PLANNING UNDER UNCERTAINTY IN COMPLEX STRUCTURED ENVIRONMENTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Carlos Ernesto Guestrin

August 2003

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Daphne Koller
Computer Science Department
Stanford University
(Principal Advisor)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Leslie Pack Kaelbling
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Benjamin Van Roy
Department of Management Science and Engineering
Stanford University

Approved for the University Committee on Graduate Studies:

Abstract

Many real-world tasks require multiple decision makers (agents) to coordinate their actions in order to achieve common long-term goals. Examples include: manufacturing systems, where managers of a factory coordinate to maximize profit; rescue robots that, after an earthquake, must safely find victims as fast as possible; or sensor networks, where multiple sensors collaborate to perform a large-scale sensing task under strict power constraints. All of these tasks require the solution of complex long-term multiagent planning problems in uncertain dynamic environments.

Factored Markov decision processes (MDPs) allow us to represent complex uncertain dynamic systems very compactly by exploiting problem-specific structure. Specifically, the state of the system is described by a set of variables that evolve stochastically over time using a compact representation called a *dynamic Bayesian network* (DBN). A DBN exploits locality by assuming that the short-term evolution of a particular variable only depends on a few other variables, *e.g.*, the state of a section of a factory is only directly affected by a few other sections. In the long-term, all variables in a DBN usually become correlated. Factored MDPs often allow for an exponential reduction in representation complexity. However, the complexity of exact solution algorithms for such MDPs grows exponentially in the number of variables, and in the number of agents.

This thesis builds a formal framework and approximate planning algorithms that exploit structure in factored MDPs to solve problems with many trillions of states and actions very efficiently. The main contributions of this thesis include:

Factored linear programs: A novel LP decomposition technique, using ideas from inference in Bayesian networks, that can exploit problem structure to reduce exponentially-large LPs to polynomially-sized ones that are provably equivalent.

Factored approximate planning: A suite of algorithms, building on our factored LP decomposition technique, that exploit structure in factored MDPs to obtain exponential reductions in planning time.

Distributed coordination: An efficient distributed multiagent decision making algorithm, where the coordination structure arises naturally from the factored representation of the system dynamics.

Coordinated reinforcement learning: A simple, yet effective, framework for designing algorithms for planning in multiagent environments, where the factored model is not known *a priori*.

Generalization in relational MDPs: A framework for obtaining general solutions from a small set of environments, allowing agents to act in new environments without replanning.

Empirical evaluation: A detailed evaluation on a variety of large-scale tasks, including multiagent coordination in a real strategic computer game, demonstrating that our formal framework yields effective plans, complex agent coordination, and successful generalization in some of the largest planning problems in the literature.

Acknowledgements

I am deeply grateful to my advisor Daphne Koller, whose thoughtful guidance, insightful vision, and continuing support have lead me to grow immensely over the last five years. Daphne's ability to select interesting problems, her high standards, and hard work are contagious, allowing her students to achieve their full potential. I am very happy to be one of these students, and I am thankful for the opportunity to learn from Daphne. Thank you!

Ron Parr has taught me more than I can describe in these short lines. Throughout my Ph.D. career, Ron has been a friend, an insightful coauthor, and a guide in a field of study that was new and uncertain. The path to my Ph.D. would have been a lot less rewarding without Ron. Jean-Claude Latombe has taught me a great deal over the last five years. I have enjoyed learning about the motion of robots, molecules, and climbers from him. I admire Jean-Claude's ability to balance research interests and personal pursuits.

I am also indebted to the other members of my Reading Committee. I am very grateful to Leslie Kaelbling, for insightful comments both in my work and in this thesis, for her tremendous ongoing help and support, and for many fun and motivating discussions. I greatly enjoyed the interactions with Ben Van Roy, whose work has shaped many developments in MDPs over the years. I would also like to thank the other members of my Orals Committee, Yoav Shoham and Claire Tomlin, for their probing and stimulating questions during my defense.

