures $x(s^t, \vec{o}^t, a)$:

$$x(s^t, a) = \sum_{\vec{o}^t} x(s^t, \vec{o}^t, a), \tag{5.7}$$

which follows from the fact that two different observation histories cannot both occur in a single execution trajectory. Analogously, the POMDP LP objective function is simply $\max_{\boldsymbol{x}} \sum_s \sum_{\vec{o}} \sum_a x(s, \vec{o}, a) R(s, a)$.

The flow constraint in the POMDP LP (which is an extension of the first constraint in Equation 5.1) must account for the probability of encountering state-observation pair (s^{t+1}, o^{t+1}) given that action a^t was taken in state s^t upon observing history $\vec{o}^t$:

$$\begin{aligned} &\forall s^{t+1}, \forall \vec{o}^{t+1} = \left\langle \vec{o}^t, o^{t+1} \right\rangle, \\ &\quad \sum_{a^{t+1}} x(s^{t+1}, \vec{o}^{t+1}, a^{t+1}) - \sum_{s^t} \sum_{a^t} x(s^t, \vec{o}^t, a^t) P\left(s^{t+1} | s^t, a^t\right) O(o^{t+1} | a^t, s^t + 1) = 0, \end{aligned} \tag{5.8}$$

which includes both the POMDP state transition probability and the probability of the new observation (as prescribed by the observation function O introduced in Section 2.2.2). Equation 5.8 constrains the flow from one observation to the next. An additional constraint is required to account for the start state distribution $\boldsymbol{\alpha}$:

$$\forall s^0 \in S, \forall \vec{o}, \sum_{a^0} x(s^0, \vec{o}, a^0) = \begin{cases} \alpha(s^0), & \text{if } \vec{o} = \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

Just as in Section 5.2.1, we will use the integer variables $\boldsymbol{z}$ to constrain the policy to be deterministic. For the POMDP LP, we need a $z(s, \vec{o}, a)$ value for every element of $\boldsymbol{x}$. The deterministic policies constraints (not shown here) are otherwise identical to those given in Section 5.2.1.

Given that our occupation measures store both state and observation history information, one additional set of constraints is needed. For the POMDP, policies map observations histories (but not states) to actions. Thus, a valid policy must assign the same action to all state-observation-history pairs with identical observation history:

$$\forall \vec{o}, \forall a, \sum_{s \in S} z(s, \vec{o}, a) = \|S\| \cdot z(s_0, \vec{o}, a) \tag{5.10}$$

In combination with the deterministic policy constraints and involving the variables z developed in Section 5.2.1, Equation 5.10 requires that, for every unique observation history and for every action, the summation of z values must be equal to the number of states multiplied by the z value of one (arbitrarily chosen) state s_0. The consequence is that all observation-history-action pairs must have the same z value. By the semantics of $\boldsymbol{z}$, the same deterministic action must be chosen for every observation history regardless of action. Just as in Section 5.2.1, the deterministic action is easily recovered by selecting the single action with a nonnegative value $z(s, a)$ for each state:

$$\pi^*(\vec{o}) = \arg\max_a z^*(s_0, \vec{o}, a), \tag{5.11}$$

where state s_0 is arbitrarily chosen. Note that the POMDP extension described above may be used in combination with any of the other extensions developed in the remainder of this chapter.

5.3 Probabilistic Goal Achievement

As I have described in the last Section, the dual LP expresses *occupation measures* that suffice as an alternate representation of an agent's policy. Whereas the conventional representation of a deterministic policy maps states to actions, occupation measures instead articulate expected state-action statistics, thereby providing a richer encoding. Subject to the assumption that the state space is acyclic, occupation measures encode an agent's probabilistic action effects (which are the result of it policy applied to its transition model) in addition to its action choices. The fact that this probabilistic information is intrinsic to the MDP linear program means that we can manipulate the policy formulation process at its core, explicitly specifying

constraints and objectives on desired probabilistic effects. In this section, I describe a simple application of this concept before explicitly connecting it to influence-based abstraction in the next section.

Consider that, in addition to maximizing utility, an agent has other aspirations that cannot easily be accounted for in its utility function. In particular, the agent would like to find the highest-valued policy that reaches a set of *goal states* $S_g \subset S$. Let us assume that in any given trajectory, at most one state $s_g \in S_g$ may be encountered. In the TD-POMDP, states are time indexed and so cannot be visited more than once in any execution trajectory. Thus, the probability of reaching any goal state s_g is equal to the summation of the probabilities of reaching each goal state. Moreover, the probabilities of reaching s_g is captured by the summation of occupation measures associated with s_g.

By adapting the basic MDP dual LP from Equation 5.1, the agent can compute a policy guaranteed to reach exactly one of its goal states with the addition of a single constraint:

$$\left(\sum_{\{s \in S_g\}} \sum_{a \in A} x(s,a) \right) = 1$$

In essence, the agent is directly constraining its policy to achieve its goals. Although the objective function remains the same—to maximize the expected summation of rewards—the agent will now compute the highest-valued policy that reaches a goal state (if such a policy exists). More generally, the agent can constrain its policy to achieve its goal with probability $\geq \rho$ by constraining the occupation measures as:

$$\left(\sum_{\{s \in S_g\}} \sum_{a \in A} x(s,a) \right) \geq \rho$$

Yet another alternative is to alter the objective function such that the agent achieves its goals with maximal probability:

$$\rho_{\max} = \max_{\boldsymbol{x}} \left(\sum_{\{s \in S_g\}} \sum_{a \in A} x(s,a) \right)$$

The strength of this approach, in contrast to other policy formulation techniques (e.g. policy iteration, value iteration, dynamic programming), is its ability to constrain policies precisely while still maintaining optimality (with respect to the constraints). If there does not exist a policy that will satisfy the agent's probabilistic goal constraints,

the LP solver will return "no solution" and the agent will know that its goals are over-constraining.

5.4 State-Dependent Influence Achievement

Just like probabilities of fulfilling goals, agents' *influences* (as formalized in Section 4.3) are also directly related to the MDP LP's occupation measures. Let us begin by considering a state-dependent influence $\Gamma_{\pi_i}(\bar{n})$, which consists of a set of parameters, each taking the form $\gamma = Pr\left(\bar{n}^{t+1} = \hat{n} | \bar{f}^t = \hat{f}\right)$, whose semantics are as follows: given that agents are in a state $s^t \in D_\gamma^t \equiv \left\{s^t \in S | \bar{f}(s^t) = \hat{f}\right\}$ at time t, then with probability γ the influencing agent's policy will cause the transition into a state $s^{t+1} \in E_\gamma^{t+1} \equiv \{s^{t+1} \in S | \bar{n}(s^t) = \hat{n}\}$ at time $t+1$ in which the prescribed effect has been achieved. Starting from this definition, we can rewrite the equation for γ as follows:

$$\gamma = Pr\left(\bar{n}^{t+1} = \hat{n} | \bar{f}^t = \hat{f}\right) = Pr\left(E_\gamma^{t+1} | D_\gamma^t\right)$$

given the semantics of influence

$$= \frac{Pr\left(E_\gamma^{t+1}, D_\gamma^t\right)}{Pr\left(D_\gamma^t\right)}$$

by definition of conditional probability

$$= \frac{\sum_{s^t \in D_\gamma^t} \sum_{s^{t+1} \in E_\gamma^{t+1}} Pr\left(s^t, s^{t+1}\right)}{\sum_{s^t \in D_\gamma^t} Pr\left(s^t\right)}$$

because agents cannot occupy multiple states simultaneously

$$= \frac{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} \sum_{s^{t+1} \in E_\gamma^{t+1}} Pr\left(s^t, a^t, s^{t+1}\right)}{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} Pr\left(s^t, a^t\right)}$$

by the law of total probability

$$= \frac{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} \sum_{s^{t+1} \in E_\gamma^{t+1}} Pr\left(s^{t+1} | s^t, a^t\right) Pr\left(s^t, a^t\right)}{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} Pr\left(s^t, a^t\right)}$$

by definition of conditional probability

$$= \frac{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} \sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1} | s^t, a^t\right) x\left(s^t, a^t\right)}{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} x\left(s^t, a^t\right)} \tag{5.12}$$

subsituting agent i's transition function P() and occupation measures $\boldsymbol{x}$

Alternatively, a state-dependent influence may have no conditioned evidence, thereby expressing a prior probability $Pr\left(\bar{n}^0 = \hat{n} | \emptyset\right)$. In this case, γ depends only on the start state distribution α:

$$\begin{aligned}
\gamma &= Pr\left(\bar{n}^0 = \hat{n}\right) = Pr\left(E_\gamma^0\right) \\
&\textit{given the semantics of influence} \\
&= \sum_{s^0 \in E_\gamma^0} Pr\left(s^0\right) \\
&= \sum_{s^0 \in E_\gamma^0} \alpha\left(s^0\right) \qquad (5.13) \\
&\textit{substituting agent i's start state probabilities } \boldsymbol{\alpha}
\end{aligned}$$

By the above derivations, given any candidate policy π_i, agent i can compute its outgoing influence Γ_{π_i} by (1) evaluating policy π_i using the LP described in Section 5.2.2, thereby returning a vector of occupation measures $\boldsymbol{x}$, and (2) evaluating the derived expressions in Equation 5.12 (or Eq. 5.13) for each $\gamma \in \Gamma_{\pi_i}$. The MDP LP thereby suffices as a method of state-dependent influence abstraction, translating from policies to influences.

The other direction of translation—from influence to policy—may be achieved by incorporating influence constraints, which I derive as follows, by turning equation 5.12 on its head using algebraic manipulation.

$$\frac{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} \sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1}|s^t, a^t\right) x\left(s^t, a^t\right)}{\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} x\left(s^t, a^t\right)} = \gamma$$

$$\Leftrightarrow$$

$$\left[\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} \sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1}|s^t, a^t\right) x\left(s^t, a^t\right)\right] - \left[\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} \gamma x\left(s^t, a^t\right)\right] = 0 \tag{5.14}$$

$$\Leftrightarrow$$

$$\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} x\left(s^t, a^t\right) \left(\left[\sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1}|s^t, a^t\right)\right] - \gamma\right) = 0$$

Computing a policy that constrains agent i to fulfill an influence Γ_{π_i} is thereby achieved with the addition of constraints of the form derived in Equation 5.14, one for each $\gamma \in \Gamma_{\pi_i}$, to the standard MDP LP from Equation 5.1. Putting it all together:

$$\begin{aligned}
&\max_{\boldsymbol{x}} \sum_{s \in S} \sum_{a \in A} x(s,a) R(s,a) \\
&\left|\begin{array}{l}
\forall s^{t+1} \in S, \sum_{a^{t+1} \in A} x(s^{t+1}, a^{t+1}) - \sum_{s^t \in S} \sum_{a^t \in A} x(s^t, a^t) P\left(s^{t+1}|s^t, a^t\right) = \alpha(s^{t+1}) \\
\forall \gamma \in \Gamma_i, \sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} x\left(s^t, a^t\right) \left(\left[\sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1}|s^t, a^t\right)\right] - \gamma\right) = 0 \\
\forall s \in S, \forall a \in A, x(s,a) \geq 0
\end{array}\right.
\end{aligned} \tag{5.15}$$

The solution $\boldsymbol{x}^*$ to the LP in Equation 5.15 corresponds to a policy $\pi_i^*(\Gamma_i)$ that maximizes agent i's local utility with respect to the candidate influence setting Γ_i. If the LP returns no solution, this means that there is no local policy that achieves influence setting Γ_i, in which case we say that Γ_i is *infeasible*.

5.4.1 History-Dependent Influence Achievement

Extending the state-dependent influence calculations and constraints from Equations 5.12–5.15 to the case of history-dependent influences is straightforward. The key is to incorporate the necessary history into the state of the best-response model. That

is, for a history-dependent influence of the form $\gamma = Pr\left(\bar{n}^{t+1} = \hat{n} | \vec{f}^t = \hat{f}\right)$, simply model $\vec{f}^t$ as a feature of the state at time t. From the influencing agent's standpoint, a history-dependent influence is effectively no different from a state-dependent influence.

5.5 Alternative Approaches to Constraining Influence

In the preceding sections, I have used linear programming to address the problem of computing an agent's policy that fulfills a desired influence (on its peers). In essence, my LP constraints enforce that the influencing agent achieve a requisite behavior. I now turn to an alternative approach sometimes referred to as *reward shaping* that others have used to enforce desired agent behaviors (e.g., Musliner et al., 2006; Varakantham et al., 2009; Williamson et al., 2009). Here, the (PO)MDP reward function is tweaked by adding rewards or penalties (i.e., negative rewards) to bias the agent towards desirable behavior or away from undesirable behavior (e.g., for coordination with other agents). Upon setting these additional rewards, any standard (PO)MDP solver may be used compute an optimal local policy, which in this case is optimal with respect to the manipulated reward function. Along these lines, reward shaping could be used to bias an agent to achieve a particular influence.

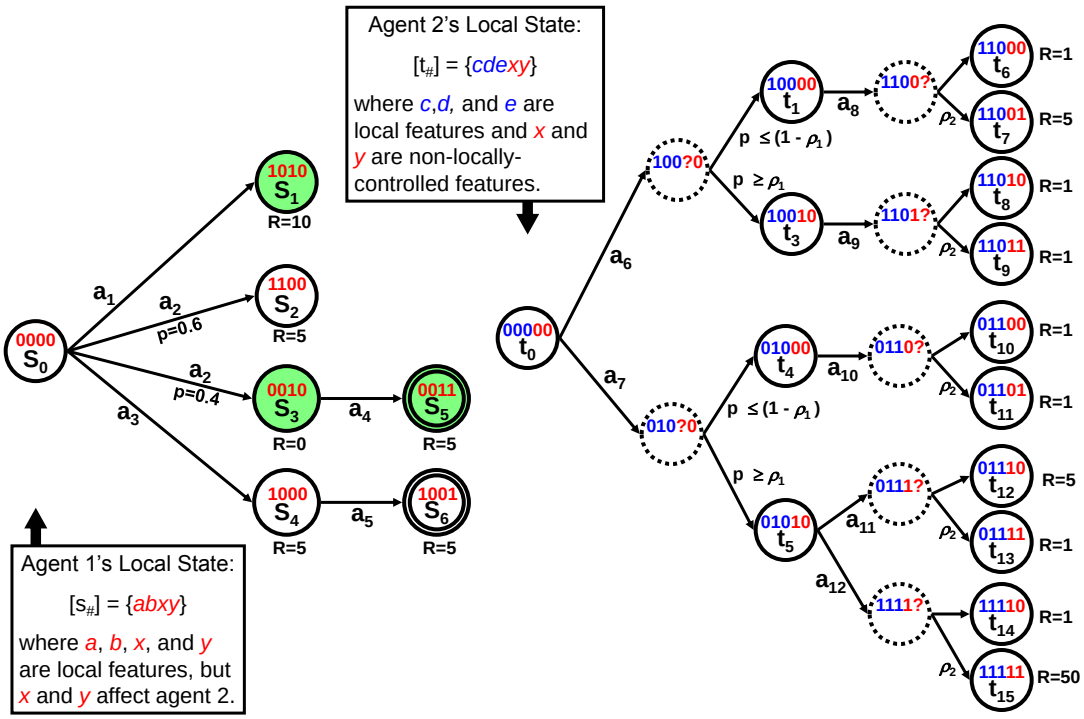


Figure 5.2: A simple, concrete example of influences modeled by two agents.

Example 5.1. The problem shown in Figure 5.2 depicts two agents with simplistic influences, each encoded as a single probability value (ρ). Assume that the objective is to compute a policy that enforces that agent 1 influence agent 2 by setting bit x with probability $\rho_x = 0.4$. In this case, the reward shaping approach will then add extra reward value r_x to states 0010 and 1010 (since these are the states that agent 1 enters upon setting bit x to 1). Analogously, a penalty $p_x \leq 0$ is added to the reward values of states 1100 and 1001, since these are the states at agent 1's time horizon for which arrival means that bit x has never been and will never be set. Notice that if $r_x > 10$ or if $p_x < -10$, action a_1 will be strictly preferred by agent 1. Running an MDP solver on the shaped MDP will invariably yield a_1 as the optimal action choice for agent 1 in state 0000. This enables agent 1 to satisfy its influence whilst maximizing its local utility.

The reward shaping methodology may be effective in some situations, but it is often difficult to set the reward and penalty values appropriately. Moreover, there may not be any values that correctly enforce the commitment.

Example 5.2. Returning to the same two-agent problem shown in Figure 5.2, consider the joint influence $\Gamma = \{\rho_x = 0.4, \rho_y = 0.4\}$, indicating that agent 1 will set x with probability 0.4 and y with probability 0.4. In this case, we will use a reward-penalty pair $\langle r_x, p_x \rangle$ to encourage the setting of bit x (and discourage transitions in which x is not set), and another reward-penalty pair $\langle r_y, p_y \rangle$ to encourage the setting of bit y. Towards selecting appropriate values for our rewards and penalties, let us express agent 1's local value of each of its three policies (which, in this case, correspond to actions a_1, a_2, and a_3), which is the expected sum of rewards received over the course of executing the policy, adding in the r's and p's where appropriate:

$$\begin{aligned} V_1(a_1) &= r_x + p_y + 10 \\ V_2(a_2) &= 0.6(p_x + p_y + 5) + 0.4(r_x + r_y + 5) = 0.4r_x + 0.4r_y + 0.6p_x + 0.6p_y + 5 \\ V_3(a_3) &= r_y + p_x + 10 \end{aligned} \tag{5.16}$$

First, notice that a policy *does* exist which will satisfy the influence $\{\rho_x = 0.4, \rho_y = 0.4\}$. The deterministic local policy which does this is the one that prescribes action a_2 in state 0000. With probability 0.4, the agent will transition into 0010, satisfying $\rho_x = 0.4$ and from there with certainty into 0011, satisfying $\rho_y = 0.4$. Furthermore, it turns out that the *optimal joint policy* dictates that agent 1 should select action a_2. However, as I prove below, we cannot compute this policy by adding extra rewards and penalties.

Theorem 5.3. *There exists an MDP and a set of influences for which:*

1. *there exists a deterministic local policy achieving an influence* Γ*, and*
2. *using reward shaping along with standard deterministic MDP solution techniques, no tuple of the form* $\langle r_x \geq 0, p_x \leq 0, r_y \geq 0, p_y \leq 0\rangle$ *will yield a policy that achieves* Γ*.*

Proof. Consider the MDP in Figure 5.2 and the influence $\Gamma = \{\rho_x = 0.4, \rho_y = 0.4\}$. The only deterministic policy that satisfies this influence, and indeed the only one by which x is set with positive probability and y is set with positive probability, is the policy that selects action a_2. Thus, it suffices to prove that for no values of $\langle r_x \geq 0, p_x \leq 0, r_y \geq 0, p_y \leq 0\rangle$ is action a_2 preferred over action a_1 or action a_3.
To begin with, let us derive preference relations by manipulating the value functions in Equation 5.16 (presented in Example 5.2):

$$\begin{aligned} a_1 \succ a_2 \quad &\iff \quad r_x + p_y + 10 > 0.4r_x + 0.4r_y + 0.6p_x + 0.6p_y + 5 \\ &\qquad \textit{by Equation 5.16} \\ &\iff \quad 5r_x + 5p_y + 50 > 2r_x + 2r_y + 3p_x + 3p_y + 25 \\ &\qquad \textit{by multiplying both sides by } 5 \\ &\iff \quad 3r_x - 3p_x + 25 > 2r_y - 2p_y \qquad (5.17) \\ &\qquad \textit{by subtracting } (2r_x + 3p_x + 5p_y + 25) \textit{ from both sides} \end{aligned}$$

$$
\begin{aligned}
a_3 \succ a_2 \quad &\iff \quad r_y + p_x + 10 > 0.4r_x + 0.4r_y + 0.6p_x + 0.6p_y + 5 \\
&\quad \textit{by Equation 5.16} \\
&\iff \quad 5r_y + 5p_x + 50 > 2r_x + 2r_y + 3p_x + 3p_y + 25 \\
&\quad \textit{by multiplying both sides by } 5 \\
&\iff \quad -2r_x + 2p_x + 25 > -3r_y + 3p_y \qquad (5.18) \\
&\quad \textit{by subtracting } (2r_x + 3p_x + 5r_y + 25) \textit{ from both sides}
\end{aligned}
$$

Now, consider the following cases, which cover all possible combinations of values of $\langle r_x \geq 0, p_x \leq 0, r_y \geq 0, p_y \leq 0 \rangle$:

$\underline{\text{case 1}} : [-2r_x + 2p_x + 25 > -3r_y + 3p_y]$.

$\Rightarrow a_3 \succ a_2$ *by Equation 5.18*

$\underline{\text{case 2}} : [-2r_x + 2p_x + 25 \leq -3r_y + 3p_y]$.

$\Rightarrow 3r_x - 3p_x - 37.5 \geq 4.5r_y - 4.5p_y$ *by multiplying both sides by -1.5*

$\Rightarrow 3r_x - 3p_x + 25 > 4.5r_y - 4.5p_y$ *because* $25 > -37.5$

$\Rightarrow 3r_x - 3p_x + 25 > \frac{4}{9}(4.5r_y - 4.5p_y)$ *by* $\{r_y \geq 0, p_y \leq 0\} \Rightarrow (4.5r_y - 4.5p_y) \geq 0$

$\Rightarrow 3r_x - 3p_x + 25 > 2r_y - 2p_y$

$\Rightarrow a_1 \succ a_2$ *by Equation 5.17*

Thus, for no combinations of values of $\langle r_x \geq 0, p_x \leq 0, r_y \geq 0, p_y \leq 0 \rangle$ is it the case that a_2 is preferred. Therefore, reward shaping cannot be used, in combination with deterministic policy formulation techniques, to enforce Γ. □

In Example 5.2, there are no perfect rewards and penalties that enable the computation of a policy for agent 1 that adheres to the optimal influence setting. For other problems, even if perfect values of r and p exist, it may be difficult to identify what they are. Semantically, the agent is forced to assign value to satisfying the probabilistic effect of the influence versus failing to satisfy it. This value is inherently tied to local policy values dictated by the MDP reward model. If r and p are too close to zero, a policy may be formulated that fails to achieve the desired nonlocal effect (or else achieves it with too small of a probability). But if r and p are too far from zero, then the agent may sacrifice some of its local quality so as to build a policy that

achieves the nonlocal effect with a higher probability than desired.

The primary advantage of the LP approach presented in Section 5.4 is its ability to construct policies that capture influence probabilities perfectly while still maintaining optimality. It is possible that a desired influence settings is infeasible for the influencing agent, meaning that no deterministic policy achieves the influence. In this case the LP solver will return "no solution" and the agent will know immediately that this influence point should not be considered. Otherwise, the returned policy is guaranteed to satisfy the influence. Reward shaping, on the other hand, will return a policy regardless of whether or not a desired influence setting is feasible. Post-processing of the policy (e.g., using Equation 5.12) is then needed to determine whether or not the achieved influence setting matches the desired influence setting.

Due to the difficulties of setting r and p and the lack of optimality guarantees, I do not consider reward shaping further in this dissertation. Instead, my solution approach utilizes constrained linear programming to enumerate influences and to compute optimal local policies around those influences. However, reward shaping has several distinct advantages in other algorithmic contexts. For instance, reward shaping inherently strives for a balance in the costs of nonlocal effects and their anticipated advantages to other agents, making it useful for rapidly converging on approximate joint policies (Musliner et al., 2006; Varakantham et al., 2009). Moreover, unlike constrained linear programming, reward shaping has the added flexibility of accommodating any (PO)MDP solver.

5.6 Exploring the Space of Feasible Influences

In addition to constraining policies to achieve desired influence settings, agents can employ the same methodology to generate the set of all feasible influence settings. That is, for a given influence parameter γ, the influencing agent can enumerate all feasible values $\{\hat{\gamma}\}$ of the parameter achievable by any deterministic policy. It can do so by solving a series of MILPs, each of which looks for a deterministic policy that constrains γ to take on a value that has not previously been considered.

Let the influencing agent iteratively check for the existence of a feasible probability value within an input interval $\hat{\gamma}_{\min} < \hat{\gamma} < \hat{\gamma}_{\max}$ by running an MILP solver on the following program, adapted from Equation 5.15.

$$\max_{\boldsymbol{x}}(\boldsymbol{x} \cdot 0) \quad [\textit{no utility-mazimizing objective}]$$

$$\left|\begin{array}{l}
\textit{...usual constraints for computing deterministic policies (Sec 5.4)...} \\
\forall s^{t+1} \in S, \sum_{a^{t+1} \in A} x(s^{t+1}, a^{t+1}) - \sum_{s^t \in S} \sum_{a^t \in A} x(s^t, a^t) P\left(s^{t+1} | s^t, a^t\right) = \alpha(s^{t+1}) \\
\forall s \in S, \forall a \in A,\ -1 \leq (x(s,a) - z(s,a)) \leq 0 \\
\forall s \in S, \left(\sum_{a \in A} z(s,a)\right) = 1 \\
\forall s \in S, \forall a \in A,\ x(s,a) \geq 0 \\
\forall s \in S, \forall a \in A,\ z(s,a) \in \{0,1\} \\
\textit{...influence parameter setting } \gamma \textit{ must be greater than } \hat{\gamma}_{\min}\textit{...} \\
\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} x\left(s^t, a^t\right) \left(\left[\sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1} | s^t, a^t\right)\right] - \hat{\gamma}_{\min}\right) > 0 \\
\textit{...influence parameter setting } \gamma \textit{ must be less than } \hat{\gamma}_{\max}\textit{...} \\
\sum_{s^t \in D_\gamma^t} \sum_{a^t \in A} x\left(s^t, a^t\right) \left(\left[\sum_{s^{t+1} \in E_\gamma^{t+1}} P\left(s^{t+1} | s^t, a^t\right)\right] - \hat{\gamma}_{\max}\right) < 0 \\
\textit{...all other influence parameters } \gamma' \textit{ must be as prescribed} \\
\forall \gamma' \in \Gamma_i | \textit{prescribed}(\gamma'), \\
\qquad \sum_{s^t \in D_{\gamma'}^t} \sum_{a^t \in A} x\left(s^t, a^t\right) \left(\left[\sum_{s^{t+1} \in E_{\gamma'}^{t+1}} P\left(s^{t+1} | s^t, a^t\right)\right] - \hat{\gamma}'\right) = 0
\end{array}\right. \tag{5.19}$$

Equation 5.19 includes two constraints that enforce an upper and lower bound on the setting of parameter γ (in addition to the constraints from Eq. 5.15 that constrain any other prescribed influence $\gamma' = \hat{\gamma}'$, as well as the deterministic policy constraints from Equation 5.4). Deterministic policy constraints are required so that the agent does not cycle through an infinite set of nondeterministic policies, the elements of which may exert influences whose settings are arbitrarily close to one another.

The agent begins by checking interval ($\hat{\gamma}_{\min} = -\infty, \hat{\gamma}_{\max} = \infty$). If the LP from Equation 5.19 returns a solution, the agent has simultaneously found a new influence $\gamma = \hat{\gamma}_0$ (which may be computed using Equation 5.12) and computed a policy that exerts that influence, subsequently uncovering two new intervals $\{(\hat{\gamma}_{\min}, \hat{\gamma}_0), (\hat{\gamma}_0, \hat{\gamma}_{\max})\}$ to explore. Alternatively, if the LP returns "no solution" for a particular interval, there is no feasible influence within that range. By divide and conquer, the agent can uncover each feasible setting of γ, stopping only after all subintervals have been

Algorithm 5.1 Feasible Influence Enumeration : Single Parameter

EnumerateFeasibleSettingsForParam($\gamma, POMDP_i, \Gamma_i^{\text{prescribed}}$)

...Initialize Interval Queue and settings list...

1: $intervalQ = \emptyset$
2: Push($intervalQ, (-0.1, 1.1)$)
3: $feasibleSettingList = \emptyset$

...Explore Sub-intervals...

4: **while not** IsEmpty($intervalQ$) **do**
5: $(\hat{\gamma}_{\min}, \hat{\gamma}_{\max}) \leftarrow$ Pop($intervalQ$)
6: $\{\boldsymbol{x}, feasible\} \leftarrow$ SolveIntervalLP($POMDP_i, \Gamma_i^{\text{prescribed}}, (\hat{\gamma}_{\min}, \hat{\gamma}_{\max})$) ▷ [Eq. 5.19]
7: **if** $feasible$ **then**
8: $\hat{\gamma}_{\text{new}} \leftarrow$ computeInfluenceSetting($\boldsymbol{x}, \gamma$) ▷ [Eq. 5.12]
9: Add($feasibleSettingsList, \hat{\gamma}_{\text{new}}$)
10: Push($intervalQ, (\hat{\gamma}_{\min}, \hat{\gamma}_{\text{new}})$)
11: Push($intervalQ, (\hat{\gamma}_{\text{new}}, \hat{\gamma}_{\max})$)
12: **end if**
13: **end while**
14: **return** $feasibleSettingsList$

explored. Operating as such, Algorithm 5.1 performs enumeration of all feasible settings of an individual influence parameter γ influenced by agent i (whose local model is denoted $POMDP_i$, and whose already-prescribed influences are denoted $\Gamma_i^{\text{prescribed}}$).

Example 5.1 (continued). Agent 1, from Figure 5.2, passes its ρ_x parameter as the first argument to Algorithm 5.1, its MDP (shown on the left of Figure 5.2) as the second argument, and $\emptyset$ as the third argument (since it is not influenced by any other agents). Within interval $(-\infty, \infty)$, the MILP (line 6) return policy $\pi_1 \equiv a_1$, for instance, which achieves $\rho_x = 1.0$. Next, the algorithm searches intervals $(-\infty, 1.0)$ and $(1.0, -\infty)$, the latter of which will result in an overconstrained MILP. In the end, settings $\rho_x = 1.0$, $\rho_x = 0.4$, and $\rho_x = 0.0$ are returned.

Moreover, an agent i can enumerate all the feasible combinations of its outgoing influence parameter settings. The agent does so by constructing a tree, wherein at each level of the tree, it enumerates the feasible settings of a different parameter γ. As long as the agent orders the parameters consistently with the partial order of the influence DBN, it can reason about one parameter after another, at each node branching for all of the feasible settings of a parameter given the constraint that it achieves the settings

Algorithm 5.2 Feasible Influence Enumeration : All Outgoing Influence Parameters

GenerateFeasibleInfluences($POMDP_i$)

...Start With Unprescribed Influence Parameters...

1: $\Gamma_i \leftarrow$ InitializeOutgoingInfluenceParameters(i)
2: **return** EnumerateSettingsOfRemainingParams($POMDP_i, \Gamma_i$)

EnumerateSettingsOfRemainingParams($POMDP_i, \Gamma_i$)

...Initialize...

1: $settings \leftarrow \emptyset$
2: $\Gamma_i^{\text{prescribed}} \leftarrow$ GetPrescribedSettings(Γ_i)
3: $\gamma \leftarrow$ FirstRemainingParameter(Γ_i)

...If all outgoing influence parameters set, evaluate and return...

4: **if** $\gamma = NIL$ **then**
5: $\quad \langle localVal, \pi_i \rangle \leftarrow$Evaluate($POMDP_i, \Gamma_i$)
6: $\quad$ Add($settings, \langle localVal, \Gamma_i \rangle$)
7: $\quad$ **return** *settings*
8: **end if**

...Enumerate settings of first unprescribed parameter...

9: $\{\hat{\gamma}\} \leftarrow$ EnumerateFeasibleSettingsForParam($\gamma, POMDP_i, \Gamma_i^{\text{prescribed}}$)

...IncorporateSettingsOfRemainingParameters...

10: **for each** $\hat{\gamma} \in \{\hat{\gamma}\}$ **do**
11: $\quad \Gamma_i^{\text{copy}} \leftarrow$ CopyAndPrescribe($\Gamma_i, \gamma, \hat{\gamma}$)
12: $\quad settings_{\hat{\gamma}} \leftarrow$ EnumerateSettingsOfRemainingParams($POMDP_i, \Gamma_i^{\text{copy}}$)
13: $\quad$ AddAll($settings, settings_{\hat{\gamma}}$)
14: **end for**
15: **return** *settings*

of the parameters of its tree ancestors. The leaves of the tree thereby correspond to all feasible combinations of agent i's outgoing influences. Algorithm 5.2 shows the pseudo-code for i's feasible influence generation, which invokes Algorithm 5.1 at each level of the tree (line 9).

Example 5.2 (continued). In the example from Figure 5.2, agent 1 models its influences on agent 2 with two parameters $\{\rho_x, \rho_y\}$. Thus, agent 1 can enumerate all of its feasible outgoing influence settings as follows. First, agent 1 finds each feasible value of ρ_x using Algorithm 5.1, and for each setting $\rho_x = \hat{\rho}_x$ that it finds, agent 1 enumerates the settings of ρ_y that are feasible in combination with $\hat{\rho}_x$.

Consider that, as an alternative to my MILP-guided exploration of the space of an agent's feasible influence points, the agent could instead iterate through all of its possible local policies and manually partition its local policy space into classes with

equivalent influence. In this case, the greater the number of policies, the greater the computation required to enumerate the feasible influences (regardless of the number of influences). The advantage of the MILP-guided enumeration is that it does not directly depend on the number of local policies. Given the structure and character of the feasible influence tree, the number of nodes can be no greater than the product of the number of feasible influence points (i.e., the number of leaves) and the number of parameters (i.e., the depth). Hence, the number of MILPs required to compute the feasible influence tree for influence Γ is $O\left(numberOfParameters(\Gamma) \cdot numberOfFeasibleSettingsOf(\Gamma)\right)$ irrespective of the size of the policy space.

5.7 Summary

The primary contribution of this chapter is a principled approach for constraining agents' policies to adhere to proposed influences. This approach is made possible by drawing a conceptual connection between agents' influence parameters and the transition probabilities implied by occupation measures in the MDP LP. Through the formalization of this connection, I have derived (1) a mapping from the agent's policy to its implied influences on peers and (2) an extension of the MDP LP for computing an influence-constrained policy. In contrast to alternative approaches that use *reward shaping*, which encourage the enforcement of various forms of influence by biasing the MDP model, my approach strictly enforces agents' influences without the need for parameter tuning or model manipulation. Moreover, by constraining the policy directly, my approach is guaranteed to compute a local policy that is optimal with respect to the prescribed influences if such a policy exists; if not, the LP will determine that the influence is infeasible and return "no solution". The same cannot be said for reward shaping. While I am not the first to constrain agents' policies with additional MDP LP constraints (Dolgov & Durfee, 2006; Wu & Durfee, 2010), my approach is the first to formulate constraints pertaining to transition-dependent agents' interactions.

Practically, and in the broader scheme of my optimal solution methodology, I have also developed a novel extension to the influence-constrained linear programming methodology for generating feasible influences. By solving a series of such MILPs, an agent can enumerate its entire set of feasible influences. The significance of this algorithm is that it *avoids* explicit enumeration of all of the agent's local policies. Instead, its computational complexity is dictated by the size of the influence encoding and the size of the feasible influence space.

CHAPTER 6

Optimal Influence-space Search

In relation to the preceding chapters, which provided techniques for modeling individual and joint behavior (Ch. 3), abstracting influences (Ch. 4), and computing influence-constrained local policies (Ch. 5), I now integrate these components into a partially-decentralized algorithm for computing optimal solutions to TD-POMDP problems. My algorithm, Optimal Influence-space Search (OIS), is motivated by the intuition (Sec. 4.5) and empirical evidence (Sec. 4.6) that the space of feasible influence points is potentially significantly smaller than the space of joint policies. Using OIS, agents reason *jointly* about abstract influences and *individually* about their detailed local policies. OIS ultimately returns the *optimal influence*, which is the influence point corresponding to the optimal joint policy.

OIS gains traction and scalability over existing algorithms by leveraging weakly-coupled problem structure. Inherently, OIS takes advantage of agents' low *degree of influence* by searching over the space of influence points (a concept which was introduced in Section 4.5). By decoupling the optimal joint policy formulation into a well-ordered series of influence generations and evaluations, OIS is also able to exploit agents' locality of interaction. In particular, for weakly-coupled problems with a small fixed *agent scope* (Def. 3.30), OIS scales well beyond the state of the art[1], a claim I defend with empirical results in Section 6.6.5.

6.1 Overview

Before developing the mechanics of influence-space search, I first prove that it yields optimal solutions in Section 6.2. Over the course of the remainder of the chapter, I gradually unveil my algorithm, OIS, for searching through the influence space. I

[1] I refer to the state of the art as algorithms (whose results are published) for computing optimal solutions to commonly-studied flavors of transition-dependent Dec-POMDP problems.

begin, in Section 6.3, by presenting OIS in its simplest form—a depth-first search that follows the natural ordering of an acyclic interaction digraph. Next, in Section 6.4, I describe how to adapt the search process to accommodate graphs with directed cycles. Next I present an empirical comparison with four other optimal solution algorithms in Section 6.5, focusing on two-agent problems for which all algorithms are tractable. My analysis serves to assess the degree to which OIS gains a computational advantage through its exploitation of weakly-coupled structure, and also, continuing where my earlier experiments left off, to characterize the problems for which OIS is advantageous as well as those for which it is disadvantageous. I conclude this empirical analysis with a discussion in Section 6.5.7, where I relate my results back to my original claims regarding the efficacy of influence-based abstraction in computing optimal solutions efficiently by exploiting weakly-coupled problem structure. Next, in Section 6.6.3, I develop a substantial enhancement to optimal influence space search for exploiting reduced agent scope, and provide empirical results illustrative of revolutionary advances in agent scalability that are made possible by the complementary exploitation of *degree of influence* along with *agent scope size*.

6.2 Correctness of Optimal Influence-space Search

Let us address the claim that agents can compute the optimal joint policy by exhaustively enumerating and evaluating the space of feasible influence points. Here, I denote an *influence point* as Γ, referring to agents' collective influences (manifested in the form of the *influence DBN* described in Section 4.3.5 whose conditional probabilities are fully specified). Although discussions in previous chapters treated each agent as either *influencing* or *influenced*, this dichotomy does not generalize to problems with more than two agents. Consider an who agent is influenced by its peers but also influences its peers. Such an agent needs to reason about *incoming influences* exerted by its peers in addition to *outgoing influences* that it exerts. Consequently, both of these influence types are contained in the influence DBN.

The following axioms follow directly from the treatment of *influence* presented in Section 4.3:

(A1) Every joint policy π *maps to* some influence point Γ whose conditional probabilities reflect all of the agents' nonlocal features' transition probabilities resulting from the agents adopting π. I will denote this mapping as $\pi \mapsto \Gamma$, and denote the set of feasible influence points as $\{\Gamma | \exists \pi, \pi \mapsto \Gamma\}$.

(A2) Given an influence point Γ and agent i's local policy π_i, agent i can compute its *local value with respect* to Γ, which I will denote $V_i(\pi_i, \Gamma)$, by evaluating π_i in the context of a best-response model injected with Γ's encoded transition probabilities for i's nonlocal features, as long as π_i is consistent with Γ's encoded transition probabilities of i's locally-controlled features.[2]

(A3) Given an influence point Γ, each agent i can compute a *best response to* Γ, which I will denote $\pi_i^*(\Gamma)$, by finding the policy whose local value is greatest $(\arg\max_{\pi_i} V_i(\pi_i, \Gamma))$ subject to the constraint that $\pi_i^*(\Gamma)$ is consistent with Γ's encoded transition probabilities of i's locally-controlled features.

Note that, in contrast to the conventional notion of a best response, here a best response must account for the agent's outgoing influences (corresponding to transitions of its locally-controlled features) as well as its incoming influences (corresponding to the transitions of its nonlocal features). In the context of influence-space search, the agent's best response maximizes its utility *conditioned on* the influences exerted by its peers *subject to* its promised influences on its peers. In this sense, the agent considers the downstream effects of its behavior on others in addition to its own local value. As I prove below, by iterating through these more sophisticated best responses, the team of agents can maximize their joint value.

Definition 6.1. The **optimal joint policy with respect to influence point** Γ, denoted[3] $\pi^{*|\Gamma} = \left\langle \pi_1^{*|\Gamma}, ..., \pi_n^{*|\Gamma} \right\rangle$, is the highest-valued joint policy that maps to Γ:

$$\pi^{*|\Gamma} = \arg\max_{\pi \in \Pi | \pi \mapsto \Gamma} V(\pi)$$

Definition 6.2. The **value of an influence point** Γ, denoted $V(\Gamma)$, is the value of the optimal joint policy with respect to Γ:

$$V(\Gamma) = V\left(\pi^{*|\Gamma}\right) = \max_{\pi | \pi \mapsto \Gamma} V(\pi)$$

[2]The consistency of π_i with respect to Γ may be checked using, for instance, the MILP methodology that I have presented in Chapter 5. Specifically, π_i is consistent if the MILP from Equation 5.1, with additional constraints specified by Equations 5.6 and 5.12, returns a solution.

[3]Note the purposeful difference in notation between agent i's *best response* $\pi_i^*(\Gamma)$ and the ith component $\pi_1^{*|\Gamma}$ of the optimal joint policy with respect to Γ, which are, in principle, two different policies. In Theorem 6.4, I prove that the two must be equally valued.

Theorem 6.3. *An optimal joint policy π^* maps to an influence point Γ^* whose value is the greatest of any feasible influences: $\pi^* \mapsto \left[\Gamma^* = \arg\max_{\Gamma|\exists\pi,\pi\mapsto\Gamma} V(\Gamma)\right]$.*

Proof. By Axiom A1 above, π^* maps to some influence point Γ^*. There can be no other feasible influence $\Gamma' \neq \Gamma^*$ such that $V(\Gamma') > V(\Gamma^*)$. If there were, this would imply the existence of another joint policy π' such that $\pi' \mapsto \Gamma'$ and $V(\pi') > V(\pi^*)$, contradicting the premise that π^* is the optimal joint policy. □

Theorem 6.4. *The value of any influence point Γ is equal to the summation of local values of all agents' best responses: $V(\Gamma) = \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$, where $\pi_i^*(\Gamma)$ is agent i's best response to Γ (using the notation of the above axioms).*

Proof. Let us assume that this theorem is false: that is, $V(\Gamma) \neq \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$.

case 1: $V(\Gamma) < \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$

$\Rightarrow V\left(\pi^{*|\Gamma}\right) < \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$ *by definition of influence value (Def. 6.2)*

$\Rightarrow V\left(\pi^{*|\Gamma}\right) < V\left(\langle \pi_1^*(\Gamma), ..., \pi_n^*(\Gamma)\rangle\right)$ *by Theorem 3.8.*

$\pi^{*|\Gamma} = \arg\max_{\pi\in\Pi|\pi\mapsto\Gamma} V(\pi)$ *by Definition 6.1*

$\Rightarrow$ Contradiction.

case 2: $V(\Gamma) > \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$

$\Rightarrow V\left(\pi^{*|\Gamma}\right) > \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$ *by Definition 6.2*

$\Rightarrow \sum_{i\in\mathcal{N}} V_i\left(\pi^{*|\Gamma}\right) > \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$ *by Theorem 3.8*

$\Rightarrow \sum_{i\in\mathcal{N}} V_i\left(\pi_i^{*|\Gamma}, \Gamma\right) > \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$ *by Axiom A2 above*

$\Rightarrow \exists i$ s.t. $V_i\left(\pi_i^{*|\Gamma}, \Gamma\right) > V_i\left(\pi_i^*(\Gamma), \Gamma\right)$ *by arithmetic*

$\forall i, \pi_i^*(\Gamma) = \arg\max_{\pi_i} V_i(\pi_i, \Gamma)$ *by Axiom A3 above*

$\Rightarrow$ Contradiction.

Therefore, $V(\Gamma) = V\left(\pi^{*|\Gamma}\right)$. □

Corollary 6.5. *Agents can compute an optimal joint policy π^* by:*

1. *exhaustively enumerating all feasible influence points $\{\Gamma|\exists\pi \in \Pi, \pi \mapsto \Gamma\}$,*
2. *evaluating each influence point by individually maximizing the agents' local utilities with respect to the influence point and summing $V(\Gamma) = \sum_{i\in\mathcal{N}} V_i\left(\pi_i^*(\Gamma), \Gamma\right)$,*
3. *selecting the highest-valued influence point $\Gamma^* = \arg\max_\Gamma V(\Gamma)$, and*
4. *individually computing best responses $\pi_i^*(\Gamma^*) = \arg\max_{\pi_i} V_i(\pi_i, \Gamma^*)$ to Γ^*.*

Proof. By Axiom A1, the agents will generate a set of influences in step 1 that includes the optimal influence point Γ^* mapped from the optimal joint policy π^*.

By Theorem 6.4, in step 2, the agents will correctly evaluate each feasible influence, including Γ^*. By Theorem 6.3, the agents will correctly determine that Γ^* is the optimal influence in step 3. Lastly, in step 4, the agents will recover the optimal joint policy by computing best responses to Γ^* (by Axiom A3, and Definition 6.1). □

At a high level, Corollary 6.5 precisely describes the structure of optimal influence-space search. It also proves that this exhaustive search methodology does indeed return optimal solutions, and validates the intuitions that agents can still behave optimally if they jointly reason at the abstract influence level. The search process involves largely-decentralized computation. Each agent individually computes its own local value of each influence point, and each agent individually computes its local policy with respect to the optimal influence point. In the next section, we will see that the generation of influences is similarly decomposable into individual agent computation.

6.3 Depth-First Search

Influence-space search boils down to generating the feasible space of (combined) influence points, where each is a setting of all agents' influences and fully-specifies the influence DBN, and selecting the one that maximizes the sum of agents' best-response utilities. From Section 5.6, we already have a methodology for generating a single agent's outgoing influence settings. The challenge is composing agents' individual generations and evaluations in such a way as to efficiently search the space of combined influence settings. In this section, I develop one relatively simple composition that forms the basis for the more advanced search methods presented in Sections 6.4 and 6.6 (as well as the approximate search methods presented in Chapter 7).

The simplest way to compose agents' individual feasible influence generations is to construct a search tree wherein each node represents a partially-specified setting of agents' influences. The root node of the tree represents a completely unspecified influence DBN. At the next level down, each node is assigned a particular setting for just one agent's *outgoing influences*. At the next level, each node is assigned a particular setting for each of two agents' outgoing influences, and so on all the way down to the leaf level, where leaf nodes are assigned a complete setting of all agents' influences. In essence, we have divided the parameters of the influence DBN according to which agent controls each, placing one agent's influence generation at each level of the tree.

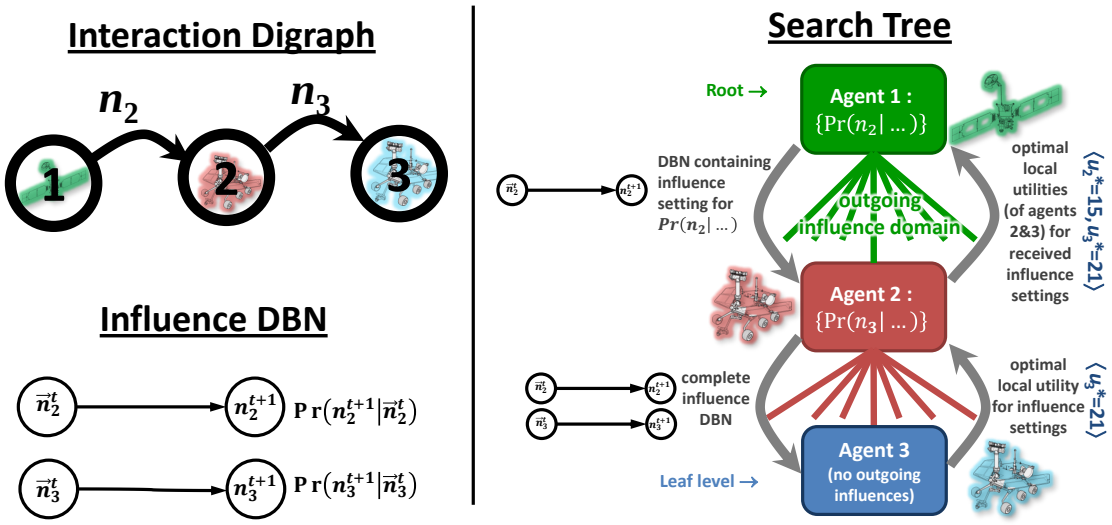


Figure 6.1: One path through the influence search tree.

Depth-first OIS searches the space of feasible influence settings one by one, for each traversing a path from root to leaf. As shown in Figure 6.1, a path consists of a combination of agents' outgoing influence settings, each of which I will refer to as an *outgoing influence point*. The optimal (combined) influence point corresponds to the path leading through the optimal combination of agents' outgoing influence points.

6.3.1 Structure of Search Tree

In the event that the interaction digraph contains no directed cycles[4], we can define a natural ordering over the agents' generation problems, thereby placing one agent at each level of the search tree. Figure 6.1 shows a simple three-agent example problem with an acyclic interaction digraph, the corresponding encoding of the influence DBN, and the resulting structure of the search tree.

At the root of the search tree, influences are considered that are independent of all other influences. At lower depths, feasible influence settings are generated by incorporating any higher-up influence settings on which they depend. This property is guaranteed for any total ordering of agents that maintains the partial order of the acyclic interaction digraph. Given one such ordering, the agents can use the pseudo-code presented in Algorithm 6.1 to perform a depth-first search of the influence space.

[4]I relax this restriction in Section 6.4.

Algorithm 6.1 Depth-First Optimal Influence-Space Search

DF-OIS($i, ordering, DBN$)

...At Leaves, simply compute best response...

1: **if** $i = \text{LastAgent}(ordering)$ **then**
2: $POMDP_i \leftarrow \text{BuildBestResponseModel}(DBN)$
3: $\langle localVal, \pi_i \rangle \leftarrow \text{Evaluate}(POMDP_i)$
4: **return** $\langle localVal, DBN \rangle$
5: **end if**
6: $nextAgent \leftarrow \text{NextAgent}(i, ordering)$

...Enumerate feasible outgoing influence points...

7: $POMDP_i \leftarrow \text{BuildBestResponseModel}(DBN)$
8: $I \leftarrow \text{GenerateFeasibleInfluences}(POMDP_i)$

...Branch for each outgoing influence point...

9: $bestJointVal \leftarrow -\infty$
10: $bestDBN \leftarrow nil$
11: **for each** $\langle influence_i, localVal \rangle \in I$ **do**
12: $DBN_i \leftarrow \text{CopyAndAssign}(DBN, influence_i)$

...Pass Influence Settings Down...

13: $\langle descendantsVals, DBN_{child} \rangle \leftarrow$ DF-OIS($j, ordering, DBN_i$)
14: $jointVal \leftarrow localVal + descendantsVals$
15: **if** $jointVal > bestJointVal$ **then**
16: $bestJointVal \leftarrow jointVal$
17: $bestDBN \leftarrow DBN_{child}$
18: **end if**
19: **end for**

...Pass Highest-Valued Influence Up...

20: **return** $\langle bestJointVal, bestDBN \rangle$

The search begins with the call DF-OIS($root, ordering, nil$), prompting the root agent to build its (independent) local POMDP (line 1) and to generate all of the feasible settings of its outgoing influence parameters (line 8), each in the form of a partially-specified DBN. The root agent creates a branch for each feasible outgoing influence setting, as computed using Algorithm 5.2 from Section 5.6, passing down the setting (in line 13). Each branching operation is a recursive call to DF-OIS that prompts the next agent to construct a local POMDP in response to its ancestors' influence settings (using the best response model I presented in Chapter 4), compute its feasible influences, and pass those on to the next agent.

At the root of the tree, the influence DBN starts out as completely unspecified and is gradually filled in as it travels down the tree, at each subsequent level accumulating another agent's influence settings. The agent at the leaf level of the tree does not influence others, so simply computes a best response to all of the influence settings

of its ancestors, for each passing up its best-response utility value (lines 2–4). At each intermediate node, the respective agent evaluates each of its outgoing influence settings that it passed down by taking the sum[5] of the combined utility value passed up from its descendants (denoted $descendantsVals$) and its local utility value (line 14). In this manner, from the leaves to the root, the best outgoing influence setting is selected (lines 15–18) at each level of the tree, accounting for both local cost (or reward) as well as descendant reward (or cost). When the search completes, the result is an optimal influence-space point: a DBN that encodes the feasible influence settings that achieve the optimal team value. As a post-processing step (not shown in Algorithm 6.1), the agents compute their optimal joint policy by each computing a best response to the optimal influence DBN returned by the search.

6.3.2 Enumerating Feasible Influences

At each non-leaf node of the tree, the respective agent calls GenerateFeasible-Influences() to generate its feasible outgoing influence points. One such influence generation scheme is presented in Section 5.6 comprising a series of mixed-integer linear programs that the agent uses to find each feasible combination of settings of its outgoing influence parameters. Recall that the agent's outgoing influence parameters specify the probabilistic transitions of the agent's locally-controlled features that affect other agents. Interestingly, the MILP-driven generation of an individual agent's influence settings takes the same form as OIS's generation of combinations of agents' feasible influences. Just like OIS's generation, the MILP-driven generation of outgoing influences (Algorithm 5.2) constructs a tree, in this case placing an individual influence parameter at each level of the tree. As such, the branches in OIS's DFS tree comprise the leaves of each agent's MILP-driven search tree.

The operation of OIS does not depend upon my constrained linear programming methodology, however. Any alternative generation scheme would suffice. For instance, the agent could simply enumerate all of its local policies, for each computing the implied settings of conditional probabilities that the influence DBN requires, and then manually partition the agent's local policy space into *impact-equivalence classes* (Def. 3.45) whose influence parameter settings are equivalent.

[5]Algorithm 6.1, as presented, relies on the property that the joint utility is a *summation* of local utilities (Thm. 3.8). In recent work (Witwicki & Durfee, 2010), I have published a slightly more general version of DF-OIS that accommodates arbitrary monotonic value compositions functions. It does so by passing values down the tree as well as up the tree, so that at every intermediate node, the full joint value is straightforwardly evaluable through invocation of the composition function.

6.3.3 Incorporating Ancestors' Influences

At each node below the root, with a call to BUILDBESTRESPONSEMODEL(), agent i incorporates the outgoing influences of its ancestors that have been communicated with the DBN object passed into DF-OIS(). However, just as an ancestor agent's outgoing influences make up a subset (and not necessarily the whole) of the DBN parameters, agent i's incoming influence parameters sufficient for its best response reasoning also make up a subset (and not necessarily the whole) of the DBN parameters. In other words, some of the information contained within the communicated DBN is inessential (and unusable) to agent i. In this case, agent i may use marginalization to remove any unneeded variables (specifically, those that it cannot observe) from the conditional probabilities represented by the DBN parameters.

Example 6.6. For instance, consider the interaction digraph, and corresponding influence DBN shown in Figure 6.2. Here, agent 7 models two nonlocal features, one (n_{7a}) influenced by agent 1 and the other (n_{7b}) influenced by agent 6. Additionally, agent 6 models a nonlocal feature n_6 influenced by agent 1. The undirected digraph cycle between agents 1, 6, and 7 implies a conditional dependence relationship between n_{7a} and n_{7b} by way of n_6. Consequently, agent 1 encodes the influence-dependent distribution $\Gamma_{\pi_i}(n_{7a}, n_6) = Pr\left(n_{7a}^{t+1}, n_6^{t+1} | \vec{n}_{7a}^{\,t}, \vec{n}_6^{\,t}\right)$, generating settings to all of the respective DBN parameters as it enumerates its feasible influence points. Agent 6 encodes a history-dependent distribution $\Gamma_{\pi_i}(n_{7b}) = Pr\left(n_{7b}^{t+1} | \vec{n}_6^{\,t}, \vec{n}_{7b}^{\,t}\right)$ that is conditioned on the history of feature n_6. Altogether, the two agents' influences make up an influence DBN that connects the variables associated with all three nonlocal features (as illustrated in Figure 6.2).

The influence DBN that agent 1 passes down the tree to agent 6 contains parameters of the form $Pr\left(n_{7a}^{t+1}, n_6^{t+1} | \vec{n}_{7a}^{\,t}, \vec{n}_6^{\,t}\right)$. However, note that according to the TD-POMDP local state for this problem, agent 6 does not model nor does it observe feature n_{7a}. The only information that agent 6 needs (to reason optimally about its own behavior and its own outgoing influence) is $Pr\left(n_6^{t+1} | \vec{n}_6^{\,t}\right)$. By Theorem 4.18, in constructing its best response model, agent 6 can safely marginalize out $\vec{n}_{7a}^{\,t+1}$ from the joint distribution. Similarly, agent 7 can marginalize out $\{n_6^0, ..., n_6^T\}$ from the complete DBN, leaving only parameters of the form $Pr\left(n_{7a}^{t+1}, n_{7b}^{t+1} | \vec{n}_{7a}^{\,t}, \vec{n}_{7b}^{\,t}\right)$.

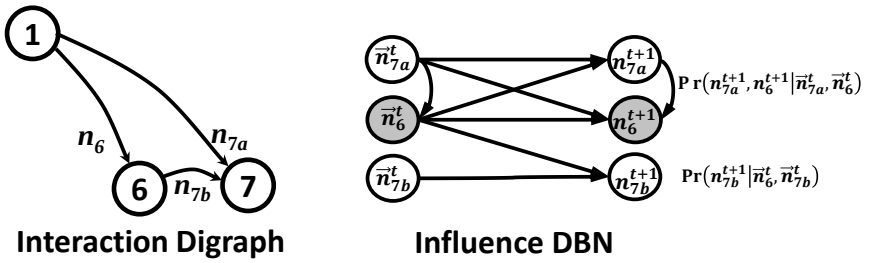


Figure 6.2: Example of marginalization of unneeded DBN parameters by Agent 7.

Upon extracting the necessary conditional probabilities from the influence DBN, agent i injects these into the transition model of its best-response POMDP (developed in Section 4.2). Agent i uses this model, denoted $POMDP_i$, to generate its feasible outgoing influence settings, thereby accounting for its ancestors' specified influence settings.