Long discussions with my friend Craig Boutilier, whose research on factored MDPs provides part of the foundation for this thesis, have significantly improved our work. Dale Schuurmans, a friend and coauthor, has inspired many new research directions. I have greatly enjoyed the opportunity to work with Geoff Gordon, whose mathematical insight is sharp and accurate. I am particularly thankful to Dirk Ormoneit, who guided me as a young student through the nuances of the mathematical foundations of planning under uncertainty. My work has greatly benefited from suggestions and kind encouragement from Tom Dietterich. I have enjoyed discussions and a photography excursion with Sebastian Thrun. I want to thank Nir Friedman for making me think harder, and barbecue better. Carlo Tomasi, who listened even at the very beginning of my career, has taught me a great deal about the technical groundwork of AI. I am thankful to Nils Nilsson for his kind and welcoming personality, and for his support.

I would not be at Stanford today if not for Eric Krotkov and Fabio Cozman, who believed in my potential, and invited me to Carnegie Mellon University in 1996. I am very grateful to Eric for this opportunity, which had such an immense positive impact on my life. I am also thankful to Fabio for teaching me a great deal about pursuing a research project and tackling a large complex problem. At CMU, I also had the great pleasure of meeting Illah Nourbakhsh, who has inspired me over the years with ideas, kind words of support, his contagious energy for life, and his faithful friendship.

I am grateful to every one of my co-authors over the last several years, from whom I learned about new research directions, open problems, solution techniques, high standards, writing style, and the fun of pursuing challenging and ambitious research goals: Serkan Apaydın, Doug Brutlag, Fabio Cozman, Chris Gearhart, Geoff Gordon, David Hsu, Neal Kanodia, Daphne Koller, Eric Krotkov, Michail Lagoudakis, Jean-Claude Latombe, Dirk Ormoneit, Ron Parr, Relu Patrascu, Dale Schuurmans, Ben Taskar, Chris Varma, Shobha Venkataraman. Many of the results in this thesis would not have been possible without the hardwork and perseverance of Shobha Venkataraman, Chris Gearhart and Neal Kanodia. Their efforts in researching new directions and applications, and creating an efficient implementation of our methods have fundamentally improved in the work in this thesis.

The work in this thesis was supported by the DoD MURI program, administered by the Office of Naval Research under Grant N00014-00-1-0637, by Air Force contract F30602-00-2-0598 under DARPA's TASK program, and by the Sloan Foundation. The author was also supported by a Siebel Scholarship and a Stanford School of Engineering Fellowship.

One of the most interesting and instructive elements of my Ph.D. career has been the interaction with the members of Daphne's research group. I have immensely enjoyed being part of one of the most collaborative and productive groups that I have seen. I am proud to have shared this experience with: Pieter Abbeel, Drago Angelov, Xavier Boyen, Urszula Chajewska, Chris Gearhart, Lise Getoor, Neal Kanodia, Uri Lerner, Uri Nodelman, Dirk Ormoneit, Ron Parr, Avi Pfeffer, Eran Segal, Christian Shelton, Ben Taskar, Simon Tong, Shobha Venkataraman, David Vickrey, Haidong Wang, and, of course, Daphne Koller.

At Stanford, I have had the pleasure of pursuing two of my non-academic passions: art and outdoor activities. I am immensely grateful for the opportunities that the Stanford Art Department has given me to learn, explore, and discover. The influences and friendship of Joel Leivick, Robert Dawson, Thom Sempere, and of the other students have forever touched my life. I am very happy to have shared my introduction into the art of photography with Fabienne Delpy. I am grateful to Bryan Palmintier and the members of the Stanford Outdoor Education Program for giving me skills to climb many peaks. Arturo Crespo, my friend and "consultant for random facts," thank you for leading me to some of the most beautiful parts of the Sierra Nevada. Kristina Woods, my partner from hot springs in Sequoia National Park, through beautiful cracks in Lover's Leap, to the rocks of the Canadian Rockies, thank you for sharing these and many other memorable experiences with me over the last few years. And, thank you for your support when I have needed the most.

Over the last few years, Ben Taskar has been a faithful friend and confidant. Thank you for your friendship, and for teaching me so much in our new joint research project. I have shared my experiences at Stanford with Kevin Leyton-Brown since the visit weekend for prospective students. I am grateful for your help and friendship over these years. I am very grateful to Tania Ogasawara for her friendship over the last 10 years. Thank you for sharing these years with me.