6.4 Interaction Digraph Cycles

As I have presented it thus far, Algorithm 6.1 requires an acyclic agent interaction digraph. The difficulty with cyclic graphs lies in the fact that there no longer exists a total ordering over the agents with the property that each agent can compute its best response and reason about its outgoing influence settings independently of agents that appear later on in the ordering.

In this section, I describe one high-level strategy for searching the influence space when there are cycles. The basic idea is that, by examining the lower-level structure of agents' influence parameters, we can transform the cyclic interaction digraph into an equivalent form that does not contain cycles, so as to apply (essentially) the same depth-first search techniques without loss of generality. This strategy obviates having to face the added complexity of other common approaches to coping with graphical model cycles, such as cycle cutset (Dechter, 2003).

Example 6.7. Consider the interaction digraph and corresponding influence DBN shown in Figure 6.3. In this problem, agent 1 influences agent 2 through nonlocal feature n_2 and agent 2 influences agent 1 through nonlocal feature n_1. This means that agent 1 cannot reason about its feasible outgoing influence settings without

accounting for 2's influence, which 2 cannot reason without accounting for 1's outgoing influence settings. Clearly, neither agent is able to generate its feasible influences independently of the other. The decomposition of influence generation by agent (developed in Section 6.3) will not work for this problem.

Fortunately, by digging deeper into the structure of the TD-POMDP model, we find that there is an inherent acyclicity at the nonlocal feature value level. Regardless of whether of not a problem's interaction digraph contain cycles, the influence DBN cannot contain cycles. No DBN can. In the case of the TD-POMDP, the impossibility of cyclic dependence among individual influence variables stems from the non-concurrency of agents' interaction effects (described in Section 3.4.3.1). Intuitively, agent 1's actions can affect agent 2's at time step t, but agent 2's concurrent actions cannot interfere. Agent 2 cannot use the consequences of agent 1's influence to influence agent 1 back until time step $t+1$. In other words, agent 2's outgoing influence may be dependent on past outgoing influences of agent 1, but is conditionally independent of concurrent outgoing influences of agent 1.

This insight leads us to a reformulation of the search process, wherein at each level of the tree, an agent reasons about a subset of its influence parameters. In essence, we can define an ordering over individual influence parameters with the necessary property that a parameter's values are conditionally independent of the values of parameters that appear later in the ordering conditioned on the values of parameters appearing earlier in the ordering. Any ordering consistent with the ordering of the variables in the influence DBN suffices.

Example 6.7 (continued). As illustrated by the search tree in Figure 6.3, instead of ordering the influence parameters by agent, we can order the influence parameters by the time indices of their nonlocal feature variables, such that agents 1 and 2 consider influence probabilities pertaining to earlier times before those pertaining to later times. The result is a depth-first search that iterates back and fourth between agent 1's generation of feasible parameter values and agent 2's generation of feasible influence values.

Just as before, feasible influence parameter settings are passed down the tree and values are passed up the tree, so as to select the optimal parameter settings at each

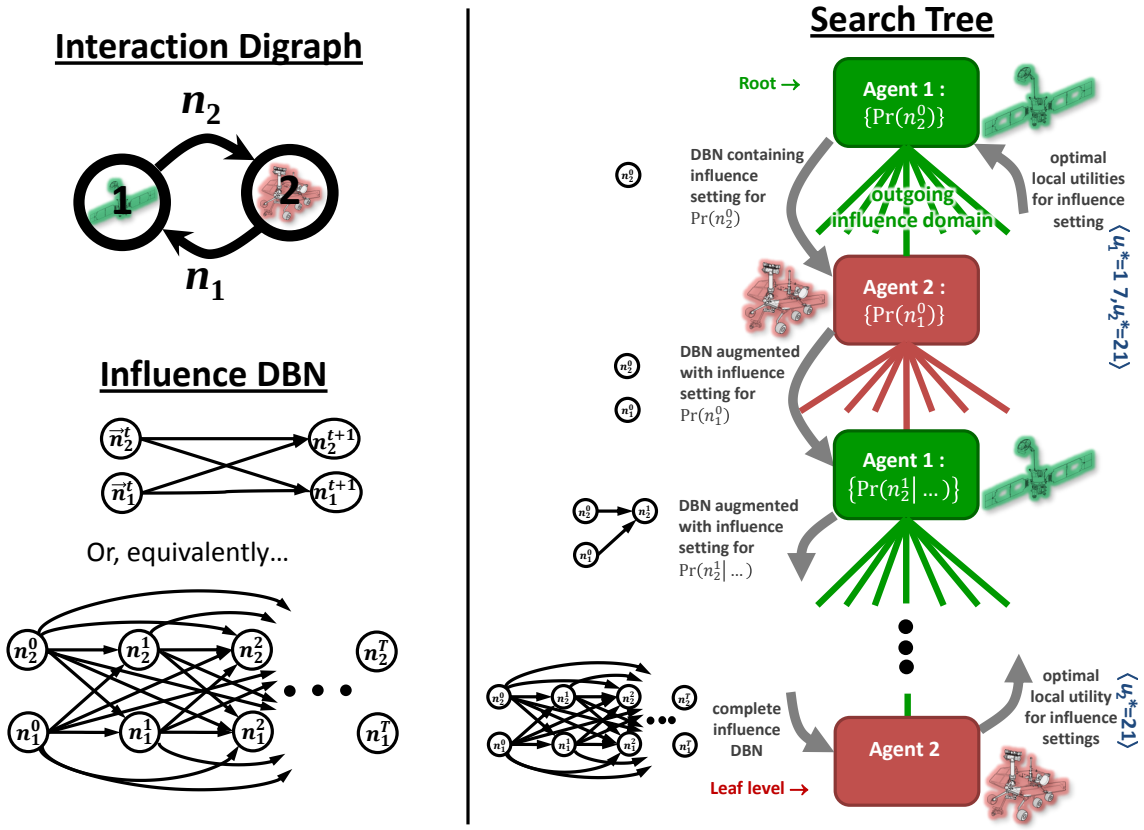


Figure 6.3: Example of Influence-Space Search on a cyclic interaction Digraph.

level. However, in the cyclic case, agents do not evaluate their influences at every level of the tree. Instead each agent computes an exact best-response value after all of its outgoing influence parameters have been assigned. In Example 6.7, evaluation occurs at the lowest two levels of the tree.

Another difference is that agents build best response models given only partially-specified incoming influences. For instance, at the 3rd level, agent 1 formulates a best response to partial influence $Pr(n_1^0)$. This computation is sound because, in order to generate a feasible setting of its current outgoing influence parameter, $Pr(n_2^1)$, agent 1 requires only a partially-specified best response model and it only needs to consider the possible policy decisions at time 0. $Pr(n_2^1)$ is independent of all of the unspecified influences and future actions.

6.5 Empirical Results

Recall that the empirical results presented in Chapter 4 uncovered trends in the size of the influence space and the degree of influence as they relate to various problem characteristics. I now evaluate the extent to which these trends translate to computational advantages (and disadvantages) for OIS over other optimal solution algorithms that do not employ influence-based abstraction. In particular, I test the general hypothesis that OIS outperforms its competitors on problems that are weakly-coupled. After describing each of four other optimal algorithms and my experimental setup in Section 6.5.1, the subsections that follow (6.5.2–6.5.5) are each devoted to comparing OIS against one of the four other algorithms. In each instance, I compare runtime systematically across a space of random problems so as to expose the strengths and weakness of OIS in relation to my characterization of influence-space size and degree of influence from Section 4.6. I conclude my analysis with a summary and discussion of the results in Section 6.5.7.

6.5.1 Experimental Setup

Here, I describe the other optimal algorithms and the space of problems considered in this analysis.

6.5.1.1 Other Algorithms

I compare the runtime performance of OIS with four other algorithms designed to compute optimal solutions to restricted flavors of transition-dependent Dec-POMDPs. The first two are straw men of my own design: decoupled joint policy search (without influence abstraction) and centralized joint policy formulation (using a MILP adapted from the single-agent MILP methodology that I presented in Chapter 5). The third and fourth are state-of-the-art algorithms whose development has been published by others within the last three years.

Note that both of the state-of-the-art algorithms are designed for solving specialized flavors of transition-dependent Dec-POMDP problems (both of which are less general than the TD-POMDP). Moreover, the implementation of OIS has not been given any advantages for exploiting specialized structure beyond that which the other implementations exploit.

Policy-Space Search. The first algorithm is an implementation of OIS (described in Section 6.3) that has been stripped of its policy abstraction. Instead of searching

for the optimal influence, agents search directly in the policy space. At each step, they exchange local policies instead of influence settings, but are able to exploit the interaction digraph structure in the same way that OIS does, and use the same best-response models used by OIS. In essence, comparing OIS against this method illustrates the advantages of policy abstraction.

Centralized MILP approach. I have also implemented a method that applies my MILP methodology (developed in Chapter 5) to a centralized decision model with a joint transition matrix, joint state space, and joint action space. The objective of the MILP is to maximize the joint utility subject to the constraints that each agent must base its decisions solely on its own observations (using a variation of the same technique I developed in Section 5.2.3). As such, comparing OIS against this method illustrates the advantages of decoupling the joint policy formulation (on TD-POMDP problems, for which such a decomposition is efficient).

SPIDER. SPIDER (Varakantham et al., 2007) is a decoupled best-response search method that performs a policy-space search but employs its own pruning to reduce the search space. It was originally developed for solving transition and observation independent problems, and was recently extended for application to a specialized class of two-agent transition-dependent problems involving interdependent tasks (Marecki & Tambe, 2009). As of yet, it has not been extended to solve any flavors of transition-dependent problems containing more than two agents. In the experiments that follow, I use the implementation of SPIDER graciously provided by its authors.

Separable Bilinear Programming. The last algorithm, which I will denote SBP (Mostafa & Lesser, 2009), was designed for solving EDI-CR problems (as contrasted with TD-POMDPs in Section 3.4.1.5). SBP frames the joint policy formulation problem as a separable bilinear program (Petrik & Zilberstein, 2009). In the experiments that follow, I use the implementation of SBP graciously provided by its authors. Like the centralized MILP approach (described above), SBP is a centralized algorithm. However, unlike the centralized MILP, SBP exploits the factored structure of agents' subproblems.

6.5.1.2 Random Problem Generation

For the moment (and up until Section 6.5.6), I restrict the focus of my analysis to two-agent problems that are generated according to the same parameterization

described in Section 4.6. For each of the problems from my original testbed, I add a second agent, agent 2, and for each of agent 1's nonlocally affecting tasks ($task_{1x}$), one of agent 2's tasks ($task_{2y}$) is randomly selected (without replacement) as being nonlocally-affected. I restrict that every nonlocal effect $e_{1x,2y}$ takes the form of an *enablement*, such that agent 1's completion of $task_{1x}$ allows agent 2 to execute $task_{2y}$ without achieving an automatic failure outcome. Agent 2's tasks (including its positive-quality outcomes) are instantiated using the same parameters as agent 1's. In this analysis, I restrict consideration to problems with a single influencing agent and a single influenced agent. I impose both of these restrictions (*enablement* effects and acyclic interaction digraphs) for compatibility with the two state-of-the-art algorithms ("SPIDER" and "SBP") against which I am comparing OIS.

Due to the fact that the implementations of "SPIDER" and "SBP" were tailored to problem domains with differing assumptions, I could not run them on exactly the same set of problems.[6] Instead, the results presented in Sections 6.5.4 and 6.5.5 respectively compare OIS to SPIDER and OIS to SBP on separate sets of problems, each generated via slight alteration of my original problem generation scheme.[7]

6.5.2 Comparison with Policy-Space Search

I begin by comparing OIS with policy-space search. In a sense, this comparison is the purest evaluation of influence-based abstraction because both algorithms behave identically except that OIS abstracts each agent's local policy space and policy-space search does not. Based on my empirical findings regarding influence space sizes and policy space sizes (presented in Section 4.6), I offer the following hypotheses. First, I posit that policy space search will be limited in its tractability to problems wherein each agent's local decision model is small. Second, I hypothesize that out of the problems where policy-space search is tractable, it will outperform OIS only when the degree of influence is high (indicating that there are almost as many feasible influence

[6] My implementation of OIS, however, is compatible with either set of assumptions.

[7] SPIDER required that agents not observe their nonlocal features and that tasks were constrained via a *latest start time* instead of a *latest finish time*. In my generation of problems for Section 6.5.4, I redefined a task's window size parameterization accordingly and treated each *enablement* feature as a latent variable in the state of the affected agent. In addition to the same partial observability assumption as SPIDER, SBP required that each task have only two durations, that task windows be specified with an earliest start time but not a latest finish time, and that agents are not allowed to "wait" between their executions of tasks. As such, for the set of problems used in Section 6.5.5, I generated tasks whose latest finish times were constrained to be time T but whose earliest start time was selected according to parameters $localWindowSize$ and $NLATWindow$ (whose semantics were introduced in Section 4.6.1.3 and summarized in Table 4.1) and I removed the "wait" action from agents' decision models.

points as there are policies). In this case, the overhead of finding each unique influence point, by abstracting it from a policy, should outweigh the benefits of the reduced search space size.

As in the empirical analysis from Chapter 4, I have systematically generated problems over the entire space of parameter settings. Again, in the interest of space, here I present select results that illustrate the trade-offs that I have observed across the entire space of parameter settings. As a metric for tractability, I allocated each method at most 10 minutes of computation time per problem. For any given setting of parameters, if any problem from that setting was not solved within 10 minutes, the point corresponding to that average runtime measurement is omitted from the plotted results.

For this experiment, the empirical evidence corroborates both of my hypotheses. In general, policy-space search was able to solve problems wherein the local policy space size of the influencing agent contained no more than 10,000 policies. Although this may sound impressive, note that the policy space grows exponentially with the state and action spaces and the time horizon, and recall from my earlier analysis (Sec. 4.6) that problems with local policy space sizes in excess of 10^8 were not uncommon. Figure 6.4 plots the policy space size, degree of influence, and runtime of both OIS and policy-space search as a function of increasing problem time horizon for three different settings of the remaining problem parameters.

For each individual plot, from left to right, the increase in time horizon results in an increase in average policy space size and a decrease in the degree of influence. Further, from top to bottom in Figure 6.4, the three cases (A, B, and C) represent three gradations of increasingly-large local problem size. As illustrated, for very small problems, policy-space search is faster than influence-space search due to the overhead of OIS's feasible influence generation. However, except in very small problems such as in case A, as time horizons grow longer, the decreasing degree of influence is accompanied by an increase in the runtime of policy-space search such that it surpasses that of OIS. As problems become larger, the additional overhead of OIS is far outweighed by the growing policy space size. As the degree of influence falls lower and lower, the gap widens. For instance, in case C, when T=4, policy-space search takes two orders of magnitude longer than OIS, and for T=5, cannot compute solutions to each problem in 10 minutes whereas the average computation time taken by OIS just one second.

As a testament to my weak coupling theory from Section 3.5.2, the trends observed in the computational advantages of OIS (over policy space search) are a direct

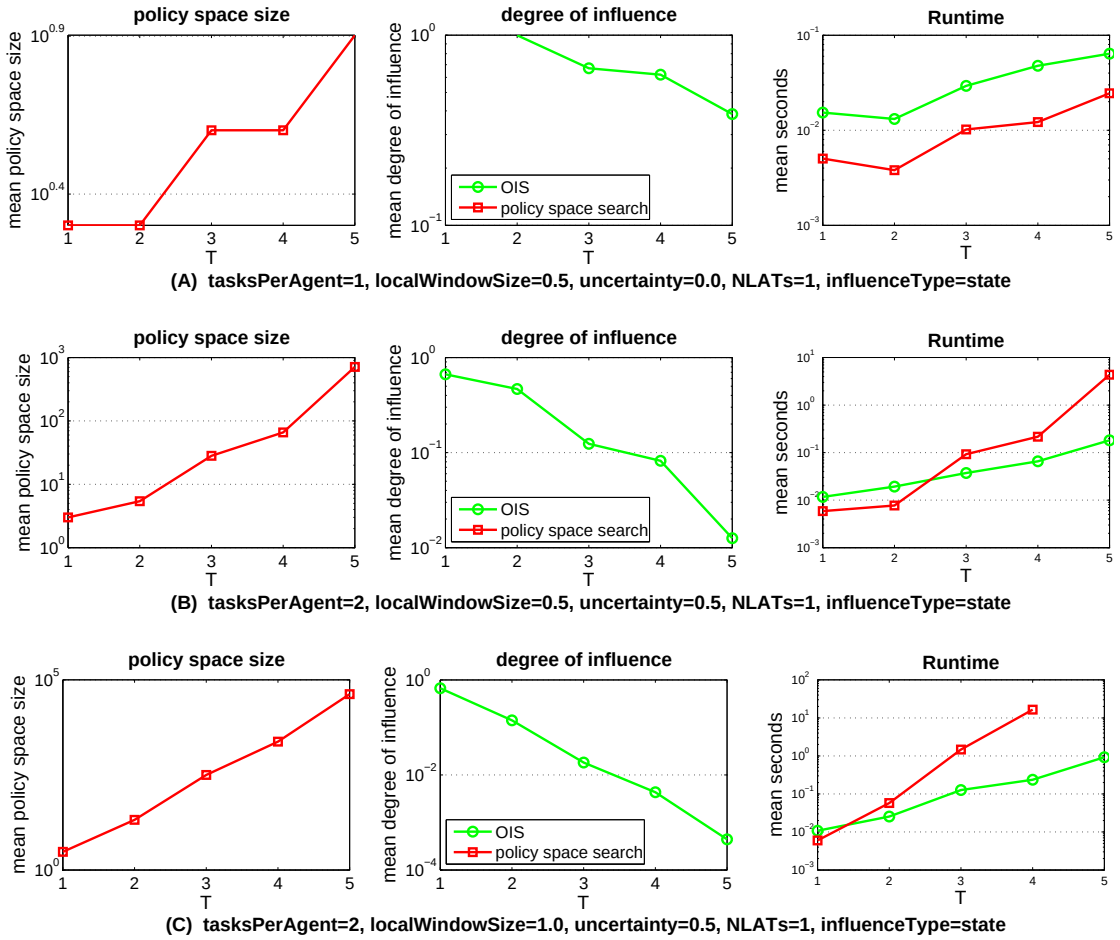


Figure 6.4: OIS vs. Policy Space Search : growing problem size

translation of the trends observed in the degree of influence. The same translation of trends can also be seen when the size of the agent's nonlocally-affecting task window (Figure 6.5E) and the earliest start time of the nonlocally-affecting task window (Figures 6.5F and 6.5G) are varied. These last three plots, though seemingly complex, depict the same empirical trends evidenced and discussed in Section 4.6.2.5.[8]

[8]Figure 6.5E confirms that, as the nonlocally-affecting task's window increases, although the policy space increases, the degree of influence decreases; as a result, we observe that the computation time of naïve policy-space search grows more steeply than that of influence-space search. Figure 6.5F and 6.5G confirm that the relationship of policy space size to the earliest start time ($NLAT\_est$) of the nonlocally-affecting task depends heavily on the sizes of local task windows. When local task windows are small (Fig. 6.5F), the policy space size grows very slightly with the nonlocally-affecting task window, causing an increase in the computation time of policy-space search when compared with influence-space search. When local task windows are large (Fig. 6.5G), we see the opposite trend: the policy space size decreases significantly, causing a decrease in the computation time of policy-space search while the computation of influence-space search remains relatively flat.

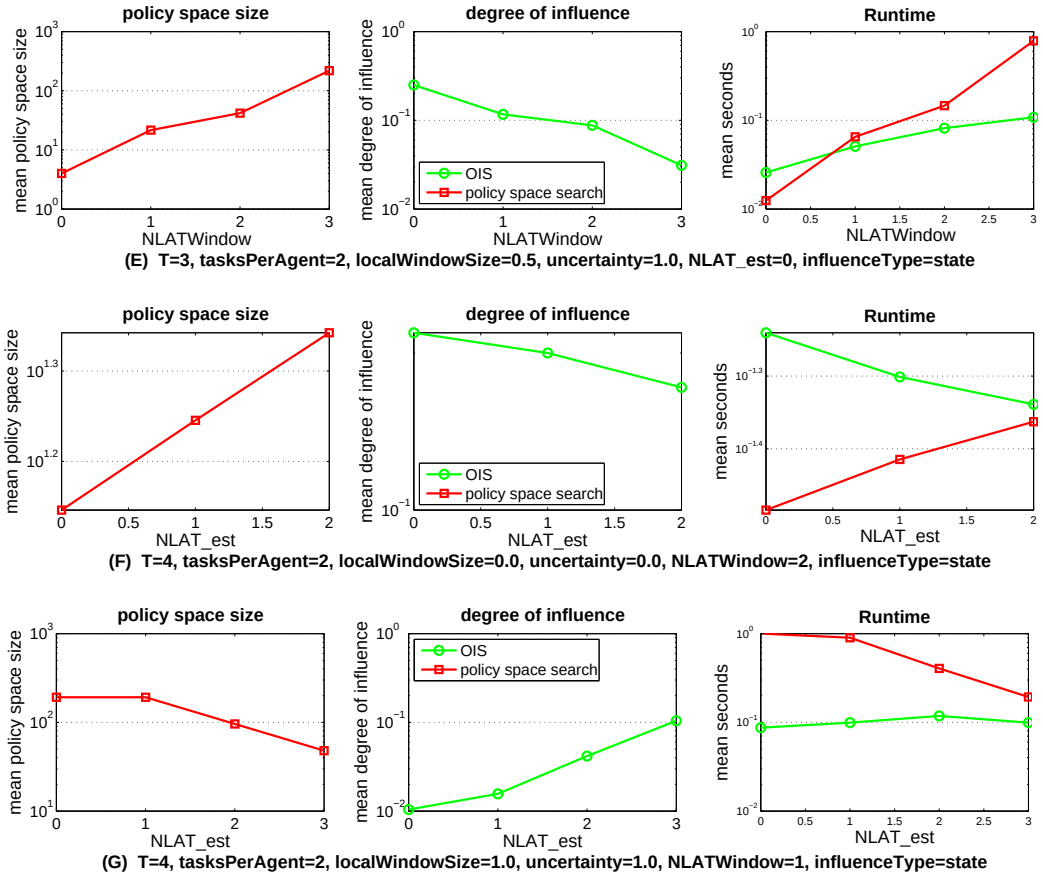


Figure 6.5: OIS vs. Policy Space Search : window of interaction

6.5.3 Comparison with the Centralized MILP Approach

Although the degree of influence appears to be strongly correlated with the computational requirements of OIS relative to policy-space search, it does not necessarily characterize OIS's computation relative to other solution algorithms. Intuitively, the centralized MILP approach does not search the policy space directly; instead, it searches through agents' joint occupation measures. Moreover, it is apparent from my empirical observations (not shown) that the size of the influencing agent's policy space is not a strong predictor of the computation time of the MILP approach.

Instead, the computation of the MILP approach is dependent on the size of the program (i.e. the number of variables and constraints), which is the product of the number of world states and the number of joint actions. This alternative dependence is advantageous for problems with few agents, few tasks, and small state spaces. As my empirical results confirm, the centralized approach computes optimal solutions faster than does OIS for a large portion of the problems in my testbed. Figure 6.6

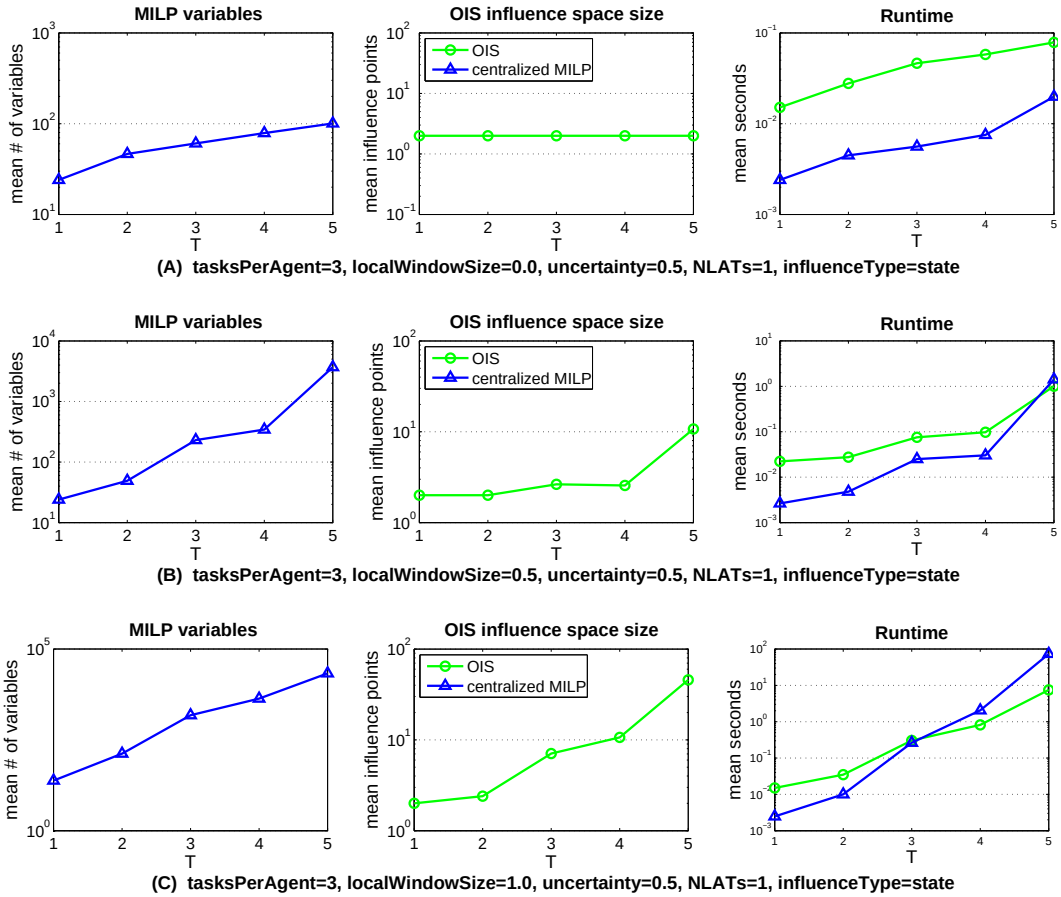


Figure 6.6: OIS vs. Centralized MILP : scaling

plots MILP variables, influence-space size, and computation time of both methods for three gradations of increasingly-large problem size (labeled as A, B, and C).

Moving from top-to-bottom in Figure 6.6, case A has $localWindowsize = 0.0$, case B has $localWindowsize = 0.5$, and case C has $localWindowsize = 1.0$. As shown, increasing the time horizon in each of these cases increases the size of the centralized MILP, but the rate of increase is heavily dependent on the local window size; as a consequence, so is the rate of increase of the centralized MILP's computation time. This brings us to the disadvantage of the centralized MILP's dependence on the joint state and joint actions. As weakly-coupled agents' local problem sizes increase, the joint state space increases significantly, ultimately yielding poor scalability of the MILP (as well as any other approach that works directly with the flat joint state and action representation). Notice that in all three cases, the centralized method's runtime grows more steeply than did that of OIS. This same trend was observed across the board and when varying other attributes relating to local problem size such as number

of tasks and uncertainty.

OIS's superior scalability is not due exclusively to the fact that it works in the policy space, or even that it decomposes the joint policy formulation. Both of these traits are present in the policy-space search algorithm (Sec. 6.5.2), whose scalability was inferior to that of the centralized MILP. OIS's scalability comes from its abstraction. For problems with large local model sizes but highly-constrained influences, OIS gains significant advantage over centralized methods such as the centralized MILP. This advantage is evident in Figure 6.7, which shows the effect of varying the windows size of an influencing agent's nonlocally-affecting task.

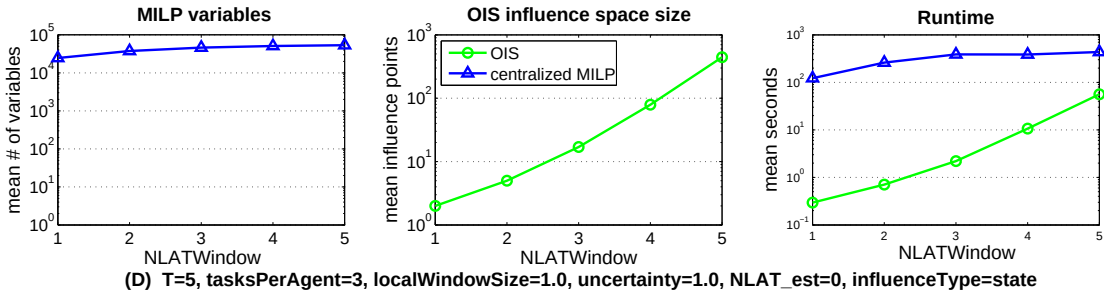


Figure 6.7: OIS vs. Centralized MILP : NLAT Window Size

As shown, when the influence is highly-constrained such that the interaction can only occur during a restricted interval, OIS gains significant advantage over the centralized MILP by searching through a greatly-reduced search space. As the size of the nonlocally-affecting task window increases, so does the computation of OIS. Moreover, I observed that, for all settings involving problems hard enough such that either approach took longer than a second on average, the same qualitative trend shown in Figure 6.7 occurred, and in none of these instances did the average computation time of OIS surpass that of the centralized MILP at the window's maximum value.