My life at Stanford has been significantly more gratifying, interesting and fun, because it has been filled with many smart, enthusiastic, and supportive friends. The list of people who had positive influences on my experience does not fit in these lines, but I am grateful to all of you for sharing this time with me.

I would finally like to thank my family for their endless love, support and encouragement. I am particularly grateful to my parents, sisters, brother, and grandparents. Although our life paths may have been tortuous, your care and love have been immutable over my life. Thank you with all my heart!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Decision making problems in uncertain dynamic environments arise in many real-world situations. The efficiency of a factory is optimized by the decisions of a set of managers. A good manager has “vision” and “communication skills”, that is, she can design long-term plans in collaboration with other managers. After an earthquake, members of the rescue team must coordinate their decisions to safely find victims, as fast as possible. Air traffic control routes hundreds of planes, balancing safety and speed.

All of these problems involve decision makers, or *agents*, selecting a sequence of actions in order to maximize multiple long-term goals. Additionally, uncertainty is ubiquitous in these domains, both in the effects of a decision makers’ actions, and in the evolution of the actual system. In recent years, advances in technology and algorithms have lead to increased interest in automated methods for solving each of these tasks. Commercial tools are now available for problems ranging from supply-chain management to matchmaking (another example of a complex decision-making under uncertainty problem). Unfortunately, these problems usually tend to be very large and complex, and most of the existing automated solution methods either build on heuristic procedures, or do not fully address the long-term or the uncertain aspects of these sequential decision problems.

Although such domains are very large, real-world problems usually possess large amounts of structure. As argued by Herbert Simon [1981] in “Architecture of Complexity”, many complex systems have a “nearly decomposable, hierarchical structure”, with the subsystems interacting only weakly between themselves. This is the type of structure that a human decision maker will probably exploit to solve such large-scale problems.

Briefly, this thesis focuses on building a formal framework and a suite of automated methods for exploiting problem-specific structure in order to scale up planning under uncertainty to very large, complex systems. Our framework addresses three aspects of decision making in complex dynamic systems: finding successful strategies for decision-making agents; obtaining efficient methods for coordinating the actions of multiple agents; and, generalizing strategies obtained in a set of environments to new ones without replanning. We also empirically demonstrate that our formally well-founded methods yield effective plans, complex agent coordination, and successful generalization in some of the largest planning problems in the literature.

1.1 Sequential decision making in collaborative multiagent problems

This section presents a brief, high-level, overview of the type of decision-making problems we address in this thesis.

States and actions: A sequential decision making problem is often formulated in terms of one or many *agents*, or decision makers, interacting with a *system*, and with each other. The *state* of this system is often specified by a set of *state variables* describing the state of each part of this system. In our factory example, the agents are the managers of each section of the factory. Each section is described by one or more state variables. The state of the entire factory is then specified by these variables, in addition to other state variables defining the demand, the stock levels, etc. At every time step, each agent makes an *action* choice. The dynamics of the system are then influenced by the joint action assignment of all agents.

Dynamics: We model these system dynamics in discrete time, and assume that the state at the next time step depends only on the state at the current time step and on the joint action choice of all agents. Such systems are called *Markovian*. We also formulate the dynamics of the system as *stochastic*, that is, the state evolution is uncertain, but follows some probability distribution.

Rewards: In addition, each agent has preferences over states of the system and over action choices. Specifically, the preferences of each agent are defined by a *reward function* that assigns a real value to each state and joint action of all agents. In a factory, a manager's reward function may depend positively on the overall throughput of the factory, and negatively on the maintenance cost of its section, for example.