Notice that, in contrast to the comparison of OIS and policy-space search in Section 6.5.2, *degree of influence* does not correlate with the difference in computation times among OIS and the centralized MILP. My empirical results from Section 4.6.2.5 suggest that as the nonlocally affecting task window increases, the degree of influence decreases, and that influence-based policy abstraction should be more effective. Although OIS may be increasingly-effective compared to policy-space search, here we see that the gap in computation time between OIS and the centralized MILP narrows as the nonlocally-affecting task window is increased, suggesting that OIS is losing its advantage even though the degree of influence is decreasing. Intuitively, while *degree of influence* is computed from the policy space size, the centralized MILP does not

search the policy space exhaustively, making this method insensitive to the *degree of influence*.

6.5.4 Comparison with SPIDER

Next, let us turn to an algorithm that searches through the policy space, though not as naïvely as the policy-space search method from Section 6.5.2. SPIDER does not exhaustively evaluate each agent's policies but instead performs pruning using heuristic evaluations of partially-specified policies (Varakantham et al., 2007). Its pruning makes it a much stronger competitor, scaling well beyond the reach of the naïve policy-space search method in Section 6.5.2. In certain instances in my testbed, its average computation time scaled more gracefully than did that of OIS (e.g., Figure 6.8A), and in other cases (e.g., Figure 6.8B) not as well.

Figure 6.8, which plots only computation times, scales the problem time horizon along the x-axis, varying *uncertainty* from left to right, and varying $localWindowSize$ from top to bottom. Notice that although SPIDER outperforms OIS in some cases, it starts out with a higher computation time, presumably due to the computational overhead of SPIDER's pruning. As expected, both methods are affected by uncertainty as well as by local window size.[9] However, the average runtime of OIS grows more steeply (on an exponential scale) than does SPIDER's when $localWindowSize = 0.5$ (case A) and less steeply than SPIDER's when $localWindowSize = 1.0$, affirming the hypothesis that influence-based abstraction is most effective in comparison with other approaches when agents' local decision models are more complex.

Evidently, SPIDER is able to significantly reduce the size of its search space under some circumstances. However, the pruning that SPIDER employs does not seem to exploit the same structure as influence-based abstraction. Figure 6.9 shows the effects of holding local problem size still and increasing the size of the influencing agent's nonlocally-affecting task window. Here, regardless of the local tasks' window sizes, SPIDER exhibits relatively little improvement for smaller nonlocally-affecting task window sizes. OIS, on the other hand, is exponentially faster for smaller nonlocally-affecting task window sizes (just as we observed in Figure 6.7). When nonlocally-affecting task windows are larger, and influences are less constrained, OIS may be at a disadvantage to algorithms such as SPIDER. The good news is that this disadvantage appears to dwindle as agents' local problem sizes become larger.

[9] Although these problems were too large for naïve policy-space search, the trends in runtime growth of SPIDER due to uncertainty and local window size are consistent with those that I observed in the running times of the naïve policy-space search on the problem set from Section 6.5.2

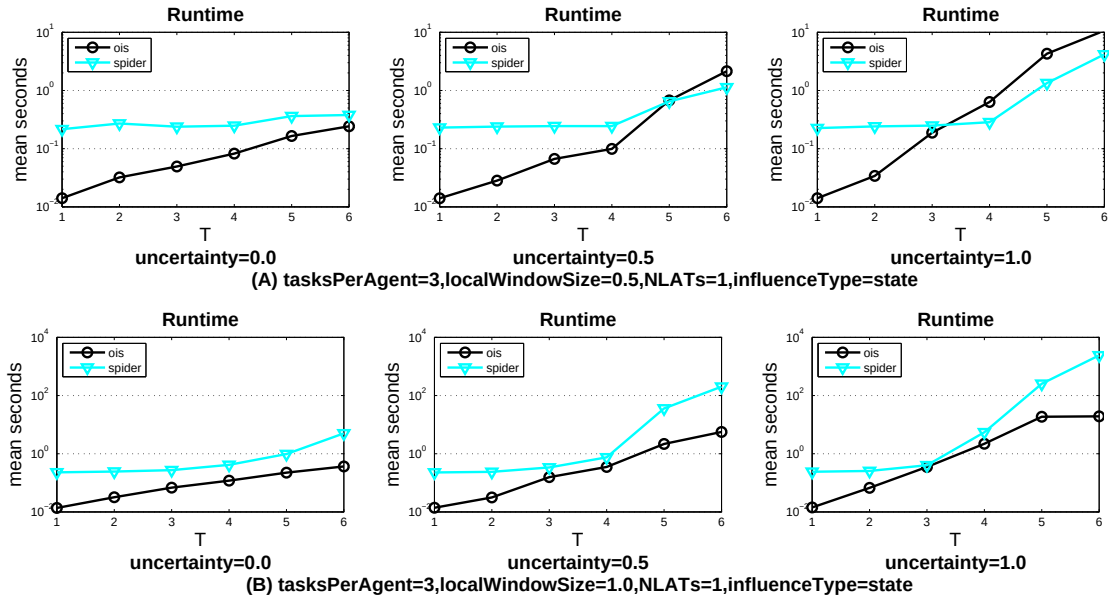


Figure 6.8: OIS vs. SPIDER : scaling local problem size

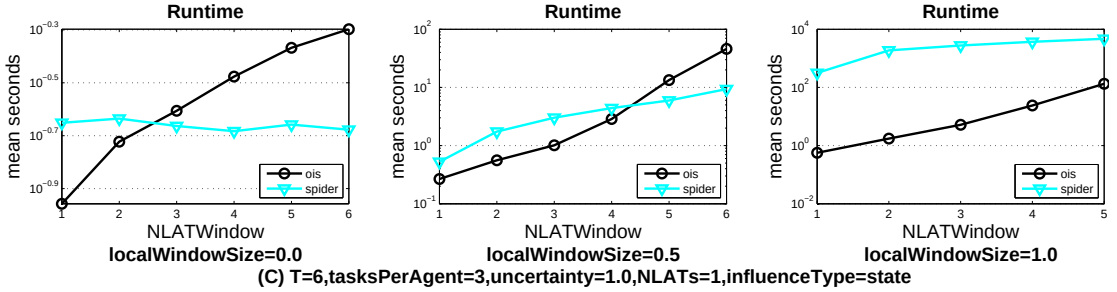


Figure 6.9: OIS vs. SPIDER : NLAT Window Size

6.5.5 Comparison with SBP

The last algorithm that I consider in this analysis is separable bilinear programming (SBP), which computes the agents' joint policy using a centralized representation that models the agents' joint behavior in such a way that it can exploit their largely-independent factored transition structure (Mostafa & Lesser, 2009). The question then becomes whether or not the computational advantages of SBP's structural exploitation outweigh those of OIS's influence-based abstraction. For weakly-coupled problems wherein agents influences are constrained, my empirical results suggest the opposite.

Across all parameter settings, I observed a qualitatively-identical trend: OIS was orders of magnitude faster than SBP when the nonlocally-affecting task's window was highly-constrained, but approached SBP's runtime as the nonlocally-affecting task

window was expanded to its maximum value. Figure 6.10 illustrates the trend for one particular setting of parameters.[10] I could not find a single parameter setting for which OIS's computation time (statistically) significantly exceeded that of SBP when $NLATs$ reached its maximum value.

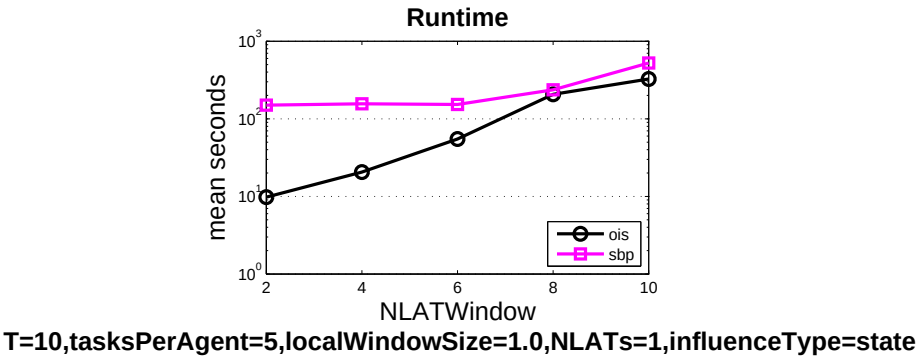


Figure 6.10: OIS vs. SBP : NLAT Window Size

6.5.6 Scaling Beyond Two Agents

The positive results in the preceding subsections indicate the potential of OIS to compute optimal solutions for problems with more weakly-coupled transition-dependent agents than is currently possible with any other solution algorithm. I now provide an initial demonstration of OIS's scalability. Using the same parameterization of agents' local decision problems as in earlier experiments (Table 4.1), I create random problems wherein n agents are connected in a chain of the same form depicted in Figure 6.1 (top-left). Here, agent 1 influences agent 2, who influences agent 3, who influences agent 4, and so on. Solving this problem using OIS entails constructing a search tree as developed in Section 6.3 that generates and evaluates each agent's feasible influences in a depth-first manner.

Figure 6.11 shows the runtime of OIS on a set of 25 random problems per point, wherein the number of agents (n) was varied, and the problem generator parameter

[10]Note that, given the constraints of the developers' implementation of SBP (described in Section 6.5.1.2), my comparison of OIS and SBP required a variation of the test sets used in past experiments that turned out to be significantly different. In particular, for the problems in the experiment, agents could not perform a *wait* action in between task executions, which resulted in variations in the length of the time horizon having little effect on the size of the policy space or on the computation required by OIS. Further, the number of outcomes per task was necessarily two. As such, in order to make problems challenging for OIS, I needed to instead increase the number of tasks per agent.

settings were fixed such that agents have moderately-sized local decision models that remain weakly-coupled (tied together with a single nonlocally-affecting task per agent, with the exception of the agent at the end of the chain). As shown, OIS is able to compute optimal solutions to five-agent problems in a reasonable about of time (10 minutes on average). Although the computation time taken by OIS is exponential in the number of agents, its exponential curve is far less steep than that of the centralized MILP approach, which is able to solve three-agent chains in 10 minutes on average, and uses up all of its allotted 2GB of memory in the process of solving any of the 4-agent problems.

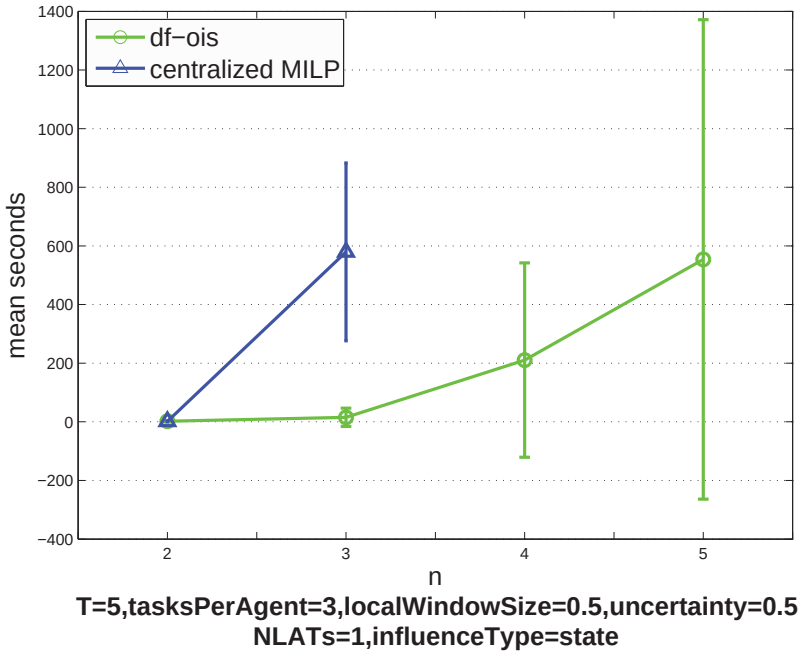


Figure 6.11: Scalability of OIS and Centralized MILP to more than two agents.

This scalability result is significant in that it is the first demonstration of *optimal* solution computation to (relatively-unrestricted) transition-dependent problems containing more than three agents. Not only does it demonstrate the tractability of TD-POMDP problems with five weakly-coupled agents, but it also affirms that influence-based policy abstraction indeed enables scalability beyond the state-or-the-art, surpassing that of any other algorithms' available implementations.

The reader may be left wondering why such a result has not been possible with prior approaches. Towards answering this question, I offer the following intuitions. The four algorithms that I included in my empirical comparison, each of which could be considered contenders for scaling transition-dependent problems, face inherent

obstacles that (as-of-yet) prevent their scalability to teams of > 3 agents:

1. Policy-space search is obligated to perform a number of policy evaluations on the order of $\|\Pi_i\|^{n-1}$, a term which, when $n > 3$, is completely intractable for any sizable value of $\|\Pi_i\|$. Although depth-first OIS also performs a number of best-response calculations that is exponential in the number of agents, it exploits a significant reduction in the base of the exponent by abstracting agents' local policies.

2. The centralized MILP approach is weighed down by the exponentially-increasing size of the joint decision model. The addition of each agent corresponds to an exponential increase in the number of joint actions that the MILP considers, not to mention the exponential increase in the state space (resulting from the cross product of weakly-coupled agents' local state spaces). Without significant exploitation of factored structure to keep the joint model compact, any centralized solution method will be inherently limited in its scalability.

3. It is unclear how one would implement SBP on a problem with more than two agents, given that it formulates the problem as a bilinear program.

4. In principle, SPIDER could be scaled to solve problems with more than two agents. In fact, it has already been scaled to problems with more than two *transition-independent* agents Varakantham et al. (2007). However, to handle transition-dependent agent problems, SPIDER needs to address additional issues that do not arise in the transition-independent case. For instance, its generation and pruning of candidate policies must be pursued in a manner that is consistent with the directional topology of agents' influences. I faced these same issues in my design of OIS. Implementation issues aside, although initial results suggest OIS's abstraction engenders a smaller search space than does SPIDER's pruning, it remains an open question whether or not future implementations of SPIDER could accomplish the scalability results achieved by OIS in Figure 6.11.

6.5.7 Summary and Discussion

The results of my empirical comparison may be summarized as follows:

- The tractability of naïve policy-space search is restricted to relatively small two-agent problems.

- The centralized MILP outperforms OIS on two-agent problems whose local decision models are relatively small. However, it scales much more poorly than does OIS as agents' local decision models become more complex, or as the number of agents is increased.

- When applied to my space of test problems, the computational advantages of OIS over naïve policy-space search are well-characterized by my earlier empirical evaluation of influence space size. The same sets of problems that have a low degree of influence also tend to result in a lower computation time of OIS relative to that of policy-space search.

- For algorithms that prune the policy space in their own way (e.g. SPIDER) or instead search the space in a fundamentally different manner (e.g. the centralized MILP), their computation time is less sensitive to the degree of influence. It is thus not surprising that degree of influence has less effect on their computation time than it does on OIS's computation time. As a consequence, the relative computational advantage of OIS is less predictable with respect to these algorithms than with respect to naïve policy space search. In comparison with algorithms other than naïve policy-space search, the constrainedness of agents' influences (as captured by the size of the window of interaction) appears to be a stronger predictor of OIS's computational advantage.

- For weakly-coupled problems wherein agents' influence constrainedness is relatively low, OIS computes optimal solutions orders of magnitude faster than either of the state-of-the-art algorithms (SPIDER and SBP) across the entire space of test problems. Moreover, the advantage of OIS in these circumstances tends to increase as agents' local decision models become more complex.

- I have demonstrated that OIS can compute optimal solutions for transition-dependent problems with more agents than has been achieved by any other algorithms (that are not substantially restricted in their applicability). The computation of optimal solutions for problems with five agents, where the previous state-of-the-art was three, is a significant advance considering that, for all known algorithms, computational complexity is doubly-exponential in the number of agents (by Observation 3.20).

Despite its advantages over existing algorithms, OIS has several key shortcomings. For small problems, in particular those involving tightly-coupled agents, the overhead of

OIS's influence-based abstraction makes it slower than other algorithms. Furthermore, for problems that are not weakly-coupled, OIS loses its ability to reduce the size of the search space, inevitably relinquishing its advantage over alternative solution methods. Although I have shown OIS to scale beyond two agents, it cannot escape the exponential increase in runtime with each new agent, and hence is limited in scalability to just a handful of weakly-coupled agents (using the implementation I have presented).

These empirical results fulfill an important purpose in the overall scheme of this dissertation. Ever since Chapter 1, I have claimed that the focus of this work is the study of transition-dependent problems that, in the presence of weakly-coupled interaction structure, admit efficient and scalable solution algorithms. I began by formalizing the TD-POMDP in Chapter 3, claiming that, although generally intractable, the class of TD-POMDPs contains sets of weakly-coupled problems that can be solved efficiently. My weak coupling theory, developed in Section 3.5, allowed me to be more concrete in this claim: all else being equal, problems that accommodate a lower *degree of influence* should be easier to solve.

In Chapter 4, I introduced influence-based abstraction as a methodology by which to exploit weakly-coupled structure, and, in defense of this claim, empirically characterized those problems for which influence-based abstraction achieves a low degree of influence. However, my claim that weakly-coupled problems enable more efficiently-computed solutions remained unaddressed. It was not until developing a complete solution algorithm, in this chapter, that I was able to analyze the extent to which influence-based abstraction could be used to compute solutions efficiently. The empirical results that I have presented in the preceding subsections do just this. My comparison of the computational cost of OIS with that of other solution algorithms affirms that influence-based abstraction provides a significant advantage over existing methods in the computation of optimal solutions to weakly-coupled transition-dependent problems, in that OIS solves such problems in orders of magnitude less time.

These results do not just affirm the efficacy of influence-based abstraction. Bootstrapping off of my earlier analysis in Chapter 4, they also evaluate the circumstances under which influence-based abstraction gains the most traction in practice. Analogously, they have exposed circumstances under which influence-based abstraction is disadvantageous. By examining the benefits and limitations of OIS, these results may serve as a guide for researchers and developers with which to make informed decisions about the suitability of influence-based abstraction and of optimal influence-space

search to the problems that they address.

Having come full circle, I have now fulfilled the primary contributions set fourth in Section 1.4. In the remainder of this chapter, I develop an extension of OIS for exploiting additional structure to yield more efficient solutions on problems with more than two agents. Then, in the next chapter, I develop extensions for computing approximate solutions. The development and evaluations of these last pieces is more preliminary, and the evaluation less systematic.

6.6 Scaling Beyond a Handful of Agents

The scaling of OIS to five agents in Section 6.5.6 is a significant achievement, but for larger agent teams, DF-OIS hits a wall just as other methods hit a wall at two or three agents. This is to be expected, since the DF-OIS search tree is exponential in the number of agents. However, I claim that in the presence of additional structure, we can overcome this barrier and scale optimal influence space search to indefinitely many agents. The way forward is to exploit structure in the interaction digraph.

The depth-first optimal influence-space search only utilizes the interaction digraph to order agents' influence generations within the search tree. I now develop an extension that exploits structure in the connectivity of the interaction digraph to reduce computation. I begin by describing two situations (in Sections 6.6.1 and 6.6.2) wherein depth-first search performs redundant computation, providing suggestions of how such redundancy might be avoided. Afterwards, in Section 6.6.3, I present a more sophisticated algorithm that applies the *bucket elimination* paradigm (Dechter, 1999) to the problem of optimal influence-space search and demonstrate its scalability.

6.6.1 Independent Ancestors

Consider the interaction digraph shown in Figure 6.12, containing one agent that is influenced by all of its peers. This structure induces a depth-first ordering of the agents according to their interaction digraph indices, such that agents $\{1, ..., n-1\}$ occupy the upper levels of the search tree and agent n occupies the lowest level. To search the space, agent 1 would generate its feasible outgoing influence settings and pass those down to agent 2. For each of agent 1's settings, agent 2 would generate its feasible outgoing influence settings and pass those down to agent 3.

As we proceed down the search tree, the nodes at each level grows exponentially. Agent 3 will receive on the order of $\|\Gamma_i\|^2$ combinations of influences from agents 1 and 2. In turn, agent 3 will call GenerateFeasibleInfluences $\|\Gamma_i\|^2$ times. However,

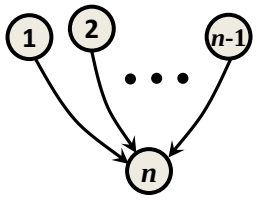


Figure 6.12: An interaction digraph wherein parents are independent.

according to the interaction digraph, agent 3 is uninfluenced by agents 1 and 2. Agent 3's local quality and outgoing influence settings are completely independent of agent 1's and agent 2's influences. Thus, agent 3 is performing an identical computation of feasible influences every time that it considers a different combination of influences from agents 1 and 2. The same is true for all of agents $\{2, ..., n-1\}$. For this problem, the only branching that is required is at the bottom of the tree, wherein agent n considers all feasible combinations of all influence settings of its ancestors.

As a high-level strategy for avoiding this redundancy, consider a refactoring of the search process such that an agent i's *influence generation* problem is explicitly separated from its *influence evaluation* problem. Regardless of the multitude of messages passed down by agent i's peers earlier in the ordering, each of which agent i will later respond to with unique value messages passed up the tree, i does not need to generate outgoing influences for each. Instead, i only needs to generate outgoing influences for each of the unique combinations of incoming influences on which its own decision model depends.

6.6.2 Conditionally Independent Descendants

In Figure 6.13, the digraph topology is such that there is only a single influencing agent (agent 1) who influences all of the remaining agents. Agents $\{2, ..., n\}$ do not share any nonlocal features and so do not influence each others' local transitions or local qualities. Using Definition 3.34, agents $\{2, ..., n\}$ are *decision-independent* of each other *conditioned on* the decisions of agent 1.

For this problem, the depth-first search tree would accurately reflect that agent 1 is the only influencing agent, such that the only branching that occurs is from the root of the search tree. Below the root, agents $\{2, ..., n\}$ generate a single branch per influence setting from agent 1. In this case, the redundancy occurs as values are passed up the tree. Each agent calculates a separate best response for each combination of influence

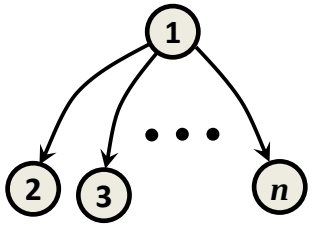


Figure 6.13: An interaction digraph wherein children are conditionally independent

settings involving all $n-1$ of agent 1's nonlocal features $\{n_2, n_3, n_4, ...\}$. If there are on the order of $\|\Gamma_1(n_i)\|$ feasible settings that uniquely specify the transitions of each nonlocal feature, agents $\{n_2, n_3, n_4, ...\}$ will each have to perform $\|\Gamma_1(n_i)\|^{n-1}$ best response calculations.

To avoid this redundancy, consider a restructuring of the search tree into two levels. As before, agent 1 sits at the root node. However, each branch corresponds to a feasible setting corresponding to only one of agent 1's influences $\Gamma_1(n_i)$ (instead of all of its influences $\{\Gamma_1(n_1), \Gamma_1(n_2), ...\}$). Each such branch leads to a leaf node controlled by agent i, such that agent i only needs to respond to $\|\Gamma_1(n_i)\|^{n-1}$ settings. In addition to the savings from computing fewer best responses, agents $\{2, ..., n\}$ will avoid unnecessary exchange of messages over the course of the search. More generally, for any two nodes i and j that are decision-independent conditioned on common ancestors' decisions, agents i and j need not exchange messages over the course of influence space search.

6.6.3 Bucket Elimination for Optimal Influence Search

To take advantage of conditional independence relations among descendants as well as those among ancestors, I now present a more sophisticated reformulation of optimal influence-space search called Bucket Elimination OIS (BE-OIS). It follows the general scheme of Dechter's *bucket elimination* algorithm for constraint optimization (Dechter, 1999). Bucket elimination performs dynamic programming using a well-ordered elimination of variables, associating a *bucket* data structure with each variable to be eliminated. Given a collection of cost functions defined over subsets of variables, and a total order over variables, bucket elimination distributes the cost functions into *buckets* (each associated with a single variable), according to the latest-ordered variable referenced by the cost function. One by one, the algorithm processes each bucket by combining its cost functions by summation, eliminating its variable by

maximization, and passing the reduced cost function down to the next bucket that references any of the remaining variables. Top-down bucket processing is then followed by bottom-up propagation of optimal variable assignments.

Analogous to the elimination of COP variables, here we would like to eliminate influence variables. As such, we will create a bucket that corresponds to each agents' outgoing influences. For simplicity of exposition, let us assume that the agent interaction digraph contains no cycles.[11] We can do so by combining agents' value functions with respect to subsets of influence parameters. As such, the collections of messages passed from one bucket to the next are of the same flavor as the influence evaluation messages passed up the DF-OIS search tree, containing an influence setting and a value. More precisely, in BE-OIS, each message consists of a setting for a subset of influence parameters and the summation of all agents' local values that are influenced by those parameters. However, BE-OIS has the potential to significantly reduce the number of such messages from that of DF-OIS. It does so by employing more sophisticated decomposition of the agents' influence generation and evaluation.

Figure 4 illustrates how BE-OIS searches the space. On the left is an agent interaction digraph, and on the right are the buckets, indexed by the agent whose outgoing influences are to be eliminated. The buckets are processed from the top-most bucket down, and the topology of message exchange during the course of bucket processing is depicted by the arrows connecting buckets. Appearing within each bucket are the single-agent value functions prior to processing and the multiple-agent value functions that have been processed by earlier buckets.

The operation of BE-OIS proceeds in three phases: *initialization*, *elimination*, and *assignment*, each of which I describe below. As with DF-OIS, I assume that BE-OIS will be initialized and invoked by a central entity. Thereafter, its operation is fully decentralized.

Initialization. To begin, an order is selected over influencing agents that is the reverse of some ordering consistent with the partial order of the interaction digraph.[12] In contrast to DF-OIS, which starts with agents that influence the most peers, BE-OIS begins by reasoning about those influences of the agents that influence fewer peers. In Figure 6.14, the influencing agents $\{4, 3, 2, 1\}$ are considered in that order. For each

[11] I make this assumption without loss of generality. The BE-OIS algorithm that I describe here can be extended to accommodate interaction digraph cycles using the same technique developed in Section 6.4 that enabled DF-OIS to accommodate cycles.

[12] One ordering may yield less computation than another. Dechter (1999) describes algorithms for determining the best ordering.

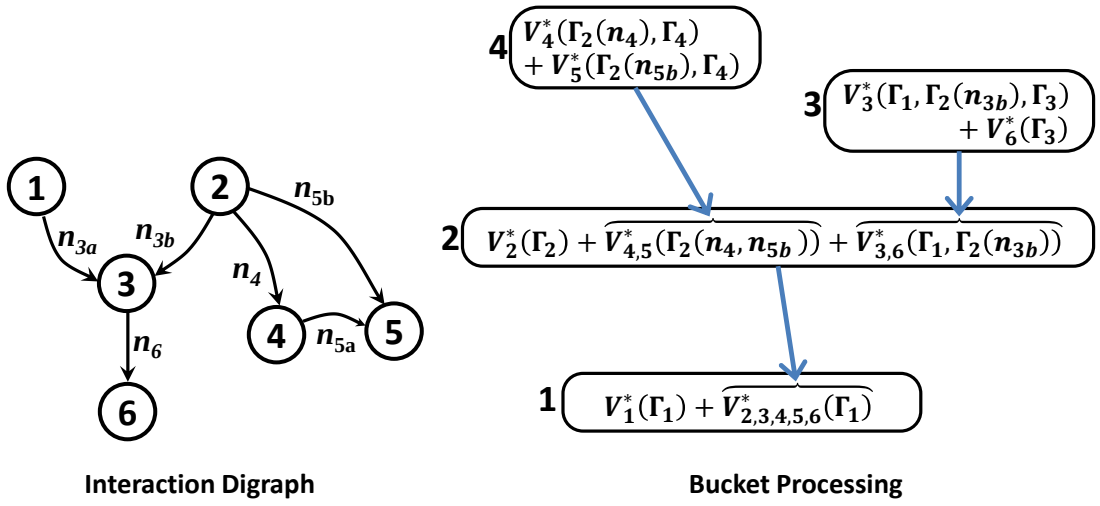


Figure 6.14: Interaction digraph (left) and processing of buckets by BE-OIS (right)

agent in the ordering, a bucket is created for reasoning about the joint value of (and ultimately eliminating) the corresponding agents' outgoing influences.

Next, each agent i's local value function is placed into exactly one bucket. If i influences other agents, i's value function is placed into bucket i. Otherwise, i's value function is placed into the bucket indexed by i' earliest-ordered agent digraph ancestor. This is consistent with Dechter's *bucket elimination* algorithm for the following reason. By Theorem 3.33, the local value function of agent i is independent of the decisions of agents other than i and i's ancestors $\Lambda(i)$. Consequently, agent i's optimal local value with respect to peers' influences is independent of the portions of influence settings that do not pertain to the influences of i' ancestors $\Gamma_{\Lambda(i)}$:

$$V_i^*(\Gamma) = V_i^*(\Gamma_i, \Gamma_{\Lambda(i)}) \tag{6.1}$$

$V_i^*()$, exactly of the form given in Equation 6.1, belongs in the earliest-ordered bucket whose influence it references. At the end of initialization, each bucket i should include i's local value function and the local value functions of any non-influencing children.[13] Henceforth, I will denote $bucket(i)$ as the set of indices of the component value functions in bucket i. I will denote $scope(i)$ as the set of indices, except for i, of all influence components referenced by any agent in $bucket(i)$. Further, I will denote $\Gamma_{scope(i)}$ as the set of all influence parameters, except for Γ_i, referenced by any agent in

[13]In Figure 6.14, notice that buckets 1, 2 and 3 contain additional components. These are components that are added from other buckets during the elimination phase.

bucket(i). Analogous to Dechter's Bucket Elimination, $\Gamma_{scope(i)}$ serves as the variables on which eliminated variables depend.