Multiagent problems: Systems involving multiple decision makers are often called *multiagent problems*. Our factory example can be approximated as a *collaborative* setting, where the multiple agents seek to fulfill a common goal, maximizing profit. Formally, a collaborative multiagent problem is one where every agent possesses the same reward function. An alternative, more general, formulation occurs when agents have the different reward functions. This type of domain is usually called a *competitive* multiagent problem. Games such as chess are *fully competitive*, as the agents have exactly opposite preferences. More generally, decision makers in competitive settings may share some aspects of their reward functions, and choose to collaborate at some points in time. In reality, a factory is a competitive problem, as managers may have self-interested terms in their reward function, such as their own salary. The algorithms and methods in this thesis will focus on collaborative multiagent problems, and on the special case of single agent problems.

Policies: Overall, this thesis will focus on efficiently obtaining strategies, usually referred to as *policies*, that seek to maximize the long-term reward of an agent, or of many collaborating agents, interacting with a system that evolves stochastically over time.

Autonomous agents: More specifically, we focus on developing policies for *autonomous agents*, where each agent is assumed to repeat three tasks at every time step:

Sensing: the agent observes some aspect of the current state of the system;

Action selection: the agent makes a decision about its action choice for the current time step, perhaps by communicating with other agents;

Actuation: the selected action is executed, affecting the evolution of the system.

In a factory, for example, each manager (agent) observes the state of its section, and, perhaps, that of neighboring sections; the managers then negotiate a course of action; and each manager implements this action in its own section.

Full and limited observability: We assume that the true state of the system is *fully observable*, that is, the agents *collectively* observe the true state at each time step. Unfortunately, settings where each agent to needs to observe a large number of state variables are often impractical in real-world situations. In our factory example, it is infeasible to expect each manager to know the whole state of the factory before selecting its action. Thus, we seek to design algorithms that lead to *limited observability*, where, at each time step, each agent only needs to observe a small set of state variables. In our factory, we would like the manager of one section to observe the state of this section and that of only a few other sections.

Limited communication: A very complex deliberation process for action selection may lead agents to spend an unmanageable amount of time negotiating the action choice, as in many management meetings. An alternative is to use a centralized action selection procedure, where a single agent makes a global decision and transmits the appropriate action to each one of the other agents. Unfortunately, as with most centralized methods, this procedure may lead both to a communication bottleneck, and to robustness problems. We thus need efficient methods for multiagent coordination, where agents select the optimal action while only needing to communicate with a small number of other agents. We call this property *limited communication*.

MDPs, planning and reinforcement learning: A *Markov decision process* (*MDP*) is a formal mathematical framework, popularized by Bellman [1957], for modelling and solving dynamic decision-making problems. There are two more specific problem definitions fitting this MDP formalism: A *planning* problem is one where a model of the environment, *i.e.*, representations of the system dynamics and of the reward function, are known *a priori*. The objective here is to find a policy that maximizes long-term reward with respect to this model. Alternatively, a model of the environment may not be known, and the agents must optimize their policy through experimentation. This second setting is called *reinforcement learning*. This thesis will mainly focus on addressing the planning problem in large-scale environments, though we will also present new algorithms for reinforcement learning in collaborative multiagent settings.

Generalization: In many real-world settings, agents will face many environments over their lifetime, and need to obtain good strategies for each one of these environments. Often their experience with one environment will help them to perform well in another, even with minimal or no replanning. For example, a management consultant may be called to optimize the production of many factories. The experience in one factory helps the consultant design good strategies for other factories. Unfortunately, most planning methods are designed to optimize the plan of agents in a fixed environment. In this thesis, we would like to build a framework that will allow us to *generalize* solutions obtained from a set of environments to a new unseen environment, without replanning.

1.2 Exploiting problem structure to tackle the curse of dimensionality

As discussed in the previous section, the state of a system is described by an assignment to a set of state variables. Therefore the number of possible states is exponential in the number of state variables, *e.g.*, the state space of a factory is exponential in the number of sections in this factory. Bellman [1957] coined the term *curse of dimensionality* to describe this exponential relationship between the number of states and the number of variables that describe each state. Multiagent problems possess another “curse of dimensionality”: Each agent’s action can be thought as an *action variable*. The set of joint actions is thus the exponential set of assignments to these action variables. For example, the action space in a factory grows exponentially with the number of managers.

The curse of dimensionality makes the representation of the model of an MDP infeasible in systems described by many state variables, or ones involving many agents. Specifically, as the reward function assigns a value to each state and joint action, a tabular representation of this function is thus infeasible in large systems. Similarly, the transition model for taking some joint action in some state assigns a probability distribution over states in the next time step. Again, a tabular representation is infeasible, as it requires an entry for each joint assignment of a state in the current time step, action, and state in the next time step.

This problem can be addressed by exploiting structure in the problem to define a compact representation to the reward function and of the transition model. The *Bayesian networks* (BNs) framework [Pearl, 1988] allows the compact representation of exponentially-large complex probability distributions. Additionally, probabilistic inferences can often be performed very efficiently in BNs. Dean and Kanazawa [1989] extend the BN framework to allow for the compact representation of Markovian transition models, such as those used in MDPs. This compact representation of the transition model is called a *dynamic Bayesian network* (DBN).

A DBN represents the transition model exactly as a product of local factors representing the transition probabilities of each variable. Often, by exploiting conditional independence structure in the system, the factor for each variable can be represented very compactly. In our factory example, the probability distribution for the state of a particular section in the next time step may depend on the state of this section, the state of neighboring section, and the action choice of this section's manager in the current time step, but not on the state of any other section or on the action of other managers. Thus, the factor representing the dynamics of this section can be represented quite compactly. If the number of state and action variables involved in each factor are small, we can obtain an exponential reduction in space complexity. Despite the fact that the transition model represented by a DBN is factored, influence propagates from variable to variable over time, an a sparsely connected DBN can still represents complex long-term correlations. In the factory example, a manager's action only affects its section in the next time step, but this decision may have long-term consequences for the entire factory.

The reward function can often be represented compactly with a similar factorization. In this thesis, we focus on additive decompositions [Howard & Matheson, 1984]. Specifically, we assume that the global reward function can be decomposed as the sum of factors, each depending only on a small number of state and action variables. For example, the reward function of a factory can be decomposed as the income from sales, minus the sum of the maintenance costs for each section, and so on. Again, this factored representation can often be exponentially more compact than the explicit tabular one. An MDP represented using factored representations for the transition model and for the reward function is called a *factored MDP* [Boutilier *et al.*, 1995].

1.3 Approximate solutions for MDPs

Most solution algorithms for MDPs seek to find a policy that assigns an (optimal) action choice for each agent at each state. There are two typical approaches: the policy can be represented explicitly, in tabular form, or implicitly by a *value function*, which describes, for each state, the long-term reward that the agents will accumulate by starting from this state. Unfortunately, both of these representations are exponential in the number of state variables.

Factored MDPs give us a very compact representation of the MDP model using extensions of BNs. Inferences in Bayesian networks can be performed very efficiently in sparsely connected problems [Pearl, 1988; Dechter, 1999]. We may thus be tempted to believe that sparsely connected factored MDPs will allow us to obtain an optimal policy or value function efficiently. Unfortunately, even though factored MDPs give us a very compact representation for large planning problems, computing exact solutions to these problems is known to be hard [Mundhenk *et al.*, 2000; Liberatore, 2002]. Furthermore, as shown by Allender *et al.* [2002], a compact approximate solution with theoretical guarantees generally does not exist.

In order to address the exponential growth, we resort to approximate solutions to the factored MDP. There are many types of approximation methods for MDPs. Typically, these methods are divided into two main classes: *value function approximation* – methods that search in a parametric space of approximate value functions; *policy search* – methods that search in a parametric space of approximate policies. We refer the reader to books by Bertsekas and Tsitsiklis [1996], or Sutton and Barto [1998] for a more in-depth discussion.

This thesis focuses mainly on *linear value function approximation*, a form of *linear regression*, as first proposed by Bellman *et al.* [1963]. Here the value function is approximated by a linear combination of (potentially non-linear) basis functions, or features. Unlike many policy search methods and more complex value function approximation architectures, the parameters of such a linear approximation can often be estimated effectively by stable global optimization procedures.