Elimination. Elimination involves the processing of each bucket by the indexed agent. For each bucket i, agent i may begin processing its bucket i immediately, and in parallel with other agents' eliminations. However, agent i can only finish processing its bucket once all of the buckets earlier in the ordering have finished.

For agent i, the objective in processing its bucket is to compute the optimal setting of its outgoing influences $\Gamma_i^{*|\Gamma_{scope(i)}} = \arg\max_{\Gamma_i} V\left(\Gamma_i, \Gamma_{scope(i)}\right)$ for each feasible setting of peers' influences on which it depends $\{\Gamma_{scope(i)}\}$. In order to compute each $\Gamma_i^{*|\Gamma_{scope(i)}}$, agent i decomposes this computation into the summation of local value functions in bucket i:

$$\Gamma_i^{*|\Gamma_{scope(i)}} = \arg\max_{\Gamma_i} \left[\sum_{j \in bucket(i)} V_j^* \left(\Gamma_i, \Gamma_{scope(i)}\right) \right] \tag{6.2}$$

Before agent i can evaluate Equation 6.2, it must recursively call upon all the agents in *scope*(i) to generate their feasible influence settings. In essence, agent i invokes a depth-first influence-space generation for the subset of agents whose influence parameters are referenced within i's bucket. However, unlike in DF-OIS, the generated influences are stored for later use by all ancestors. For the example problem in Figure 6.14, agent 4 is the first to process its bucket, and hence calls upon agent 2 to generate all of the feasible settings of Γ_2. Upon receiving all feasible combinations of ancestors' influence settings $\{\Gamma_{scope(i)}\}$, agent i decomposes these into the sets of feasible influence settings required to compute each local value function in Equation 6.2. For instance, agent 4 in Figure 6.14 decomposes the set of feasible influences $\{\Gamma_2\}$ into $\{\Gamma_2(n_4)\} \times \{\Gamma_2(n_{5b})\}$. Additionally, agent i generates its own feasible influences (if it has not already done so) for each combination of dependent ancestors' influence settings. All that remains is for agent i to pass the requisite combinations of settings of $\{\Gamma_i\} \times \{\Gamma_{scope(i)}\}$ to any descendant j that does not own a bucket, and to wait for *evaluation* messages back from j, each of the form $\left\langle \langle \Gamma_i, \Gamma_{scope(i)} \rangle, V_j^*(\Gamma_i, \Gamma_{scope(i)}) \right\rangle$. Additionally, agent i must wait for *bucket-processing-completion* messages from all buckets earlier in the ordering (before which additional *evaluation* messages may come).

Agent i concludes its processing of bucket i by performing the maximization in Equation 6.2 for every setting of dependent ancestor influence settings, storing each optimal $\Gamma_i^{*|\Gamma_{scope(i)}}$, for each creating an *evaluation* message $\left\langle \Gamma_{scope(i)}, V_{i,\Psi(i)}(scope(i)) \right\rangle$, and sending all of these evaluation messages to the earliest-ordered agent referenced by the influences in the *evaluation* messages. In the example in Figure 6.14, agent 4

sends an *evaluation* message to agent 2, thereby inserting an additional component into agent 2's bucket. Once all *evaluation* messages are sent, agent i broadcasts a *bucket-processing-completion* message to all agents later in the ordering, notifying them that bucket i has been processed and they can go ahead and finish processing their own buckets.

If agent i is the last agent in the ordering, and hence the last agent to complete its bucket processing, it enters into the *assignment* phase. Note that, in this case, agent i will have eliminated the last influence component, computing a single value Γ_i^* that is the unconditionally optimal setting of agent i's outgoing influence.

Assignment. Whereas in the *elimination* phase, evaluation messages are passed down from bucket to bucket, in the *assignment* phase, optimal influence settings are passed up. The last agent in the ordering has computed its optimal influence assignment Γ_{last}^*, broadcasting this influence setting to previously-ordered agents in an *assignment* message. Recall that each such agent i has stored its optimal influence settings $\Gamma_i^{*|\Gamma_{scope(i)}}$. As soon as i receives all of the optimal settings of $\Gamma_{scope(i)}$, i can assign its optimal influence $\Gamma_i^* = \Gamma_i^{*|\Gamma^*_{scope(i)}}$, and broadcast an *assignment* message. This process continues until all agents will have assigned their optimal outgoing influence settings. As in DF-OIS, once all influences have been assigned, each agent can compute its local component of the optimal joint policy by computing a best response to the optimal influence point.

6.6.4 Complexity of Bucket Elimination OIS

Intuitively, for problems like the one shown in Figure 6.14, BE-OIS has a lower asymptotic complexity than does DF-OIS because BE-OIS does not build a search tree whose depth is the number of agents. Instead, BE-OIS builds a set of smaller search trees, one for each bucket i whose maximum depth is equal to $\|scope(i)\|$. Recall that $\|scope(i)\|$ is the number of other agents whose influences are referenced by the value functions in bucket i. In Dechter's bucket elimination terminology (Dechter, 2003), $\|scope(i)\|$ is the number of variables in bucket i minus 1. Dechter's complexity theory (Dechter, 1999) tells us that, given that bucket elimination uses the optimal ordering over agents, the maximum number of variables in any bucket is equal to the induced width ω^* of the constraint graph. Therefore, given that the induced width of the interaction digraph is ω (Def. 3.36), the number of feasible settings of $\Gamma_{scope(i)}$ that are received by agent i is at most $O(\|\Gamma_i^{max}\|^{\omega-1})$, where $\|\Gamma_i^{max}\|$ is the largest number of feasible settings generated by any agent. In the worst case, for each of these settings,

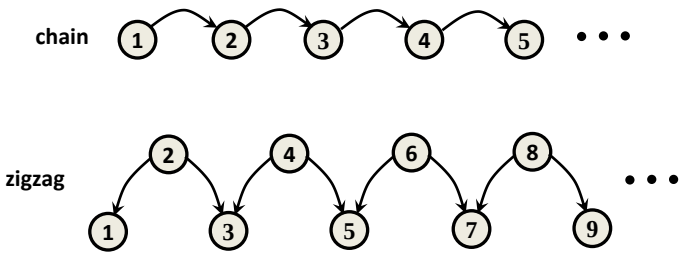


Figure 6.15: "chain" and "zigzag" interaction digraph topologies.

agent i must generate its feasible outgoing influence settings. Since there are at most n buckets, there will be at most $O(n \cdot \|\Gamma_i^{max}\|^{\omega-1})$ generations. Agent i's generation of feasible influence adds another layer to the tree of generated influence settings for bucket i, raising the worst-case total of generated settings to $O(\|\Gamma_i^{max}\|^{\omega})$. This means that the number of influence settings evaluated by any agent in any bucket is at most $O(\|\Gamma_i^{max}\|^{\omega})$. Since there are n agents, there will be at most $O(n \cdot \|\Gamma_i^{max}\|^{\omega})$ evaluations. Since generation and elimination of influence settings dominate all other operations of BE-OIS, the complexity is bounded by:

$$O\left(n \cdot C_{\mathbb{E}} \cdot \|\Gamma_i^{max}\|^{\omega} + n \cdot C_{\mathbb{G}} \cdot \|\Gamma_i^{max}\|^{\omega-1}\right) \tag{6.3}$$

where $C_{\mathbb{E}}$ is the worst-case complexity of the evaluation of any influence setting by any agent, and $C_{\mathbb{G}}$ is the worst-case complexity of the generation of one agent's feasible outgoing influence settings.

6.6.5 Empirical Results

Notice from Equation 6.3 that the complexity of bucket elimination is linear and not exponential in the number of agents. Depth-first search, on the other hand, is necessarily exponential in the number of agents. Bucket elimination does not avoid an exponential term altogether. However, its exponent is bounded by the induced width of the influence digraph. In theory, for problems whose interaction digraphs have a fixed induced width, BE-OIS should scale linearly in the number of agents. To put this hypothesis to the test, I ran both BE-OIS and DF-OIS on a set of 25 random problems (per plotted point) whose interaction digraph is shown in Figure 6.15 (with a topology that I denote *zigzag*). As shown in Figure 6.16, BE-OIS is able to compute optimal solutions for 50 agents, and in orders of magnitude less time that it takes DF-OIS to compute optimal solutions for 6 agents.

Given the ability of bucket elimination to exploit digraph structure (specifically,

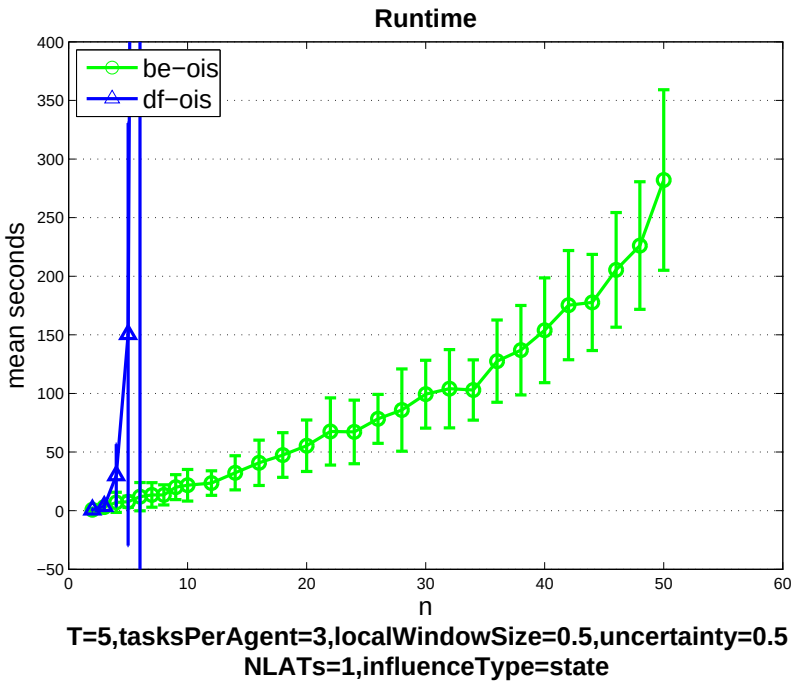


Figure 6.16: Scalability of DF-OIS and BE-OIS on "zigzag" topology.

reduced agent scope), BE-OIS is able to scale well beyond DF-OIS. Moreover, it advances the art of transition-dependent agent planning into a whole new sphere of problems. I have provided compelling evidence that the techniques I have developed are applicable to very large teams of weakly-coupled agents with structured graph topologies, which is a far cry away from the two-agent and three-agent limitations of past work. This magnitude of scalability could not have been accomplished without the exploitation of two complementary aspects of weakly-coupled problem structure (degree of influence *and* agent scope).

The topology of the agents' interaction digraph also plays an important role in this result. In both the *zigzag* topology and the *chain* topology (empirically tested in Section 6.5.6), each agent interacts with at most two other agents. However, due to the directionality of the influences, one problem is significantly harder to solve than the other. In the *chain* topology, the maximum agent scope size (Def. 3.30) is the number of agents; since the last agent in the chain is influenced by all other agents, it must reason about all combinations of other agents' feasible influences. In the *zigzag* topology, agent scope size is at most three, enabling an efficient decomposition of the search through the space of combinations of all agents' influence settings into sub-searches, each through the space of just a couple of agents' influence settings.

CHAPTER 7

Flexible Approximation Techniques

Although OIS is competitive with other optimal algorithms on weakly-coupled problems and scales to particular classes of problems with more agents than was previously possible, there are certainly problems for which computing the optimal solution (using OIS or any other algorithm) is intractable. In this chapter, I demonstrate that my influence-based framework is also suitable for computing approximate solutions. Each of the three techniques that I present has the flavor of flexibly trading optimal solution quality for computational efficiency. In contrast to the previous chapters, this chapter presents a less systematic and more preliminary investigation.

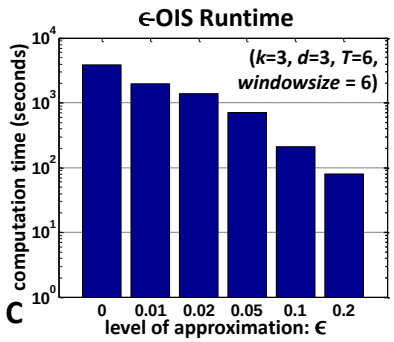
7.1 Approximation of Influence Probabilities

Recall that the influence space searched by OIS consists of vectors of probability values corresponding to the conditional probabilities implied by feasible influences. As I have developed in Section 5.6, generating new points in the influence space involves finding new probability values, component by component. OIS finds all probability combinations. Assuming a fixed influence encoding size, the more tightly-coupled a problem is, the denser the space of probabilities.

The idea behind probability approximation is to avoid the generation of a new influence whose pairwise probabilities are all within ϵ of an influence found previously. This can be done using a very simple modification to OIS's generation algorithm (Sec. 5.6). Each time a new influence γ_{found} is found, the two new intervals added to the explore queue are reduced to $\{(\gamma_{\min}, \gamma_{\text{found}} - \epsilon), (\gamma_{\text{found}} + \epsilon, \gamma_{\max})\}$ such that no parameter values within ϵ of γ_{found} are considered by future MILPs.

Figure 7.1 presents initial empirical results, comparing different values of ϵ on a set of 25 random four-agent (chain) problems (whose interaction digraphs take the form shown at the top of Figure 6.15). In each problem, each agent was given three

tasks, each with three randomly-selected durations (whose probabilities were generated uniformly at random and normalized) and randomly-selected outcome qualities (whose values were drawn randomly from the set $\{1.0, 2.0, 3.0\}$). One of agent 1's tasks (chosen at random) was set to *enable* one of agent 2's tasks (chosen at random), one of agent 2's tasks (chosen at random) was set to *enable* one of agent 3's tasks (chosen at random), and one of agent 3's tasks (chosen at random) was set to *enable* one of agent 4's tasks (chosen at random). The time horizon was set to 6 and every task's window was set to the full duration of execution, making it a strongly-coupled problems (relative to those for which the reduction in *NLATWindow* I showed in Section 4.6.2.5 to yield exponentially-fewer influences).



C

D

ϵ-OIS Solution Quality / Runtime

epsilon	0	0.01	0.02	0.05	0.1	0.2
normalized joint utility value V	1.000	1.000	0.998	0.991	0.990	0.922
stddev(V)	0.000	0.000	0.005	0.020	0.020	0.117
improvement over uncoordinated local optimization	22.7 %	22.7 %	22.5 %	21.7 %	21.6 %	14.6 %
runtime	3832	1985	1369	702.2	208.9	79.80
stddev(runtime)	4440	2502	1995	1013	207.7	60.40

Figure 7.1: Empirical evaluation of ϵ-approximate OIS.

As shown in Figure 7.1, the bar plot indicates a substantial decrease in the runtime (plotted on a log scale) as the value of ϵ is increased. In contrast, the solution quality table[1] shows that the normalized joint utility of the highest-valued influence found decreases very slightly until ϵ becomes larger than 0.1. For this particular set of problems, approximating the influence probability space achieves large computational savings at the expense of very little solution quality.

The performance of ϵ-approximate OIS affirms the intuition that, although agents may forgo finding the optimal influence point by approximating the probability space, they are still guaranteed to search the space relatively evenly (to the extent that it is populated evenly with probabilities). Parameter ϵ specifies the resolution to which they search.

[1]The third row of the table, labeled "improvement over uncoordinated local policies" measures the average percentage improvement over the policy computed by each agent maximizing its local utility without regard to the other agents in the system (assuming pessimistically that its peers will not enable it).

7.2 Time Commitment Abstraction

In the last section, we approximated the space of probabilities associated with each parameter. Alternatively, consider approximating the parameters themselves. That is, approximate the structure of the influence DBN. For example, there may be several features (e.g. cloud cover, time of day, and temperature) that are mutually-modeled by a team of rovers, but that are not all equally informative in predicting the rovers' influences on each other. Feature selection methods could be used to remove all but the most useful influence dependencies. Alternatively, we could remove DBN connections, thereby imposing faux conditional independence relationships. Ultimately, the goal is to reduce the number of parameters that encode the influence, as well as the size of the influence space.

Here, I develop one particular approximation wherein the influence encoding $\Gamma(n_{ix})$ has been reduced to just two parameters (t and ρ) of the form: $\Gamma(n_{ix}) = Pr(n_{ix}^t = true) \geq \rho$, where t is a time value and ρ is a probability value. In contrast to the usual influence information, $Pr(n_{ix}^t = true) \geq \rho$ doe not express a single probability of interaction, but instead a range of probability values (from ρ to 1). The agent proposes to adopt a policy that sets bit n_{ix} to $true$ by time t with probability at least ρ. I call this a *time commitment*. In contrast to the more general notion of an influence, a *time commitment* has the implicit semantics that the nonlocal feature is event-driven (Def. 4.21): once the influencing agent sets it to $true$, it can never be set to $false$ thereafter. Event-driven features are well-suited for modeling interactions among service-oriented agents. After describing the service-oriented context in Section 7.2.1, I present the formal details of time commitments (Section 7.2.2), discuss issues that arise when modeling time commitments (Section 7.2.3), and characterize the search space of time commitments (Section 7.2.4).

7.2.1 Service Coordination

As a context for time commitments, consider a group of agents (such as is shown in Figure 7.2.1) who interact by performing services for one another. I refer to Agent 1 as a *service-providing* agent because it has various tasks that it can perform to fulfill the service requests of other agents. I refer to Agent 2 and Agent 3 as *service-requesting* agents because they can make use of the services provided by Agent 1. In particular, Agent 1 has three services {A, B, and C}, where providing Service A entails the completion of Task A, providing Service B entails the completion of Task B, and providing Service C entails the completion of Task C (which must be preceded by the

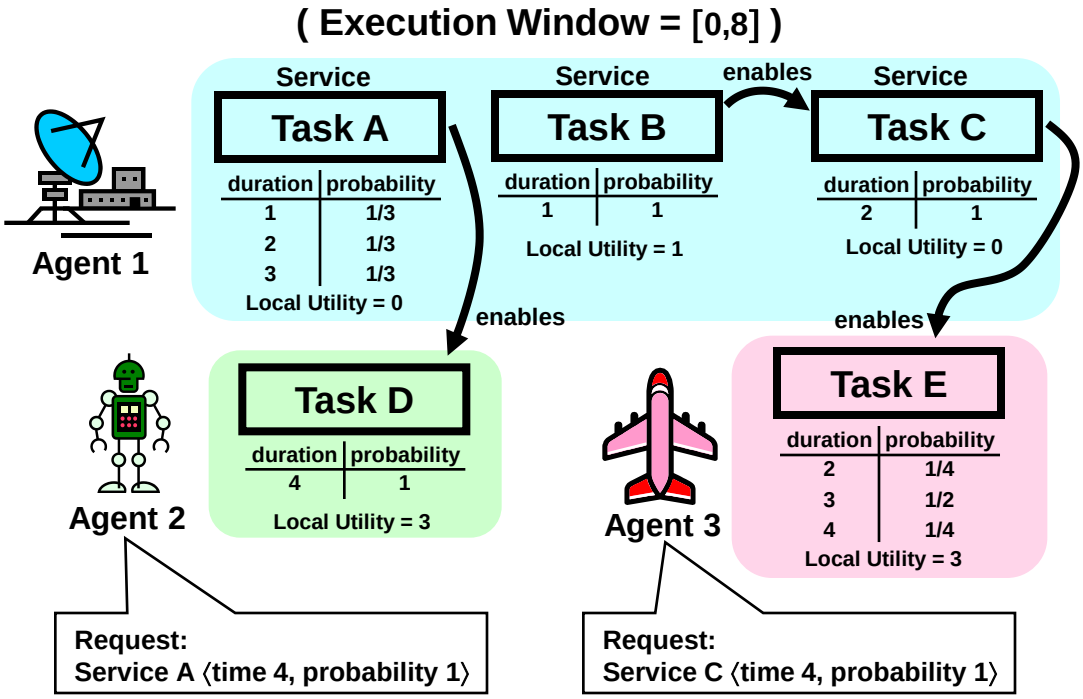


Figure 7.2: Service Coordination example

completion of Task B). These services in turn allow Agents 2 and 3 to complete their own tasks.

7.2.2 Time Commitment Formalism

Within the context of service provision, time commitments are defined as follows:

Definition 7.1. A **probabilistic time commitment** $C_{ij}(s) = \langle t, \rho \rangle$ is a guarantee that agent i will perform (for agent j) the actions necessary to deliver service s by time t with probability no less than ρ.

Probabilistic time commitments allow agents to make promises to each other in the event that they cannot fully guarantee service provision. It can be extremely beneficial to model the inherent service uncertainty in this way. In our example, Agent 1 cannot guarantee provision of Service C until time 6. If Agent 3 waits until time 6, it will only be able to complete Task E (in the case that Task E's duration is 2) with a probability of $\frac{1}{4}$. However, agent 1 *can* promise provision by time 5 with a probability of $\frac{2}{3}$, giving Agent 3 a $\frac{3}{4} \times \frac{2}{3} = \frac{1}{2}$ chance of completing Task E. Thus, by

Agent 1 committing to probabilistically providing Service C at time 5, Agent 3 can take advantage of the temporal uncertainty and effectively double its expected utility.

While this example illustrates that the semantics of the commitment's probability can capture uncertainty about whether a task will be completed in time to meet the time commitment, the probability can also summarize the likelihood that a task will even be attempted. That is, in some execution trajectories, a service provider might reach a state where it would be counterproductive to even begin one of the tasks about which it has made a commitment. This kind of behavior is captured in the commitment semantics: so long as the probability of encountering a trajectory that involves never starting a task, or not finishing it by time t, is no greater than $1 - \rho$, then the provider can make the commitment to complete the task by t with probability at least ρ.

7.2.3 Modeling, Incompleteness, and Inconsistency

Modeling time commitments is a little more tricky than modeling influences because a time commitments does not sufficiently encode the influencing agent's policy. As I describe below, the time commitment model is incomplete because it only specifies (a bound on) the probability at time t, leaving the remain transition probabilities' values unknown. Further, because of the $\geq$ inequality, the transition probability with which the nonlocal feature changes from $false$ to $true$ at the given time may not be exactly equal to ρ. It may be greater than ρ. Whatever the value with which it is modeled, that value may be inconsistent with the true value implied by the service provider's policy. I describe one strategy for coping with this inconsistency below.

A service-requesting agent cannot itself control the provision of Service C, but is concerned with whether or not C will be or has been provided. Hence it should model a nonlocal feature *Service-C-completed*. A commitment can be thought of as a promise from a service-providing agent to be, with probability at least ρ, in a state at time t in which the corresponding nonlocal feature is set. To a service-requesting agent, the commitment is a promise that a nonlocal feature (e.g., *Service-C-completed*) will be set at time t with probability no less than ρ. Thus, from a practical standpoint, the commitment probability ρ corresponds to a portion of the transition probabilities of the nonlocal features in the service-requesting agent's MDP.

Example 7.2. Consider a commitment $C_{13}(C) = \langle 5, \frac{2}{3} \rangle$ by which Agent 1 promises to Agent 3 to complete Task C by time 5 with probability $\geq \frac{2}{3}$. We can augment the transition model in Agent 3's local MDP to represent this committed behavior of Agent 1. As shown in Figure 7.2.3, the transition caused by taking action "N" in state "NN4" is expanded into two possible transitions. This is because Agent 1 has committed to setting the *Service-C-completed* feature by time 5 with probability $\frac{2}{3}$. In this simple problem, there is only one transition at time 4 that is augmented by the modeled commitment, but in general, all transitions leading from time 4 to time 5 would be expanded in this manner.

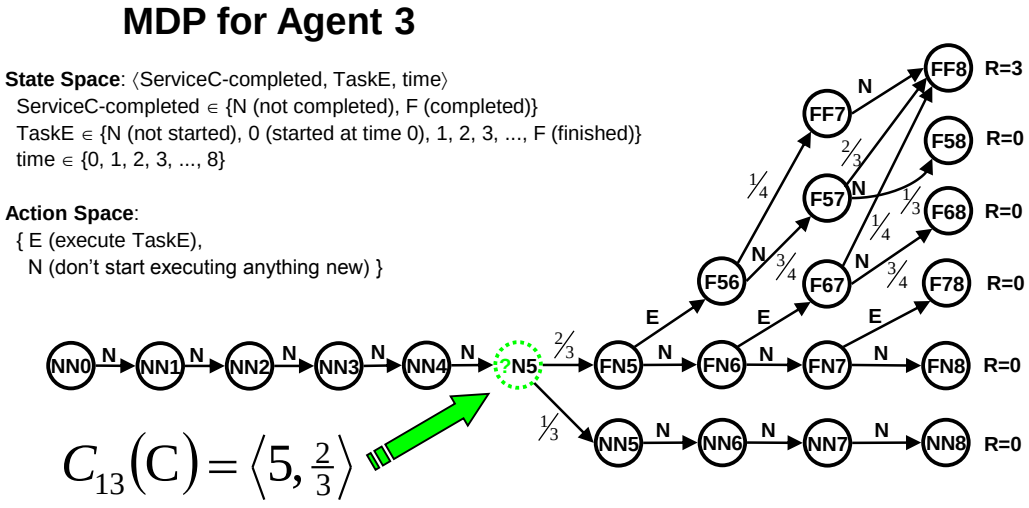


Figure 7.3: A conservative model of a time commitment.

Notice that Agent 3 models *Service-C-completed* as "(N)ot completed" before the commitment time. There is no information encoded in the commitment about the value of the feature at times 0 through 4 nor is there information, in the case that the service is not provided by time 5, about the value of the bit at times 6 through 8.

One method of dealing with the incomplete information of time commitments is for the requesting agent to construct a conservative model of the providing agent. That is, the agent assumes the worst: a zero probability that the services will be provided at all times not referred to by the time commitments. In our example, Agent 3 would assume that *Service-C-completed* takes on a value of N from times 0–4, and

cannot change from N to F after time 5. Modeling the change in feature value of *Service-C-completed* only at time 5 leads to a very compact local model. Note however, that this model is not entirely consistent with the behavior of the service provider. Agent 1 has committed to setting the value of *Service-C-completed* to "F(inished)" *by* time 5 instead of *at* time 5. But agent 3 models the feature as having value "N" at all times before 5. That is, Agent 3 is not modeling the possible completion of Task C any earlier than the commitment time, even though it is possible that Task C might finish at time 4. Similarly, given that Agent 1's policy is really to start task C as soon as it finishes Tasks A and B, then if Task C does not finish at time 5 it must finish at time 6. However, the local model does not include that possibility.

This inconsistency in the agent's local model has the effect that the policy it constructs cannot react quickly to early service provision and cannot react at all to late provision. Consequently, the time commitment abstraction provides approximate solutions. The loss in solution quality due to the approximation will depend on a variety of problem characteristics. Intuitively, the approximation will work well for scenarios involving a single critical time. However, for scenarios with more flexibility in the timing of service provision and utilization, and for highly-uncertain services with a large range of possible completion times, I expect that a time commitment search will perform more poorly.

7.2.4 Space of Time Commitments

Despite issues of incompleteness and inconsistency, an elegant aspect of the *time commitment* is that its domain is a well structured two-dimensional space of time and probabilities with some nice properties. Shown in Figure 7.4, a time commitment could, in principle, take any combination of time and probability values. However, the feasible space of time-probability pairs is bounded from above and from the left by the *maximum feasible probability* boundary. Intuitively, if a particular commitment $\langle t, \rho \rangle$ is feasible, a more conservative commitment that assigns the same probability but at a later time ($\langle t' > t, \rho \rangle$) must also be feasible. Similarly, any commitment $\langle t, \rho' < \rho \rangle$ that promises a lower probability at time t must also be feasible. As a consequence, the feasible boundary is necessarily nondecreasing as a function of the commitment time.

Definition 7.3. The **maximum feasible probability** of a commitment C_{ij} made at time t is the highest commitment probability than can be achieved by time t by any policy of agent i given its existing commitments (if any).

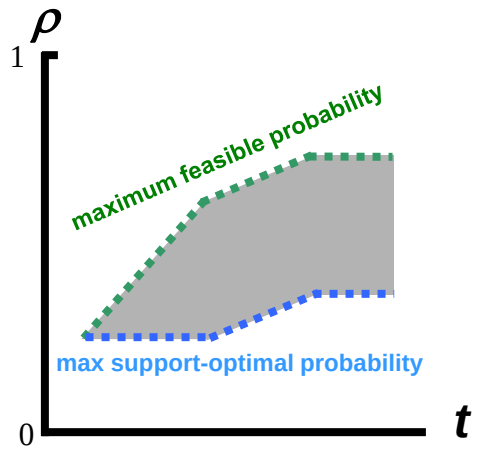


Figure 7.4: The space of feasible time commitments

Given the application of time commitments to service problems, I also assume that the values of commitments for both the provider and the requester are well structured. For the provider, considering commitments all with probability ρ, a commitment at a later time is never of lower local value than a commitment at an earlier time. Similarly, for any two commitments with the same time value, the provider's value for a commitment with a lower probability is never worse than the provider's value for a commitment with a higher probability. I define the highest probability that achieves the highest possible provider value as the *maximum support-optimal probability*:

Definition 7.4. The **maximum support-optimal probability** of a commitment C_{ij} made at time t is the highest commitment probability than can be achieved by time t by any policy of agent i given its existing commitments (if any), without sacrificing any of i's local value.

The opposite relationships hold for the requesting agent. That is, a higher probability of service provision always results in at least as much requester value as a lower probability of service provisions; similarly, earlier time commitments for services can never yield lower requester value than later time commitments. Thus, forming time commitments entails compromise, with the objective of striking a balance (in both the time and the probability dimensions) between the requesting agent's local value and the providing agent's local value. Naturally, this balance should occur between the two boundaries shown in Figure 7.4, where the lower boundary, the *maximum support-optimal probability* boundary, denotes for each time t, the highest probability ρ that does not sacrifice any provider utility. The next section presents one methodology for negotiating such a balance.

7.3 Greedy Service Negotiation

I now present an influence-space search algorithm that uses the *time commitment* abstraction of influence to greedily converge on a feasible time commitment for each interaction. Inspired by service choreography (Papazoglou et al., 2007), the algorithm takes the form of a pairwise agent negotiation between a service-providing agent and service-requesting agent. Although not guaranteed to return optimal time commitments, the negotiation algorithm has several advantageous properties when compared to my optimal influence-space search algorithms, which I list in the paragraphs below.

Greedy Search. Instead of exhaustively exploring the feasible space of influences, the service negotiation algorithm myopically assigns each influence setting one-by-one, never returning to the previous one. In contrast to OIS, service negotiation performs the equivalent of one depth-first pass down the search tree, at each level greedily selecting the time commitment for each service that maximizes the (heuristic) value associated with that service (without regards to the services negotiated thereafter). Consequently, greedy negotiation scales linearly in the number of edges in the interaction digraph regardless of the connectivity, making it robust to strongly-coupled problems where the agent scope is high.

Value-based Pruning. The service negotiation algorithm takes advantage of the properties of time commitments described in Section 7.2.4 to prune large portions of the search space. In particular, it accounts for requester value to rule out time commitments that must be of lower value than those already considered. As such, the service negotiation can be thought of as a *branch and bound* extension to OIS.

Negotiation, Not Enumeration. In contrast to OIS, which dictates that the influencing agent compute its own feasible outgoing influences, service negotiation involves the influenced agent requesting incoming influences in addition to the influencing agents proposing feasible influences. The agents thereby distribute the computational load of influence generation. Additionally, service negotiation begins, for each service, with an initially-proposed influence that maximizes the requesting agents local utility. Although this influence may not be feasible, it starts the search in a fruitful location, and enables swift convergence convergence of the requested and proposed influences.

In the subsections that follow, I develop and evaluate my greedy service negotiation algorithm.

7.3.1 Negotiation Protocol

To plan and coordinate the executions of agents' services, my algorithm utilizes a service choreography protocol. As shown in Figure 7.5, service-requesting agents submit requests to service-providing agents. The requests are dealt with through negotiations between the requester and provider that end in service provision agreements (e.g., time commitments).

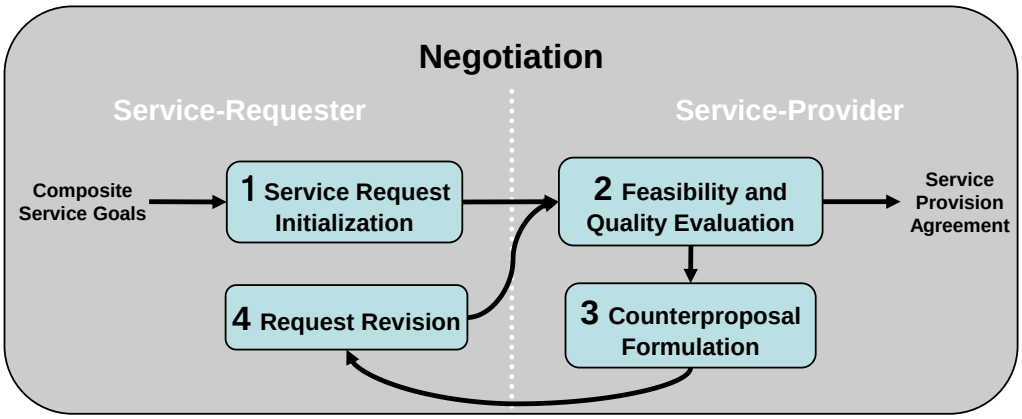


Figure 7.5: Negotiation Protocol

As I describe in the sections that follow, for steps 1 and 4, service-requesting agents employ temporal and stochastic planning to reason about the timing of when the services are needed in order for their own temporally constrained goals to be met. Because of the temporal uncertainty and service dependencies, service-providing agents also employ temporal and stochastic planning techniques in steps 2 and 3 to decide what services can be provided at what times and with what likelihoods.

The remainder of this section is structured as follows. In Section 7.3.2, I provide a methodology for service-provider reasoning: how to constrain its policy-formulation based on its commitments and in doing so evaluate the feasibility of commitments (Figure 7.5 step 2), and how to search the space of commitment values when formulating counterproposals (step 3). In section 7.3.3, I present a corresponding methodology for service requesters to evaluate counterproposals and formulate new service requests (steps 1 and 4). Having brought together all of the steps of the negotiation protocol, I

discuss how the overarching problem of coordinating service activities of the system of agents may be achieved through commitment convergence in Section 7.3.4. In Section 7.3.5, I provide empirical results of the scalability and a discussion of the solution quality of my approach.

7.3.2 Service Provider Reasoning

Next, I describe the inner workings of the negotiation protocol introduced in Figure 7.5. I begin by showing how service-providing agents can evaluate the feasibility of a received request (step 2 of the protocol) and propose alternative commitments (step 3).

7.3.2.1 Forming Commitment-Constrained Policies

Service agents can solve the local models described in the previous section using standard MDP solution methods to compute execution policies. However, in order to adhere to its probabilistic time commitments, a service-provider needs to calculate a policy that keeps its promises. For enforcing commitments, I extend the techniques from Chapter 5 to address time commitments.

We can directly modify the standard MDP LP from Equation 5.1 to constrain the solution policy to adhere to a set of temporal probabilistic commitments:

$$\max \sum_i \sum_a x_{ia} R(i,a) \left| \begin{array}{l} \forall j, \sum_a x_{ja} - \sum_{a,i} x_{ia} P(j|i,a) = \alpha_j \\ \forall i \forall a, x_{ia} \geq 0 \\ \forall s \sum_{\{i|\{time(i)=t_s \wedge Status_s(i)=F\}\}} \sum_a x_{ia} \geq \rho_s \end{array} \right. \quad (7.1)$$

Equation 7.1 adds a third constraint, requiring that the committing agent's policy visit states with $time = t_s$ and a $Finished$ status of service s with aggregate probability no less than ρ_s. This constraint exploits the fact that an occupancy measure must equal the probability of ever visiting a state at time t and taking the action. Since states are time indexed, no more than one state at time t can be visited in any one execution trajectory, nor can the probabilities of visiting any subset of states at time t sum to more than 1. Solving the new linear program yields a policy that is optimal for the committing agent with respect to its commitments to other agents if such a policy exists. If no such policy exists, the agent is overcommitted, and so the Linear Program is over-constrained and has no solution. In this case, the LP solver outputs "NO SOLUTION".

7.3.2.2 Commitment Feasibility

When a service request cannot be honored as requested, the LP formulation will find no solution. Rather than replying "no" to the requester, the protocol expects the provider to supply one or more counterproposals that represent alternative requests that it could commit to fulfilling (step 3 in Figure 7.5). In considering the space of possible counterproposals, not all commitment probabilities and times need be considered. In the following sections, I present some techniques to prune suboptimal values from the space of potential commitment counterproposals.

7.3.2.3 Pruning Commitment Times

Recall that, for the service-providing agent, commitments pertain to the potential completion of its tasks. Each task has a certain discrete probability distribution over durations. Consequently, in order to pick a time to promise a task completion with any probability greater than zero, it does not make sense to consider times that are less than the smallest positive probability duration.

In the example problem, the agent cannot complete Task A before time step 1. For tasks that depend on other tasks, we can push the earliest commitment time further forward by adding the minimum durations of all dependent tasks. Task C depends on the completion of Task B, so the earliest time that should be considered for completing C is $2 + 1 = 3$.

More sophisticated temporal reasoning may be used to push the earliest commitment time forward even further. For example, given an existing commitment by Agent 1 to deliver Service A at time 3, we can deduce that Task A must be started at time 0 and cannot finish any earlier than time step 1. Thus, given previously established commitments, Service C should not be committed to any earlier than time 4. Though I do not incorporate this level reasoning into the implementation I use for my empirical studies (Section 7.3.5), it could be automated by representing the tasks in a temporal constraint satisfaction problem (Dechter, 2003) and applying constraint tightening techniques (e.g., Tsamardinos & Pollack, 2003).

7.3.2.4 Bounding Commitment Probabilities

Having reduced the commitment space with respect to the time dimension, let us now consider the probability dimension. If the service-providing agent makes a commitment to completing Task A at time 2, it makes sense to set the commitment probability equal to the probability with which it can complete A in two time steps

or less: $^2/_3$. If the agent promises a higher probability, it will not be able to meet its commitment. Thus, $^2/_3$ is the maximum feasible probability for Agent 1's commitment to providing A at time 2.

The maximum feasible probability (Def. 7.3) of commitment to service s_k can be computed using a linear program, slightly modified from Equation 7.1, that takes as input the service-providing agent's local MDP with all previously made commitments set to their promised values (denoted $\{\forall s \neq s_k, \langle \rho_s, t_s \rangle\}$), and (using occupancy measures) maximizes the probability of service s_k being delivered at the given time:

$$\max \sum_{\left\{i|time(i)=t_{s_k} \wedge Status_{s_k}(i)=F\right\}} \sum_a x_{ia} \left| \begin{array}{l} \forall j, \sum_a x_{ja} - \sum_{a,i} x_{ia} P\left(j|i,a\right) = \alpha_j \\ \forall i \forall a, x_{ia} \geq 0 \\ \forall s \neq s_k \sum_{\{i|\{time(i)=t_s \wedge Status_s(i)=F\}\}} \sum_a x_{ia} \geq \rho_s \end{array} \right. \tag{7.2}$$

In this new linear program, ρ_{s_k} is a probability variable (unlike the rest of the $\{\rho_s\}$ constants) and the solution maximizes that probability instead of maximizing local utility (as was the case in Equation 7.1).

7.3.2.5 Forming Counterproposals

In OIS, influencing agents generate the entire space of their feasible outgoing influence settings. Here, I suggest a more efficient (though approximate) alternative for counter-proposing feasible time commitments. Instead of generating all feasible time commitments, let the service-providing agent instead compute feasible influences along its *maximum feasible probability* boundary (described in Section 7.2.4 and shown in Figure 7.4). When a request is deemed infeasible, the service provider informs the the requester of its limitations, thereby taking a useful step forward in the negotiation process. For this purpose, the service provider can use the LP in Equation 7.2 repeatedly to calculate the *maximum feasible probability* (Def. 7.3) for all relevant commitment times.

Consider the example from Figure 7.2.1. The first request for A to be completed by time step 3 can be honored and a commitment ($C_{12}(A) = \langle t = 3, \rho = 1.0 \rangle$) formed. But next, the service provider receives a request from Agent 3 to deliver C by time step 4 (with implicit probability 1). Given the first commitment made to Agent 2, a commitment $C_{13}(C) = \langle t = 4, \rho = 1.0 \rangle$ is not feasible. This is shown in Figure 7.6.

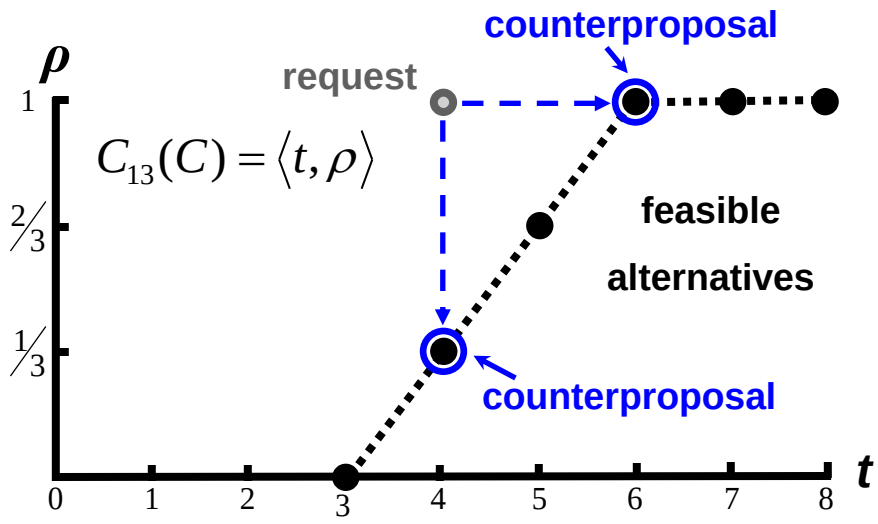


Figure 7.6: An example of counterproposal.

The service provider could, in principle, calculate the entire *maximum feasible probability* boundary over the tightened time interval $[3, 8]$ as shown in Figure 7.6 (and Figure 7.4 abstractly). However, in counter-proposing, it is more efficient to use the time and probability of the request as a basis for providing selective feedback without the provider computing a lot of unnecessary boundary points. As shown in Figure 7.6, C can be delivered at the same time as the original request but with smaller probability, yielding alternative commitment $C'_{13}(C) = \langle t = 4, \rho = \frac{1}{3} \rangle$. Or C can be delivered by a later time, 6, with the same probability as the request, yielding $C''_{13}(C) = \langle t = 6, \rho = 1.0 \rangle$. These two counterproposals give the requester a reasonable sense of the boundary capabilities of the provider near the region of the previous request. Other points along the boundary could be provided, depending on the details of the negotiation algorithm. However, my current implementation finds only these two commitment counterproposals.

The first of the two counterproposals, $C' = \langle t, \rho_2 \rangle$, may be calculated using the probability-maximizing LP from Equation 7.2. The second, $C'' = \langle t_2, \rho \rangle$, requires instead a minimization of feasible commitment time.[2] In Equation 7.3, I define a MILP that does just this, adding boolean variables f_t to account for whether or not a

[2]If the provider cannot achieve the requested commitment probability ρ at any time, the second counterproposal is computed to be $\langle t_3 =$ the earliest time at which ρ_3 can be achieved $, \rho_3 =$ the maximum probability achievable by the time horizon $T \rangle$.

commitment is feasible by time t.

$$\max \sum_t f_t \quad \left| \begin{array}{l} \forall j, \sum_a x_{ja} - \sum_{a,i} x_{ia} P(j|i,a) = \alpha_j \\ \forall i \forall a, x_{ia} \geq 0 \\ \forall s \sum_{\{i|\{time(i)=t_s \wedge Status_s(i)=F\}\}} \sum_a x_{ia} \geq \rho_s \\ \forall t < T, -1 \leq \left(\sum_{\left\{i|\left\{time(i)=t \wedge Status_{s_k}(i)=F\right\}\right\}} \sum_a x_{ia} \right) - \rho_{s_k} - f_t \leq 0 \\ \forall t, f_t \in \{0,1\} \end{array} \right. \tag{7.3}$$

In Equation 7.3, variable f_t can be set to 1 only if the commitment can be satisfied by time t with its original probability ρ_{s_k}. Thus, in maximizing the number of f_t variables that get set to 1, we are effectively minimizing the time that the commitment may be satisfied. The earliest feasible commitment time is then computed by finding the first f_t variable set to 1 ($\min_t \{f_t = 1\}$). As shown in Figure 7.6, this new MILP allows for exploration of the maximum feasible probability boundary by considering horizontal slices through the commitment space (*probabilities*) instead of vertical slices (*times*, as with the LP from Equation 7.2).

7.3.2.6 Issues of Service-Provider Utility

The discussion so far has ignored the fact that service providers also have local utility, separate from the nonlocal utility that they can indirectly increase by fulfilling servicing requests. With the added consideration of service-provider utility, the space of commitments to consider grows, because the "best" commitment in terms of maximizing *total* utility might not be along the maximal feasible probability boundary. That is, by reducing the probability with which it will satisfy another agent's request to a less-than-maximal value, the service provider might be able to develop a policy that improves its own local expected utility enough to more than compensate for the loss in the requesting agent's expected utility.

Here I summarize an extension that may be used to factor in the service providers' local utilities. Equation 7.4 introduces a new linear program that allows a service provider to compute its *maximum support-optimal probability* (Def. 7.4), which is the maximum probability for the commitment at a given time that still allows the

provider to maximize its own local value.

$$\max \rho_{s_k} \left| \begin{array}{l} \forall j, \sum_a x_{ja} - \sum_{a,i} x_{ia} P\left(j|i,a\right) = \alpha_j \\ \forall i \forall a, x_{ia} \geq 0 \\ \forall s \quad \sum_{\{i|\{time(i)=t_s \wedge Status_s(i)=F\}\}} \sum_a x_{ia} \geq \rho_s \\ \sum_i \sum_a x_{ia} R\left(i,a\right) \geq V^*_{\text{provider}} \end{array} \right. \tag{7.4}$$

Note that this is only a slight modification of Equation 7.2: a constraint has been added to ensure that the expected utility of the policy is at least V^*_{provider}, the best local utility achievable by the service provider given its currently-enforced commitments (as computed by applying Equation 5.1 and evaluating the corresponding objective function).

7.3.3 Service Requester Reasoning

Next, I develop methods for requesting services.

7.3.3.1 Request Initialization

To begin the negotiation process, a service requester must formulate an initial request to send to the service provider (step 1 in Figure 7.5). Here I present one method by which all requests may be initialized. A service requester wants to formulate its best possible policy, which it can optimistically formulate by assuming that all of its commitment requests will be satisfied fully as early as it wants. That is, it can imagine that all providers will agree to commitments at time zero with probability 1, and formulate its own optimal policy accordingly, yielding is maximal local expected utility $V^*_{\text{requester}}$. Then, given that it knows this maximal local expected utility, the requester can turn the optimization problem around to find the *latest* time for the commitments that can achieve this utility. I have developed a MILP, shown in Equation 7.5, for computing a policy that performs commitment-enabled actions as late as possible while maintaining that the local utility is no worse than $V^*_{\text{requester}}$.

In Equation 7.5, I introduce integer variables $y_t \in \{0, 1\}$ that can only take a value of 1 if a commitment-utilizing action is performed at or before time t with probability greater than 0 (enforced by using a very small ε variable). Minimizing the sum of the y values forces commitment-utilizing actions to be performed as late as possible. Upon solving the MILP, the earliest time of such an action may be calculated by finding the

first y variable that has value 1: $min_t \{y_t > 0\}$. This earliest commitment-utilization time returned by the linear program is then used as a relaxation time for the requested commitment. These relaxed requests may still be overly optimistic (in terms of the service providers' capabilities), but at least they do not impose unnecessarily demanding requirements on the providers.

$$\min \sum_t y_t \left| \begin{array}{l} \forall j, \sum_a x_{ja} - \sum_{a,i} x_{ia} P(j|i,a) = \alpha_j \\ \forall i \forall a, x_{ia} \geq 0 \\ \forall s \sum_{\{i|\{time(i)=t_s \wedge Status_s(i)=F\}\}} \sum_a x_{ia} \geq \rho_s \\ \sum_i \sum_a x_{ia} R(i,a) \geq V^*_{\text{requester}} \\ \forall t < T, -1 \leq \left(\sum_{\{i,a|\{time(i) \leq t \wedge enables(C,a)\}\}} x_{ia} \right) - y_t - \varepsilon \leq 0 \\ \forall t, y_t \in \{0,1\} \end{array} \right. \tag{7.5}$$

7.3.3.2 Request Revision

Next I discuss how a service-requesting agent like Agent 3 would process the commitments counter-proposed by a service provider in its negotiations (step 4 in Figure 7.5). Just like the service provider, the service requester can evaluate utilities of various counter-proposed commitments by solving local commitment-augmented MDPs (using the LP from Equation 7.1 in Section 7.3.2.1) and calculating the expected utilities of their respective solution policies (using Equation 2.7). The (self-interested) object of the requester is to find the best possible feasible commitment and thereby maximize its local utility.

Along these lines, one very simple method of formulating a new request is to evaluate each counterproposal, identify the best one, and request it. In my running example, Agent 3 either could choose time 6 with probability 1 (giving it an expected local utility of 0.75), or time 4 with probability $\frac{1}{3}$ (giving it an expected utility of 1.0). Agent 3 would then request the latter. A slightly more advanced variation is to further consider a commitment time and probability between the bounds of the counterproposals. The requester can simply interpolate optimistically, computing and evaluating the potential utility of a request whose time is halfway between the two counterproposals and whose probability is equal to the maximum probability of the two counterproposals. In the case of my running example, this optimistically-

interpolated request corresponds to commitment $C'''_{13}(C) = \langle t = 5, \rho = 1 \rangle$. Although this interpolated commitment request will not be feasible, in this case the provider will respond with more counterproposals to better inform the requester of the boundary capabilities. By iterating back and forth in this way, the potential commitment time window will narrow monotonically and (since time is discrete) the process must terminate when the requester is unable to interpolate further. This strategy of re-requesting is implemented in the commitment convergence algorithm presented in the next section.

From the perspective of the service requester, another response to counterproposals from potential service providers might be to consider them collectively, and accept multiple such proposals. In my running example, had there been a second potential provider for service C, the service request could have gone to it as well as to Agent 1. Assume for a moment that having service C at time 4 is important for the requester. The counterproposal from Agent 1 specifies that, at time 4, there is a probability of $\frac{1}{3}$ that service C will be accomplished. If the other provider responded that, at time 4, it could provide C with a probability of $\frac{1}{2}$, then the requester has options. It could certainly choose to enlist the other agent to provide C, because of the higher probability. However, assuming that the possible providers are otherwise idle, and that they can pursue C concurrently and independently, the requester could accept *both* counterproposals, so as to increase the probability that at least one provision of C will succeed to $\frac{2}{3}$.

7.3.4 Negotiation-Driven Commitment Convergence

Each request made to a service provider may be handled using the negotiation protocol introduced in Figure 7.5. As in the running example problem, each service-providing agent is first given a sequence of these incoming requests. The idea is to consider each request one at a time, converging on an agreement with the service-providing agent(s) through negotiation before moving to the next request. Our agents therefore search the space of commitments of all service requests greedily by setting the commitments one at a time. This strategy enables much quicker commitment convergence than would an exhaustive search (but at the potential loss of solution quality).

Pseudo-code for my commitment convergence algorithm is shown in Algorithm 7.1. Each step of the algorithm involves agents solving linear programs (as described in Sections 7.3.2 and 7.3.3) to reason about requests, counterproposals, and optimal local behavior. One by one, each original request is dealt with in a pairwise negotiation

between *provider* and *requester*. The two agents iterate through sets of potential commitment values and eventually converge on a single agreed commitment for each requested service. This convergence of commitment values is guaranteed (in a number of iterations logarithmic in the problem time horizon) because of the methods agents use for counter-proposing and re-requesting.

As is typical of greedy algorithms, a drawback of this particular commitment negotiation algorithm is that, by greedily maximizing the utility associated with the current commitment to a service provision, it can sacrifice potential solution quality of later service provisions. In my running example, if the service requests are handled in the order that they are shown in Figure 7.2.1, negotiation yields commitments $C_{12}(A) = \langle t = 4, p = 1.0 \rangle$ and $C_{13}(C) = \langle t = 5, p = \frac{2}{3} \rangle$. Given that the completion of Task A by time 3 is worth a local utility gain of u_2 to Agent 2 and the completion of Task C by time 4 is worth a local utility gain of u_3 to Agent 3, these two commitments together provide the requesters a total expected gain of $u_2 + \frac{1}{2}u_3$. If we were to reverse the order in which the requests are considered in the example problem, the negotiation protocol brings us to a different set of commitments. A commitment $C_{13}(C) = \langle t = 4, p = 1.0 \rangle$ will be made to Agent 3 promising the completion of Task C by time step 4. However, when the provider next negotiates with Agent 2, it can only make commitments involving the execution of Task A after Tasks B and C. Otherwise its first commitment would be violated. In the example, this results in Task A finishing at time 4 with probability $\frac{1}{3}$. The completion of Task A after time 4 does not benefit Agent 2 at all. Thus, by using this alternate request order, negotiations converge on a set of commitments that provide the requesters a total gain of $\frac{1}{3}u_2 + u_3$.

Which ordering produces the better solution is dependent upon the relative utility benefit values u_2 and u_3. Specifically, the first commitments are preferable when u_2 is worth at least $\frac{1}{2}$ of u_3, but otherwise the other commitments would be preferred. Although additional ordering heuristics could be overlaid on top of the greedy protocol described here, it is difficult to ensure in general that the right ordering will be attempted. Furthermore, the optimal set of commitments might not be achievable by greedy convergence using any ordering. It may be that two requesting agents will receive the greatest collective utility if they both compromise on the probability and/or time by which they are providing competing services. Such a compromise can only be achieved by simultaneously considering both potential commitments (instead of considering them one-by-one as with the greedy algorithm).

Algorithm 7.1 Greedy Request-Based Search for Commitments

```
1: procedure GRBS(p, agents)                          ▷ Input: problem p, service agents
   ...Initialization...
2:     C ← ∅                                          ▷ The commitment set (stored by all agents)
3:     for each agent ∈ agents do
4:         requests ← agent.FormInitialRequests(p)    ▷ [see Sec. 7.3.3.1]
5:         agent.CommunicateRequests(requests, agents)
6:     end for

   ...Greedy Commitment Convergence...
7:     for each provider ∈ agents do
8:         for each r ∈ provider.ReadIncomingRequests() do
9:             requester ← r.sender
10:            acceptable ← provider.EvaluateFeasibility(r, p, C)        ▷ [7.3.2.1]
11:            while acceptable = false do
12:                cp ← provider.GenerateCounterproposals(r, p, C)
                                                                          ▷ [7.3.2.5]
13:                requester.EvaluateAndMemorize(cp, p, C)
14:                r ← requester.GenerateNewRequest(cp, p, C)
                                                                          ▷ [7.3.3.2]
15:                acceptable ← provider.EvaluateFeasibility(r, p, C)
16:            end while
17:            r ← requester.RelaxRequest(r, p, C)                       ▷ [7.3.3.1]
18:            c ← provider.FormCommitment(r)
19:            provider.CommunicateCommitment(c, agents)
20:            C ← C ∪ {c}                         ▷ New commitment added (by all agents)
21:        end for
22:    end for

   ...Optimal Local Policy Formulation...
23:    for each agent_i ∈ agents do
24:        π*_i ← agent_i.ComputeConstrainedOptimalPolicy(p, C)
25:    end for
26:    return π ← ⟨π*_1, ..., π*_n⟩                  ▷ Output: joint policy
27: end procedure
```

7.3.5 Empirical Results

Two separate empirical studies follow. In the first, I analyze the scalability of greedy service negotiation and compare its quality with that of a MMDP (Boutilier, 1996) solver on several sets of randomly-generated service problems. In the second, I perform a very preliminary comparison with OIS on the problem sets tested in Chapter 6.

7.3.5.1 Comparison with MMDP Solver

The motivation for developing a greedy service negotiation approach is to be able to solve larger, more complex problems well with less computational effort. To this end, I evaluate how scaling up problem difficulty affects runtime, and how the greedy and approximate techniques impact solution quality. In this comparison, which summarizes my published results (Witwicki & Durfee, 2009), I compare greedy service negotiation against a MMDP solver. My solver uses the same machinery as the "centralized MILP" approach described in Section 6.5.1.1, but does not constrain agents' observability. Instead, it computes a joint policy that assumes that each agent observes the full global state at every time step. In contrast to my service negotiation algorithm, the MMDP is a centralized planning model that finds optimal joint policies by simultaneously accounting for all agents' policy decisions. Greedy service negotiation exploits the largely decoupled structure in service coordination problems, but produces only approximately-optimal solutions. Because the MMDP does not exploit structure or approximation like my approach can, its runtime should be viewed more as providing a worst-case bound on computational effort. On the other hand, because the MMDP solver produces optimal joint policies that assume each agent has full global state awareness at all times, the expected qualities of its joint policies provide a best-case bound on the agents' collective performance. In contrast, greedy service negotiation assumes that agents only know their local state and whether other agents have succeeded or failed in meeting their time commitments. Thus, while the bounds are not tight, the MMDP solver provides well-defined performance bounds against which to compare greedy service negotiation.

I begin by presenting scalability results that demonstrate the scalability of greedy service negotiation. Figure 7.7 shows the runtime on variations of exactly the example problem (presented in Figure 7.2.1), where each variation is scaled up by simply stretching out the timing of all tasks[3] and extending the time horizon accordingly

[3]Tasks maintain the same number of discrete durations, but each possible duration is scaled.

(from T=8 to T=96). This leads to larger MDPs, more LP constraints, and potentially more iterations of commitment requesting and counter-proposing. As can be seen in Figure 7.7, the algorithm remains tractable for time horizons as a large as T=96 (at which point CPLEX is solving constrained MDPs with over 10,000 states), converging on commitments in a minute or less. I compare this runtime with solving the Multiagent MDP, which scales much worse with the problem time horizon, taking hours to return the optimal solution (utility = 6.0) for time horizons of greater than 40, whereas the commitment-based algorithm returned a near-optimal solution (utility = 5.5) in under a minute for problems with horizons as large as 96. This result provides some evidence that, although approximate, greedy service negotiation can produce reasonable solutions tractably, scaling gracefully with the problem time horizon.

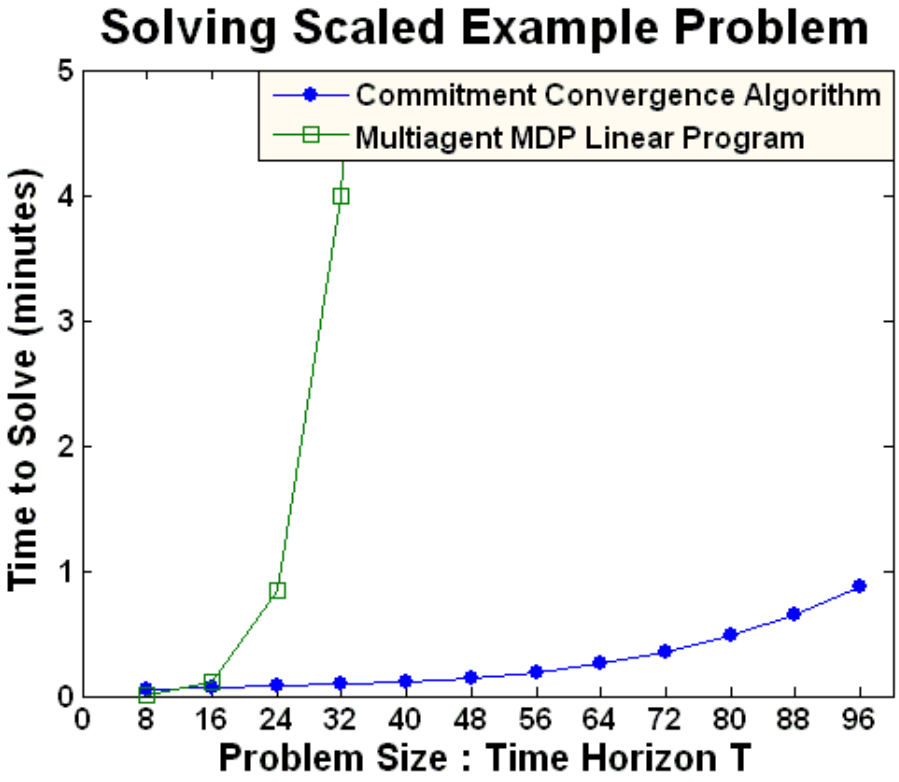


Figure 7.7: Scalability: problem time horizon.

Next, I scale the local complexity of the example problem by adding random *local tasks* , each of which is not enabled by other agents' services and may not be requested by other agents. This has the consequence that the agents' problems are tied to one another with the same interaction structure as in the running example. However, each of the agents problems becomes more complicated with additional local tasks (that may accrue utility) and additional local dependencies between internal tasks and

services. Through random task additions, I automatically generate sets of random problems (25 for each data point). Details of the problem generation schemes used throughout this section are provided in Appendix B.

Figure 7.8 shows the results achieved on these augmented problems. Average running time is plotted on a log scale. This experiment offers strong evidence that greedy service negotiation scales to problems with increasingly complex local behavior. For random locally-augmented problems with more than 8 tasks, the MMDP takes more than an hour to solve on average (not shown). Note that as more local tasks are added, the agents' individual decision problems are becoming larger, but also more weakly-coupled from one another. It is this weak coupling (as discussed in Section 7.2.3) that enables commitment-based negotiation to remain so much more efficient.

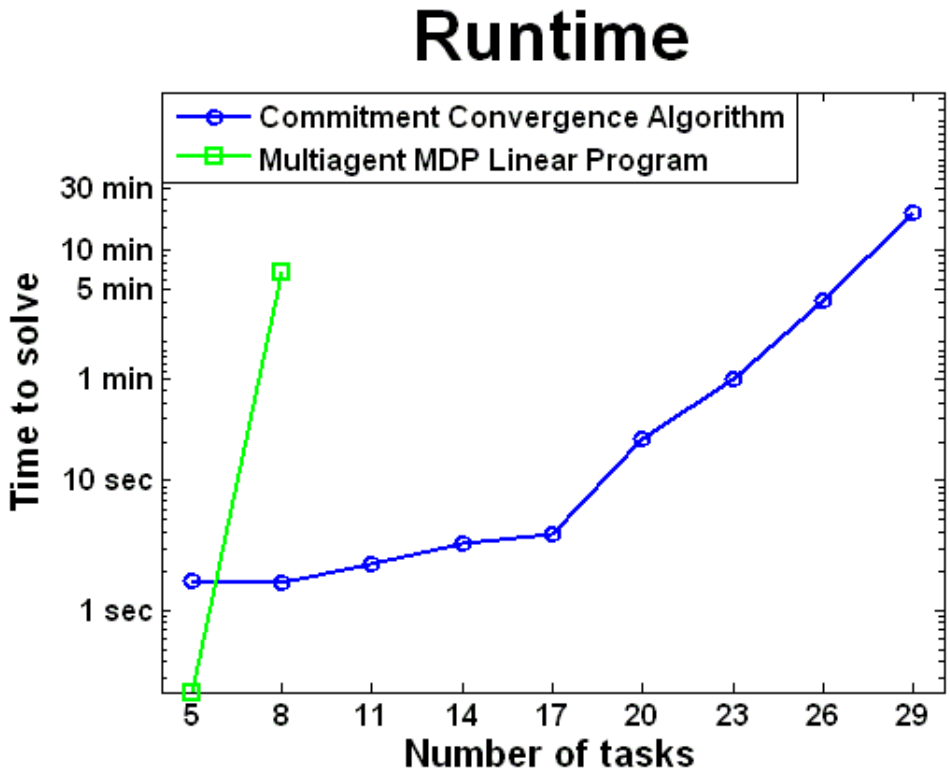


Figure 7.8: Scalability: local complexity.

In one more scalability test, I increase the size of the example problem by adding additional service-requesting agents. Maintaining the single-service-provider structure, randomly-generated agents are added, each with a small number of local tasks and a single service requirement. Further details are included in Appendix B. The average runtime for 25 random problems per data point is plotted in Figure 7.9. This time, notice that my commitment negotiation approach scales roughly linearly with the

number of agents. The MMDP scales exponentially, and therefore quickly becomes intractable.

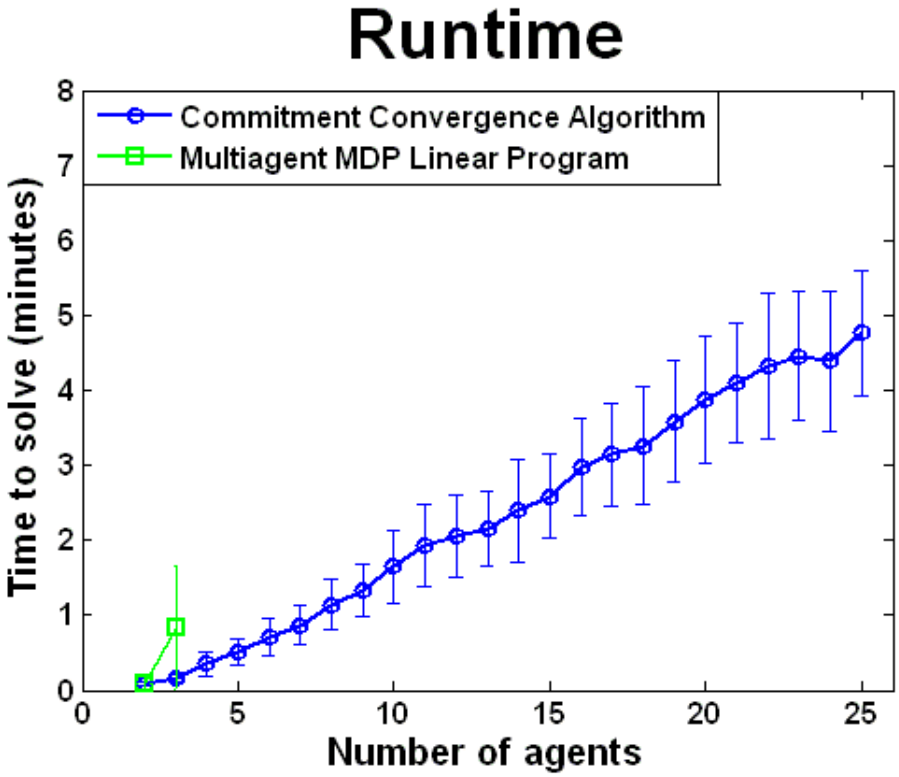


Figure 7.9: Scalability: number of agents.

An alternative metric common in multiagent and service-oriented systems is the number of messages passed between agents. With my negotiation protocol, the number of messages scales linearly with the number of service provision relationships, regardless of the number of agents involved. Each relationship requires a minimum of 2 messages (for request and agreement) and can require a number of messages logarithmic in the number of time points in the worst case, if the requesting agent continues to request the optimistic interpolation (as described in Section 7.3.3.2). Of course, the number of messages can be decreased by creating more informative messages (and incurring the costs of forming those messages). For example, if the provider replies at the outset with the entire feasible probability boundary, then no iteration is needed.

The computational benefits of greedy service negotiation over the MMDP come at a price in terms of the potential quality of the agents' joint solution. Figure 7.10 shows the difference in quality by empirical comparison on a set of 25 randomly-generated service problems. Each problem contains three agents and a total of 9 tasks randomly distributed between the agents. There are 3 random *enablement* NLEs, but unlike in

the example problem, the services are not all provided by the same agent. Agents are not exclusively providers or requesters. The dependencies between each agent's local tasks are random as are the task duration distributions. More details are provided in Appendix B.

Although none of the individual problems are based on actual real-world service composition scenarios, I sought to generate a set of problems representative of a wide range of potential three-agent scenarios, remaining impartial about characteristics such as service composition hierarchy, tightness of timing, and distribution of local utility. This evaluation provides preliminary evidence that my algorithms may produce coordinated, high-quality solutions for a variety of service composition problems.

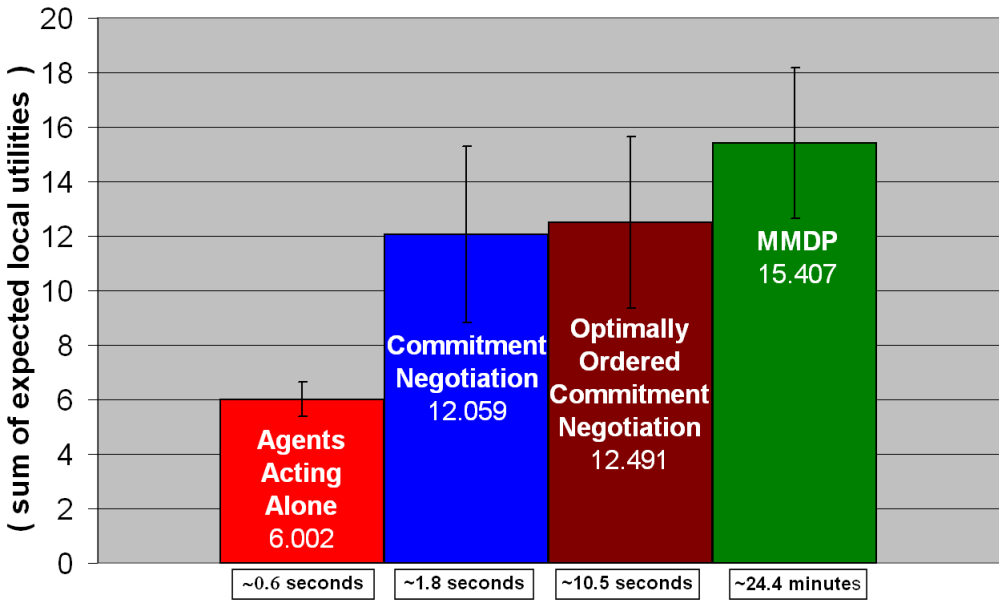


Figure 7.10: Average solution quality on 25 random problems

The height of the bars represent solution quality, measured as the sum of the expected local utilities of the three agents. The corresponding error bars represent standard deviation in the solution qualities across the 25 problems. The number under each bar represents the average time to converge or (in the case of the MMDP) to compute the optimal solution. The left-most bar in Figure 7.10 indicates the average quality of a solution approach in which agents plan with completely-independent local models that do not consider any possibility of service provision from other agents. That is, agents build optimal local policies around an empty set of commitments. This approach serves as a lower bound over which coordinated agent behavior should rise. As shown in Figure 7.10, greedy service negotiation performs significantly better.

My approach performs nearly as well as the optimal MMDP solution approach in a fraction of the computation time.

Greedy service negotiation yields coordinated policies for these random service problems, achieving higher solution quality than that of uncoordinated policies, but this solution quality is, on average, lower than that of the optimal MMDP solution. Reasons for this gap in solution quality include the following. As described in Section 7.3.4, commitment values are converged upon greedily, one by one, and in a fixed order. I have included an intermediate data point to account for a portion of this loss of quality. The bar labeled "Optimally-Ordered Commitment Negotiation" represents the commitment convergence algorithm performed on all possible orderings of commitments so that the highest-quality joint policy (corresponding to the optimal commitment ordering) is selected. However, this approach still makes greedy choices (given the optimal ordering). The MMDP formulation, on the other hand, always makes the correct choices and always converges on the globally optimal joint policy for the agents. It always finds the best balance of service provision to multiple service requesters, as well as the best balance of provider and requester utility.

Another drawback as compared with the MMDP formulation is that, in order to achieve compactness and efficiency, agents time-commitment-based models make some approximations of nonlocal agent behavior. For example, the agents forgo potential flexibility and sacrifice potential expected utility by representing each service commitment with just a single time and probability. That is, unlike the MMDP that assumes agents have global awareness and can react suitably when a service is provided earlier (or later) than planned, my approach (as described) only allows agents to model and react at the service's committed time. However, there is nothing in the negotiation protocol that precludes making multiple (conditional) commitments at different times for each request. It is the subject of future work to combine service negotiation with richer influence representations. However, this would likely enlarge the influence space space, and so should be done with care.

7.3.5.2 Comparison with OIS

I now present a very preliminary comparison of greedy service coordination and OIS. First, I demonstrate the superior scalability of greedy service negotiation on a set of 50 random problems (per number of agents) whose interaction digraphs have the *chain* topology (shown in Figure 6.15). Problems are generated using the same generator as described in Section 6.5.6, with randomly selected nonlocally-enabling tasks set as *services*. Figure 7.11 plots the computation times of OIS and greedy service

negotiation on a logarithmic scale. The plot only extends to 10 agents, but greedy service negotiation is capable of scaling to hundreds. Regardless of the interaction digraph topology, greedy service negotiation scales linearly. Moreover, notice that for two-agent problems, greedy service negotiation performs more than an order of magnitude faster than OIS, due to the advantages that I listed in Section 7.3.

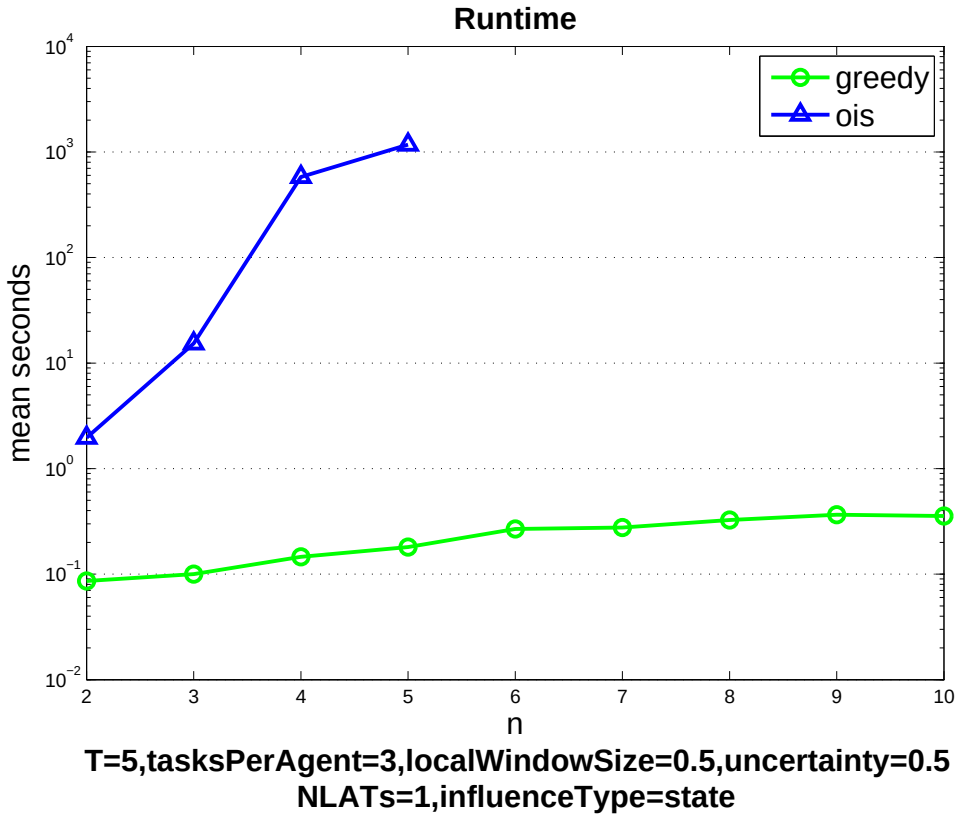


Figure 7.11: Scalability: OIS vs. Greedy Service Negotiation on "chain" topology.

However, it is not fair to compare only the runtimes of these two algorithms. In addition to imposing stricter restrictions on the problems than OIS, greedy service negotiation returns approximate solutions. Figure 7.12 shows the quality of solutions returned by greedy service negotiation relative to those returned by OIS. On top, the average absolute solution quality of both methods is plotted. On the bottom, the "Percentage Optimal" refers to the average of the ratio of returned solution quality to optimal solution quality. Clearly, greedy service negotiation trades some quality for its faster computation. However, further analysis is required to characterize the trade-off.

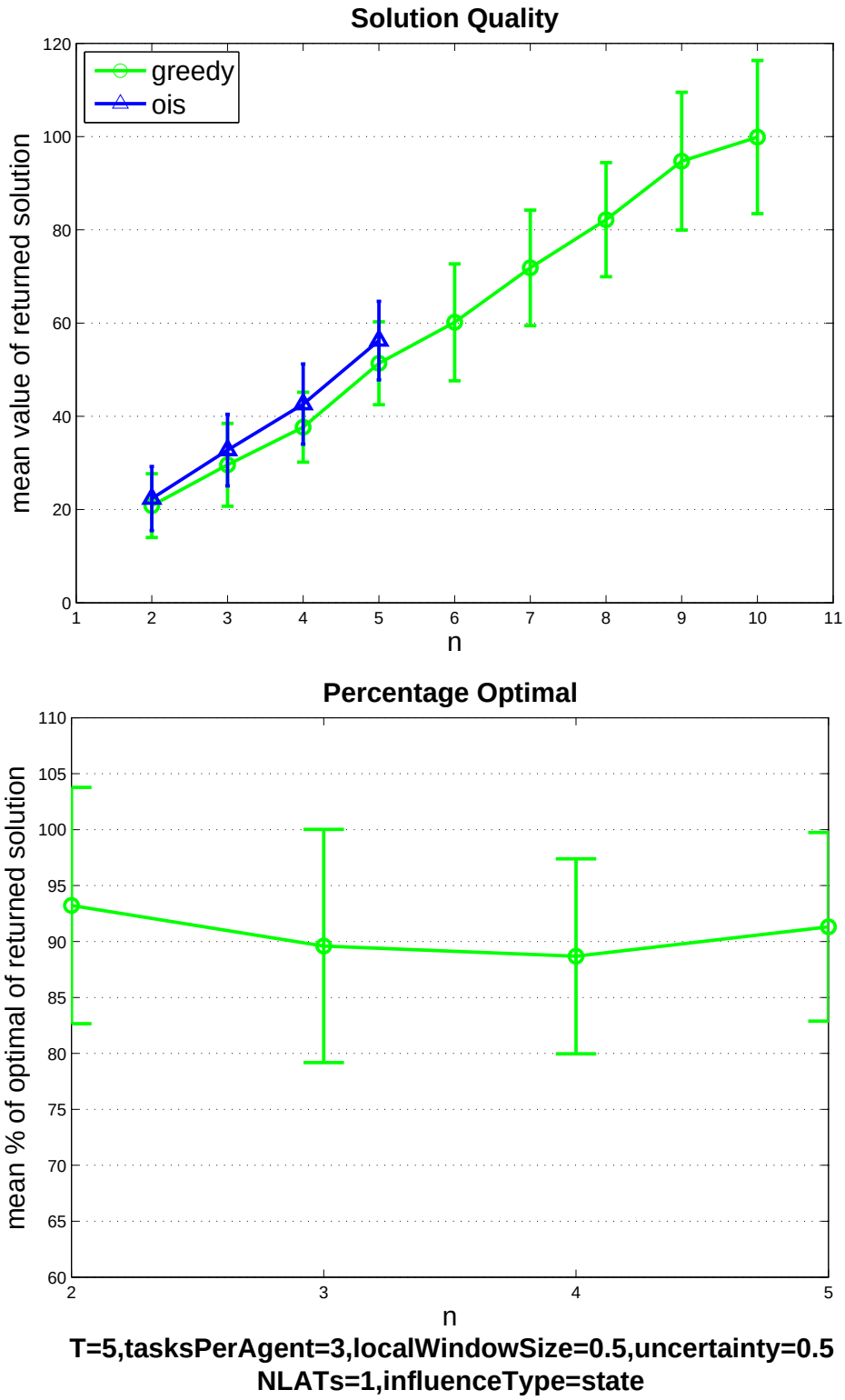


Figure 7.12: Solution quality of Greedy Service Negotiation on "chain" topology.

CHAPTER 8

Conclusions

The focus of this dissertation has been on the development of a general framework for abstracting agents' influences and coordinating using those abstractions, with the ambition of advancing the state of the art in efficiency and scalability of planning for teams of weakly-coupled agents under uncertainty. Towards this goal, I have made several contributions to the field, each of which I summarize in Section 8.1. My work has also raised new research questions, several of which I discuss in Section 8.2. I conclude in Section 8.3, reflecting upon the accomplishments of this dissertation with respect to my longer-term aspirations.

8.1 Summary of Contributions

In the subsections that follow, I organize my contributions along four research thrusts, each of which is an integral component of my coordination framework. These thrusts and their constituent contributions correspond to the work that I have presented in Chapters 3, 4, 5, and 6.

8.1.1 Identifying Structure

Chapter 3 focuses on the identification of weakly-coupled structure in transition-dependent problems.

- I have defined a model (the TD-POMDP), for teams of transition-dependent agents, that articulates several elements of exploitable structure that were previously exploited only in more restricted contexts. This structure gives rise to an abstraction of transition probabilities and a subsequent decomposition of the joint model into efficiently-solvable local models.

- I have developed a characterization that brings together three complementary aspects of weakly-coupled problem structure present in the TD-POMDP model. Along with this characterization, I contribute theory on the complexity of solving weakly-coupled problems, by exploiting the three aspects in concert, dependent on the degree to which each aspect of weakly-coupled structure is present. Aside from promoting a better understanding of the structural exploitations of other approaches, this theory motivates my investigation of influence-based abstraction as a method for reducing the size of the search space required for coordinating optimal behavior. It also guides my empirical analysis of the conditions under which influence-based abstraction is most effective at reducing the search space and most efficient in practice.

8.1.2 Abstracting Influences

Chapter 4 focuses on the abstraction of influence information that is sufficient for optimal local planning and reasoning.

- I have developed a novel best-response model for TD-POMDP agents, the complexity of which depends on the number of shared state features regardless of either the number of agents or the complexity of peers' behavior.

- Out of my best-response model comes a principled representation for policy abstraction, *influence*, that encodes information from peers' policies sufficient for optimal local reasoning, and whose encoding size is also a function of the number of shared state features irrespective of the number of agents. The conceptual contribution is the (later empirically validated) insight that, by formalizing agents' transition influences, an influence space emerges that is potentially more efficient to search than the policy space, yet is still amenable to optimal solutions.

- In Section 4.6, I have presented an empirical analysis of the size of the feasible influence space in relation to the size of the policy space. The results contribute towards characterizing the circumstances under which influence-based policy abstraction is most advantageous.

8.1.3 Proposing and Evaluating Influences

Chapter 5 presents a principled approach to constraining local policies around jointly-coordinated influences.

- By conceptually linking the probabilistic information encoded in the MDP LP occupation measures with that encoded in agents' influences, I have formalized a mapping between influence and policy. In one direction, an agent can evaluate the influence settings that its policy implies by solving a linear program corresponding to its best response and evaluating a formula for each influence parameter. In the other direction, an agent can constrain its policy to directly adhere to a proposed influence.

- I have proven that an alternative approach, reward shaping, though potentially more efficient than LP-based constrained policy formulation, is not guaranteed to enforce a prescribed influence (even if that influence is feasible) regardless of the amount of parameter tuning. Moreover, while reward shaping may only produce approximately optimal local policies with respect to a prescribed influence, influence-constrained policy formulation ensures that the local policies will be optimal among the policies that achieve the influence.

- I have developed a method for enumerating an agent's space of feasible outgoing influences efficiently. Instead of having to consider each of its policies explicitly, my algorithm solves a number of MILPs that is proportional to the number of feasible influence points.

- In employing influence constraints to perform a number of different functions in Chapters 5 and 7, I have contributed an arsenal of constrained policy formulation techniques that may be adapted and extended to solve other decision-making problems involving behavioral constraints.

8.1.4 Coordinating Influences

- My primary contribution in Chapter 6 is the development of my *optimal influence-space search* algorithm, which combines components from each of the earlier chapters so as to decompose the policy formulation problem into a series of well-ordered influence generation and influence evaluation steps that are guaranteed to search all feasible combinations of agents' influences, one of which I have proven must correspond to the optimal joint policy.

- I have performed an empirical comparison of the computation time of optimal influence-space search with that of several other optimal algorithms, thereby identifying the strengths and weaknesses of my methodology. In doing so, I contribute compelling evidence in support of my hypothesis that optimal

influence-space search provides significant computational advantage over existing methods in the computation of optimal solutions for certain classes of weakly-coupled problems. Moreover, my analysis evaluates the circumstances under which influence-space search gains the most traction in practice, providing data with which researchers and practitioners can make informed decisions about the suitability of adapting or applying influence-based abstraction and optimal influence-space search to their own problems.

- In Section 6.6.3, my initial study of exploiting agent scope in combination with degree of influence suggests the following: For problems with a low degree of influence and a fixed agent scope, influence-space search can achieve scalability in the number of transition-dependent agents well beyond the state of the art in optimal policy computation. Moreover, this portion of my work contributes a novel application of Dechter's *Bucket Elimination* to influence-space generation and optimization.

- I present additional evaluations in Chapter 7 that contribute evidence of the efficacy of approximate influence-space search to reducing computation while still achieving near-optimal solution quality.

8.2 Open Questions

This work has uncovered a number of interesting questions that are beyond the scope of this dissertation, but that are candidates for potentially-fruitful investigation in future work.

8.2.1 Quality-Bounded Influence Space Search

I have devoted this dissertation primarily to the study of *optimal* influence-based methods. To date, I remain compelled by the formal guarantees that optimal algorithms provide. However, I am also interested in developing approximate algorithms with bounds on quality loss. Intuitively, the approximation of the influence probability space (Section 7.1) should, under some conditions, provide such guarantees. Inherently, it already guarantees consideration of an influence whose parameter settings are within some bound of the settings of the optimal influence point. Further investigation is needed to determine whether or not we can also bound quality, and whether or not those bounds would be useful in practice.

8.2.2 Influence Encoding Compaction

Based on my characterization of influence encodings (e.g. state-dependent, history-dependent, influence-dependent) in Section 4.3, different problems may accommodate different sufficient encodings. In particular, my empirical analysis in Section 4.6 considered two classes of problems: one in which a history- and influence-dependent encoding was needed and another in which a state-dependent encoding sufficed. I justified this latter sufficiency for acyclic cases with the knowledge that the nonlocal features were all *event-driven* (Def. 4.21), and with my theoretical results from Section 4.4. In my experiments, I manually set the *state-dependent* flag for such cases. What I did not do is to automate the determination of the appropriate encoding. Clearly, a smaller encoding is preferred for compactness of best-response models, for efficiency of influence generation, and for reduced influence space size (as my empirical results in Section 4.6.2.4 suggest). In the interest of automation, I pose the following questions:

- How can we automatically diagnose inefficiencies in the influence encoding and reduce the size of the encoding (e.g., by eliminating unnecessary variables, removing unnecessary connections, or instead revising the semantics of the encoding to allow a smaller number of parameters)?
- What are the circumstances under which extremely compact encodings (e.g., time commitments developed in Section 7.2) yield optimal solutions?
- Is there additional problem structure that could be exploited to reduce the size of the influence DBN?

8.2.3 Other Applications of Influence Abstraction

In this dissertation, I have demonstrated that, *for the problem of planning weakly-coupled cooperative agents' decisions*, influence-based abstraction is a powerful tool for decomposing one large joint problem into smaller, more efficiently-solvable local problems. The decomposition is made possible by the fact that, given few shared features, the resulting compact model of *influence* is a sufficient summary of nonlocal information for making local predictions. Note that neither decomposition nor prediction are specific to *planning*; nor are they specific to *cooperative agents*. I believe that there are other weakly-coupled problems that could similarly benefit from my influence formalism (as well as the structural aspects of my TD-POMDP model), such as the following:

- learning (assuming that the structure of agents' interactions is known a priori);
- reasoning about influences of adversaries in a competitive domain;
- incentivizing agents' adoption of socially-beneficial influences (in the context of mechanism design).

8.3 Closing Remarks

The research that I have presented in this dissertation is largely motivated by the long-term vision of applying multiagent sequential decision making models and techniques to solving real-world problems. Realizing this vision requires, among other things, bridging the gap between the limitations of the current state of Dec-POMDP research (which as of yet has been restricted to small toy problems involving few agents, or restrictive forms of interactions, or no bounds on solution quality) and the objectives of practitioners that might ultimately apply the Dec-POMDP technologies. My work aspires to narrow the gap by extending the state-of-the-art in efficient computation of optimal solutions to teams of weakly-coupled transition-dependent agents. Although small in respect to the size of this gap, my influence-based abstraction approach has inched out beyond the reach of other methods, computing solutions faster and scaling to more agents that was previously possible on a small, but well-characterized space of problems. This achievement is a direct result of exploiting structure in transition-dependent problems. In fact, the greatest advances in agent scalability were made possible by simultaneously leveraging two different aspects of structure: *locality of interaction*, via constraint optimization algorithms, and *degree of influence*, via policy abstraction. Moreover, the combination of both techniques yielded far greater gains than were seen by applying either one individually. Such complementarity of structural exploitations, and of algorithmic frameworks, gives me hope that, by identifying and exploiting more structure, the field of Dec-POMDP research will one day close the gap, and successful applications will be realized. It may take just a few more complementary advances.

APPENDICES

APPENDIX A

Comparison of EDI-Dec-MDP and TD-POMDP

The Dec-MDP with Event-Driven Interactions (Becker et al., 2004a), otherwise known as the Event-Driven Dec-MDP (EDI-Dec-MDP), is the most closely-related subclass to that of the TD-POMDP. Here, I present the technical details of EDI-Dec-MDP and prove that it is no more general than the TD-POMDP.

A.1 Event-Driven Dec-MDP Model

The EDI-Dec-MDP is specified by the tuple $\mathcal{M} = \langle \mathcal{N}, S, A, P, R, \Omega, O, T, \{d_{ij}^k\} \rangle$, wherein the usual suspects are as follows: $\mathcal{N}$ is the set of agents, S is a set of world states, $A = A_1 \times A_2 \times ... \times A_n$ is the joint action space, $P : S \times A \times S \to [0, 1]$ is the transition function, $R : S \times A \to \mathbb{R}^n$ is the joint reward function, $\Omega = \Omega_1 \times \Omega_2 \times ... \times \Omega_n$ is the joint observation space, $O : S \times A \times \Omega \to \mathbb{R}$ is the observation function, and T is the finite horizon. Like the TD-POMDP, the world state S is *factored* into local state components S_i. However, the EDI-Dec-MDPs factoring assumes no sharing of state features among local states: $S = \times_{i \in \mathcal{N}} S_i$. Additionally, the observation function is restricted such that the EDI-Dec-MDP is *locally fully observable* (Definition 2.8): $\forall o_i, \exists s_i | Pr(s_i|o_i) = 1$. Further, the reward function is restricted such that EDI-Dec-MDP agents are *reward independent* (Definition 2.11) such that local rewards combine by summation to equal the joint reward: $R(s, a, s) = \sum_{i \in \mathcal{N}} R_i(s_i, a_i)$.

Event-Driven DEC-MDPs have structured transition dependencies Becker et al. (2004a), the set of which is denoted as $\{d_{ij}^k\}$. In particular, one agent may influence the local state transitions of another through the occurrence of a *proper event*.

Definition A.1. A **primitive event** $e = (s_i, a_i, s_i')$ is a triplet of state, action, and outcome state that may occur in agent i's execution history $\Phi_i = [s_i^0, a_i^0, s_i^1, a_i^1, ...]$.

Definition A.2. An **event** $E = \{e_1, e_2, ..., e_h\}$ is a set of primitive events that is said to occur ($\Phi_i \models E_i^k$) in an execution sequence if one of the primitive events occurs in the execution sequence.

Definition A.3. A primitive event is **proper** if it can occur at most once in any possible history. An event $E = \{e_1, e_2, ..., e_h\}$ is proper if all of its primitive events are proper and no two primitive events can both occur in any possible history.

Interactions among EDI-Dec-MDP agents thereby occur though event *dependencies* of the form $d_{ij}^k = \langle E_i^k, D_j^k \rangle$, whereby an event in E_i^k brings about a change in the transitions, D_j^k (which is made up of state-action pairs), of agent j. Dependency satisfaction is captured by Boolean variable b_{s_j,a_j}^k, which is true when an event in E_i^k has occurred. Subsequently, the transition function P of a EDI-Dec-MDP is structured such that agent j's local transition function P_j, in addition to depending on local state s_j, depends on the nonlocally-affected boolean variable b_{s_j,a_j}^k. As such, the agents' *local state* transitions are independent of one another with the exception of event dependencies captured by the b_{s_j,a_j}^k variables.

A.2 Complexity of EDI-Dec-MDP

Allen (2009) has recently proven that the computational complexity of the EDI-Dec-MDP is NEXP-complete, which is the same complexity class as the Dec-POMDP. This means that, to solve the EDI-Dec-MDP requires computation time irreducibly exponential, and space unbounded, in the model size the model size $\|\mathcal{M}^{\text{edi}}\|$).

A.3 Reduction of EDI-Dec-MDP to TD-POMDP

Any EDI-Dec-MDP can be reduced to an equivalent TD-POMDP simply by treating the event-driven boolean features b_{s_j,a_j}^k as nonlocal features. Theorem A.4 formalizes this sentiment.

Theorem A.4. *EDI-Dec-MDP* $\leq_{EXP}$ *TD-POMDP.*

Proof. Let us treat the reduction to the equivalent TD-POMDP $\mathcal{M}^{\text{td}} = \langle \mathcal{N}^{\text{td}}, \{S_j^{\text{td}}\}, \{A_j^{\text{td}}\}, \{\Omega_j^{\text{td}}\}, \{O_j^{\text{td}}\}, \{R_j^{\text{td}}\}, \{\bar{m}_j^{\text{td}}\}, \{P_j^U\}, \{P_j^L\}, T \rangle$ (as per definition 3.15) one component at a time:

- $\mathcal{N}$ is identical to $\mathcal{N}^{\text{edi}}$.

- Let the TD-POMDP world state include a boolean feature $n_j = b^k_{s_j,a_j}$ for every EDI-Dec-MDP even dependency variable b^k. Further let each n_j be a nonlocal feature controlled by agent i (where i is the other agent reference in b^k. As such, each local state space of the TD-POMDP S^{td}_j is the local state space in the EDI-Dec-MDP augmented with each corresponding nonlocal features n_j and each nonlocal feature of some other agent n_i controlled by j. The remaining features in S^{td}_j are treated as locally-controlled features (Def. 3.12). As of yet, we have used time $O(\|d^k_{ij}\|)$ and space $O(\sum_{j\in\mathcal{N}} 2^{\|d^k_{ij}\|\cdot\|S_j\|})$.

- A^{td}_j is identical to A^{edi}_j.

- Ω^{td}_j is identical to Ω^{edi}_j.

- O^{td}_j is equivalent to O^{edi}_j, such that agent j's observations depend only on the local features of s^{td}_j (or equivalently, all the features of s^{edi}_j). This reduction is valid since the EDI-Dec-MDPs local full observability makes O^{edi}_j more restrictive in its representation.

- R^{td}_j is identical to R^{edi}_j (with the same conditional independence of nonlocal features as is the case in O^{td}_j).

- $\bar{m}^{\text{td}}_j$ is the union of the set of nonlocal features $\{n_j\}$, each controlled by another agent i, that influence agent j, and the set of nonlocal feature $\{n_i\}$, each controlled by j. Since there is one nonlocal feature per dependency, each involving 2 agents, we will have performed a number of steps and used space proportional to to $2 \cdot \|\{d^k_{ij}\}\|$.

- The TD-POMDP's uncontrollable feature transition function P^U_j will be defined over an empty set since we are treating all features as *locally-controlled*.

- The only difference between the TD-POMDP's local transition function P^L_j and P^{edi}_j is that for P^{edi}_j is defined over a smaller set of states, each of which do not encode the Boolean dependency variables of the form $b^k_{s_i,a_i}$ related to agent j's events but that influences i. Since P^L_j is defined over a state space that includes these these additional variables, P^L_j must expand the necessary transition information for each combination of value of $b^k_{s_j,a_j}$ for each state s according to the whether or not the corresponding event E^k_j has occurred. The total size of the TD-POMDP local transition function P^L_j is $\|P^{\text{edi}}_j\| \cdot 2^{\|d^k_{ij}\|}$ in the worst case, so this step of the reduction takes time and space proportional to $\|P^{\text{edi}}_j\| \cdot 2^{\|d^k_{ij}\|}$.

- $T^{\text{td}} = T^{\text{edi}}$.

Upon completing the steps above, the result is a completely-specified TD-POMDP that captures the same semantics of the EDI-Dec-MDP from which it was reduced. Since each step in the reduction takes at most $O(\|P_j^{\text{edi}}\| \cdot 2^{\|d_{ij}^k\|})$ time an space, and each step has been validated (as per the semantics of each model component), the claim made in Theorem A.4 is proved. □

APPENDIX B

Random Service Problem Generation

Local Complexity Scale-up Experiment. For the evaluation of local complexity, shown in Figure 7.8, tasks were generated and added one by one to each agent's local problem (from the running example). These random local tasks were generated as follows. Each task duration distribution was computed by selecting an interval $[1, k]$ of time units, where k was randomly chosen from a normal distribution centered around $\frac{T}{2}$, and then invoking a uniformly random number of possible durations from that interval, each randomly valued within the interval and randomly assigned a probability (derived from normalizing a set of random numbers). For each local task added to an agent's local problem, that task was probabilistically connected (via dependency) to an existing task. That is, with a probability of 0.3 the task was made to enable an existing task chosen at random with equal probability (but with the constraint that the existing task wasn't already enabled by another task). Next, with a probability of 0.3 the task was made dependent on an existing local task chosen at random with equal probability. The utilities of these additional local tasks were selected uniformly randomly in the interval [0,3].

Agent Scale-up Experiment. For the evaluation of number of agents, shown in Figure 7.9, additional requesting agents were generated and added one by one to the example service coordination problem. These agents' problems were generated as follows. First, a random number n of tasks was selected in the interval $[1, 6]$ with equal probability. Next, one by one, each of the n tasks was randomly generated and added to the new agent's problem exactly as dictated by the random task generation scheme in the preceding paragraph. Finally, one of these tasks was chosen at random

as being dependent on one of the services (selected randomly from those) provided by the service-provider. Utilities of tasks requiring the service of another agent were valued uniformly randomly over the interval $[2, 5]$ and utilities of all other tasks were valued uniformly randomly over the interval $[0, 3]$.

Solution Quality Experiment. For the evaluation of solution quality, shown in Figure 7.10, problems were generated randomly from scratch. Each of the problems was initialized with 3 agents, each containing empty local problems. 9 randomly-generated tasks (as generated using the scheme in the *local complexity experiment*) were randomly distributed between the 3 agents. Next random service dependencies were introduced to connect the agents problems to one another. Three service enablements were imposed, each connecting two randomly selected tasks (constrained to come from two different agent's local problems). The effect was to create random service compositional hierarchies. For each of these random coordination problems, there was no longer (necessarily) a single service-provider. Each agent had the potential of both providing services and requesting services. The utilities of tasks in these problems was valued as in the preceding paragraph. That is, utilities of tasks requiring the service of another agent were valued uniformly randomly over the interval $[2, 5]$ and utilities of all other tasks were valued uniformly randomly over the interval $[0, 3]$.

BIBLIOGRAPHY

Allen, M. (2009). *Agent Interactions in Decentralized Environments*. Ph.D. thesis, University of Massachusetts, Amherst, Massachusetts.

Amato, C., Bernstein, D., & Zilberstein, S. (2007). Optimizing memory-bounded controllers for decentralized POMDPs. *UAI*, (pp. 1–8).

Atlas, J. (2009). A distributed constraint optimization approach for coordination under uncertainty. In *Autonomous Agents and Multiagent Systems (AAMAS-09)*, (pp. 1263–1264).

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*(4), 341–379.

Beaty, D., Grady, M., May, L., & Gardini, B. (2008). Preliminary planning for an international mars sample return mission. *Report of the iMARS Working Group*, (pp. 1–60). `http://mepag.jpl.nasa.gov/reports/iMARS_FinalReport.pdf`.

Becker, R. (2006). *Exploiting Structure in Decentralized Markov Decision Processes*. Ph.D. thesis, University of Massachusetts Amherst.

Becker, R., Zilberstein, S., & Lesser, V. (2004a). Decentralized Markov decision processes with event-driven interactions. In *Autonomous Agents and Multiagent Systems (AAMAS-04)*, (pp. 302–309).

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. (2004b). Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, *22*, 423–455.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2003). Transition-independent decentralized Markov decision processes. In *International Conference on Autonomous Agents and Multi Agent Systems*, (pp. 41–48). Melbourne, Australia.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

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

Bernstein, D., Hansen, E., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. *IJCAI*, (pp. 1287–1292).

Bernstein, D. S. (2005). *Complexity Analysis and Optimal Algorithms for Decentralized Decision Making*. Ph.D. thesis, University of Massachusetts Amherst.

Bernstein, D. S., Amato, C., Hansen, E. A., & Zilberstein, S. (2009). Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, *34*, 89–132.

Bernstein, D. S., Zilberstein, S., & Immerman, N. (2000). The complexity of decentralized control of Markov decision processes. In *Uncertainty in Artificial Intelligence*, (pp. 32–37). Stanford, California.

Bernstein, D. S., Zilberstein, S., Washington, R., & Bresina., J. L. (2001). Planetary rover control as a Markov decision process. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*. Montreal, Canada.

Beynier, A., & Mouaddib, A. (2005). A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *Autonomous Agents and Multiagent Systems (AAMAS-05)*, (pp. 963–969).

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Theoretical Aspects of Rationality and Knowledge*, (pp. 195–210). San Francisco, CA, USA.

Boutilier, C., Brafman, R., & Geib, C. (1997). Prioritized goal decomposition of Markov decision processes: Towards a synthesis of classical and decision theoretic planning. In M. Pollack (Ed.) *International Joint Conference on Artificial Intelligence*, (pp. 1156–1163).

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Brafman, R. I., & Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *International Conference on Automated Planning and Scheduling*, (pp. 28–35).

Cassandra, A. R., Kaelbling, L. P., & Kurien, J. A. (1996). Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 963–972).

Cavallo, R., Parkes, D. C., & Singh, S. (2006). Optimal coordination of loosely-coupled self-interested robots. In *the Workshop on Auction Mechanisms for Robot Coordination, AAAI-06*. Boston, MA.

Clement, B. J., Durfee, E. H., & Barrett, A. C. (2007). Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, *28*, 453–515.

Cohen, P., & Levesque, H. (1991). Teamwork. *Nous*, *35*, 487–512.

Cohen, P. R., & Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, *42*(2-3), 213–261.

Cox, J. S., & Durfee, E. H. (2003). Discovering and exploiting synergy between hierarchical planning agents. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, (pp. 281–288).

Dean, T., & Givan, R. (1997). Model minimization in Markov decision processes. In *AAAI/IAAI*, (pp. 106–111).

Dean, T., Givan, R., & Kim, K.-E. (1998). Solving stochastic planning problems with large state and action spaces. In *Artificial Intelligence Planning Systems*, (pp. 102–110).

Dean, T., & Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *International Joint Conference on Artificial Intelligence*.

Dearden, R., & Boutilier, C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, *89*(1-2), 219–283.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1-2), 41–85.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Decker, K. (1996). TAEMS: A framework for environment centered analysis & design of coordination mechanisms. In *Foundations of Distributed Artificial Intelligence, Ch. 16*, (pp. 429–448).

Decker, K., & Lesser, V. (1992). Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems*, *1*(2), 319–346.

D'Epenoux, F. (1963). A probabilistic production and inventory problem. *Management Science*, *10*, 98108.

Dolgov, D. A., & Durfee, E. H. (2004a). Graphical models in local, asymmetric multi-agent Markov decision processes. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, (pp. 956–963). New York.

Dolgov, D. A., & Durfee, E. H. (2004b). Optimal resource allocation and policy formulation in loosely-coupled Markov decision processes. In *International Conference on Automated Planning and Scheduling (ICAPS 04)*, (pp. 315–324). Whistler, BC.

Dolgov, D. A., & Durfee, E. H. (2005). Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors. In *International Joint Conference on Artificial Intelligence (IJCAI-05)*.

Dolgov, D. A., & Durfee, E. H. (2006). Resource allocation among agents with MDP-induced preferences. *Journal of Artificial Intelligence Research*, *27*, 505–549.

Durfee, E. H., & Lesser, V. R. (1991). Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(5), 1167–1183.

Fikes, R., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, *2*(3/4), 189–208.

Gmytrasiewicz, P. J., & Doshi, P. (2004). Interactive POMDPs: Properties and preliminary results. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, (pp. 1374–1375).

Goldman, C., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, *22*, 143–174.

Goldman, C. V., & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. In *International Conference on Autonomous Agents and Multi Agent Systems*, (pp. 137–144). Melbourne, Australia.

Grosz, B. J., & Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, *86*(2), 269–357.

Guestrin, C., & Gordon, G. (2002). Distributed planning in hierarchical factored MDPs. In *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, (pp. 197–206). Edmonton, Canada.

Guestrin, C., Koller, D., & Parr, R. (2001). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, (pp. 1523–1530). Vancouver, Canada.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Guestrin, C., Venkataraman, S., & Koller, D. (2002). Context-specific multiagent coordination and planning with factored MDPs. In *Eighteenth national conference on Artificial intelligence (AAAI-02)*, (pp. 253–259).

Guo, A., & Lesser, V. (2005). Planning for weakly-coupled partially observable stochastic games. In *international joint conference on Artificial intelligence*, (pp. 1715–1716).

Hansen, E., Bernstein, D., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. *AAAI*, (pp. 709–715).

Horling, B., Lesser, V., Vincent, R., & Wagner, T. (2006). The soft real-time agent control architecture. *Autonomous Agents and Multiagent Systems (AAMAS-2006)*, *12*(1), 35–92.

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

Jonsson, A., & Barto, A. (2005). A causal approach to hierarchical decomposition of factored MDPs. In *ICML '05: international conference on Machine learning*, (pp. 401–408).

Jordan, M. I. (Ed.) (1999). *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *ARTIFICIAL INTELLIGENCE*, *101*, 99–134.

Kallenberg, L. (1983). *Linear Programming and Finite Markovian Control Problems*. Math. Centrum, Amsterdam.

Kearns, M. J., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *IJCAI*, (pp. 740–747).

Kim, Y., Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2006). Exploiting locality of interaction in networked distributed POMDPs. In *In AAAI Spring Symposium on Distributed Planning and Scheduling*.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kube, C. R. (1997). Task modelling in collective robotics. *Autonomous Robots*, *4*, 53–72.

Kumar, A., & Zilberstein, S. (2009). Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, (pp. 561–568).

Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., NagendraPrasad, M., Raja, A., Vincent, R., Xuan, P., & Zhang, X. (2004). Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Journal of Autonomous Agents and Multiagent Systems*, *9*(1), 87–143.

Littman, M. L. (1996). *Algorithms for Sequential Decision Making*. Ph.D. thesis, Brown University, Providence, RI.

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995a). Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, (pp. 362–370).

Littman, M. L., Dean, T., & Kaelbling, L. P. (1995b). On the complexity of solving Markov decision problems. In *UAI*, (pp. 394–402).

Lusena, C., Goldsmith, J., & Mundhenk, M. (2001). Nonapproximability results for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, *14*, 83–103.

Madani, O., Hanks, S., & Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI '99/IAAI '99: national conference on artificial intelligence and innovative applications of artificial intelligence*, (pp. 541–548).

Marecki, J., Gupta, T., Varakantham, P., Tambe, M., & Yokoo, M. (2008). Not all agents are equal: Scaling up distributed POMDPs for agent networks. In *International Joint Conference on Autonomous Agents and Multiagent Systems*.

Marecki, J., & Tambe, M. (2007). On opportunistic techniques for solving decentralized MDPs with temporal constraints. In *International Joint Conference on Autonomous Agents and Multi-agent Systems*.

Marecki, J., & Tambe, M. (2009). Planning with continuous resources for agent teams. In *Autonomous Agents and Multiagent Systems (AAMAS-09)*, (pp. 1089–1096).

Mataric, M. J. (1997). Reinforcement learning in the multi-robot domain. *Auton. Robots*, *4*(1), 73–83.

Melo, F. S. (2008). Exploiting locality of interactions using a policy-gradient approach in multiagent learning. In *Proceeding of the 2008 conference on ECAI 2008*, (pp. 157–161).

Messias, J. V., Spaan, M. T., & Lima, P. U. (2010). Multi-robot planning under uncertainty with communication: a case study. In *The MSDM workshop at AAMAS-2010*, (pp. 54–61). Toronto, Canada.

Meuleau, N., Hauskrecht, M., Kim, K.-E., Peshkin, L., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Solving very large weakly coupled Markov decision processes. In *AAAI '98/IAAI '98: Artificial intelligence/Innovative applications of artificial intelligence*, (pp. 165–172).

Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, *16*(1–2), 149–180.

Mostafa, H., & Lesser, V. (2009). Offline Planning For Communication By Exploiting Structured Interactions In Decentralized MDPs. In *Intelligent Agent Technologies (IAT-09)*, (pp. 193–200). Milan, Italy.

Musliner, D. J., Durfee, E. H., Wu, J., Dolgov, D. A., Goldman, R. P., & Boddy, M. S. (2006). Coordinated plan management using multiagent MDPs. In *Working Notes of the AAAI Spring Symp. on Distributed Plan and Schedule Management*.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D. V., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI-03*, (pp. 705–711).

Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2005). Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. *AAAI-05*, (pp. 133–139).

Ogata, K. (1997). *Modern control engineering (3rd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Oliehoek, F. A. (2010). *Value-Based Planning for Teams of Agents in Stochastic Partially Observable Environments*. Ph.D. thesis, Informatics Institute, University of Amsterdam.

Oliehoek, F. A., Spaan, M. T. J., & Vlassis, N. (2008a). Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, *32*, 289–353.

Oliehoek, F. A., Spaan, M. T. J., Whiteson, S., & Vlassis, N. A. (2008b). Exploiting locality of interaction in factored Dec-POMDPs. In *Autonomous Agents and Multiagent Systems (AAMAS-08)*, (pp. 517–524).

Osentoski, S., & Mahadevan, S. (2007). Learning state-action basis functions for hierarchical MDPs. In *ICML '07: international conference on Machine learning*, (pp. 705–712).

Papadimitriou, C., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Papadimitriou, C. M. (1994). *Computational Complexity*. Reading, Massachusetts: Addison-Wesley.

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, *40*(11), 38–45.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Peshkin, L., Kim, K.-E., Meuleau, N., & Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *UAI*, (pp. 489–496).

Petcu, A., & Faltings, B. (2005). Dpop: A scalable method for multiagent constraint optimization. In *IJCAI 05*, (pp. 266–271). Edinburgh, Scotland.

Petrik, M., & Zilberstein, S. (2009). A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, *35*, 235–274.

Pineau, J., Gordon, G., & Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, *27*, 2006.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer.

Poupart, P., Boutilier, C., Patrascu, R., & Schuurmans, D. (2002). Piecewise linear value function approximation for factored MDPs. In *Eighteenth national conference on Artificial intelligence*, (pp. 292–299).

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Pynadath, D. V., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, *16*, 389–423.

Rabinovich, Z., Goldman, C. V., & Rosenschein, J. S. (2003). The complexity of multiagent systems: the price of silence. In *Autonomous Agents and Multiagent Systems (AAMAS-03)*, (pp. 1102–1103).

Rich, C., & Sidner, C. L. (1997). COLLAGEN: When agents collaborate with people. In *International Conference on Autonomous Agents (Agents'97)*, (pp. 284–291).

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., & Edwards, D. D. (1996). *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Seuken, S., & Zilberstein, S. (2007a). Improved memory-bounded dynamic programming for decentralized POMDPs. In *Uncertainty in Artificial Intelligence*, (pp. 344–351). Vancouver, British Columbia.

Seuken, S., & Zilberstein, S. (2007b). Memory-bounded dynamic programming for DEC-POMDPs. *IJCAI*, (pp. 2009–2015).

Seuken, S., & Zilberstein, S. (2008). Formal models and algorithms for decentralized decision making under uncertainty. *Journal of Autonomous Agents and Multiagent Systems*.

Shen, J., Becker, R., & Lesser, V. (2006). Agent Interaction in Distributed MDPs and its Implications on Complexity. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, (pp. 529–536). Japan.

Singh, S., & Cohn, D. (1998). How to dynamically merge Markov decision processes. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.) *Advances in Neural Information Processing Systems (NIPS)*, vol. 10.

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21*(5), 1071–1088.

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

Smith, S. F., Gallagher, A., & Zimmerman, T. L. (2007). Distributed management of flexible times schedules. In *Autonomous Agents and Multiagent Systems (AAMAS-07)*, (p. 74).

Spaan, M. T. J., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1-2), 181–211.

Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. *UAI*, (pp. 576–590).

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, *7*, 83–124.

Tasaki, M., Yabu, Y., Iwanuri, Y., Yokoo, M., Marecki, J., Varakantham, P., & Tambe, M. (2010). Introducing communication in Dis-POMDPs with locality of interaction. In *Journal of Web Intelligence and Agent Systems (WIAS), Vol 8, No 3*.

Tsamardinos, I., & Pollack, M. E. (2003). Efficient solution techniques for disjunctive temporal problems. *Artificial Intelligence*, *151*(1-2), 43–89.

Varakantham, P., Kwak, J., Taylor, M., Marecki, J., Scerri, P., & Tambe, M. (2009). Exploiting coordination locales in distributed POMDPs via social model shaping. *ICAPS-09*.

Varakantham, P., Marecki, J., Yabu, Y., Tambe, M., & Yokoo, M. (2007). Letting loose a spider on a network of POMDPs: Generating quality guaranteed policies. In *Autonomous Agents and Multiagent Systems (AAMAS-07)*, (pp. 817–824).

Wagner, T., Horling, B., Lesser, V., Phelps, J., & Guralnik, V. (2003). The struggle for reuse: Pros and cons of generalization in TAEMS and its impact on technology transition. *International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2003)*.

Williamson, S. A., Gerding, E. H., & Jennings, N. R. (2009). Reward shaping for valuing communications during multi-agent coordination. In *Autonomous Agents and Multiagent Systems (AAMAS-09)*, (pp. 641–648).

Witwicki, S. J., & Durfee, E. H. (2007). Commitment-driven distributed joint policy search. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007)*, (pp. 480–487). Honolulu, Hawaii.

Witwicki, S. J., & Durfee, E. H. (2009). Commitment-based service coordination. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, *3*(1), 59–87.

Witwicki, S. J., & Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *International Conference on Automated Planning and Scheduling (ICAPS-2010)*. Toronto, Canada.

Wu, J., & Durfee, E. H. (2006). Mixed-integer linear programming for transition-independent decentralized MDPs. In *international joint conference on Autonomous agents and multiagent systems (AAMAS-06)*, (pp. 1058–1060).

Wu, J., & Durfee, E. H. (2007). Solving large tæms problems efficiently by selective exploration and decomposition. In *international joint conference on Autonomous agents and multiagent systems (AAMAS-07)*, (pp. 1–8).

Wu, J., & Durfee, E. H. (2010). Resource-driven mission-phasing techniques for constrained agents in stochastic environments. *Journal of Artificial Intelligence Research*, *38*, 415–473.

Xuan, P., & Lesser, V. (1999). Incorporating Uncertainty in Agent Commitments. *International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, (pp. 57–70).

Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, *10*(5), 673–685.