In the context of factored MDPs, Koller and Parr [1999] suggest a specific type of basis function that is particularly compatible with the structure of the factored model. They suggest that, although the value function is typically not structured, there are many cases where it might be “close” to structured, that is, where the value function is well-approximated using a linear combination of functions, each of which refers only to a small number of state variables. They call such approximation architecture a *factored (linear) value function*. This representation of the value function is a central element in our efficient approximate solution algorithms, in our distributed multiagent coordination methods, and in our generalization approach.

1.4 Main contributions

We have now set the basic foundation for the work in this thesis: We seek to find efficient approximate solutions to large-scale stochastic planning problems that can be represented compactly by exploiting problem structure in factored MDPs. Specifically, we approximate the value function of such MDPs using an appropriate linear architecture, the factored value function representation, where the each basis function is restricted to depend on a small set of variables.

Efficient planning: We propose a suite of approximate planning algorithms that can exploit problem structure to optimize the basis function weights efficiently. One of these algorithms, for example, relies on a linear program (LP) formulation for optimizing the basis function weights that was first proposed by Schweitzer and Seidmann [1985]. Unfortunately, this LP formulation has one constraint for each state and joint action, a thus exponentially-large set of constraints. We address this problem by proposing the *factored LP* algorithm, a novel LP decomposition technique, which allows us to exploit structure in the factored MDP and in the factored value function to represent this exponentially-large set of constraints by a provably equivalent polynomial formulation. The complexity of our algorithm will depend explicitly on the sparseness of the interactions between state variables in the factored MDP, and on the complexity of the structure of our factored value function. Factored LPs are a core element to all of our efficient solution algorithms, both in the single agent, and in the multiagent settings.

Multiagent coordination: In collaborative multiagent settings, at every time step, agents must choose the action that maximizes the value function, among an exponential set of possible action choices. This action selection problem is generally intractable, and requires a centralized decision-making procedure. We present a novel distributed algorithm for action selection in collaborative multiagent settings. This algorithm is able to chose the action that optimally maximizes our approximate value function by exploiting problem structure. Additionally, our approach fulfills two important properties described earlier in this chapter: limited observability and limited communication. Interestingly, the communication structure between agents in our algorithm is not imposed *a priori*, but derived directly from the structure in the factored MDP and value function.

Coordinated reinforcement learning: Thus far, we have assumed that the system we are tackling has been modelled by a factored MDP. In many practical problems, this model is not known *a priori*, agents must learn effective policies through their interactions with the environment. We demonstrate that many existing RL algorithms that have been successfully applied to single agent problems can be generalized to collaborative multiagent settings by applying simple extensions of our factored value function representation, along with our multiagent coordination algorithm.

Context-specific structure: Thus far, we have assumed that the factored MDP dynamic model is composed of state variables whose evolution depends only on few other variables in the system. Unfortunately, this assumption is often incorrect. A state variable in the next time step may *potentially* depend on many other variables in the current time step, but usually not at the same time. In our factory example, a section may receive parts from any other section, thus potentially correlating the state of all sections. However, the current work order defines the specific parts that a section will need, and thus the particular sections that will influence the state of each section of the factory in the next time step. This type of structure can exploited in the representation by using the notion of *context-specific independence* (CSI) [Boutilier *et al.*, 1995], where the correlation between variables may depend on the specific context at hand. We extend our factored LP decomposition technique to allow us to design efficient planning algorithms that can exploit both the additive structure present in the standard factored MDP formulation, and context-specific structure to obtain approximate solutions to highly connected, structured problems.

Variable coordination structure: In addition to increasing the efficiency of our planning algorithm, CSI allows us to address a very important shortcoming of the multiagent coordination formulation described above. Our standard algorithm allows agents to compute the maximizing action while only communicating with a few other agents, but always with the same agents. However, if we consider our factory example, we realize that managers usually only need to communicate with the managers of sections involved in the current work order. Thus, the communication structure should not be fixed, but rather change with the state of the system. Interestingly, by exploiting both the additive structure in the factored MDP and CSI, we are able to design a multiagent coordination algorithm where the communication structure will no longer be fixed, but varies naturally with the state of the system. Again, this (varying) structure is not defined *a priori*, but derived directly from the structure in the model and in the value function.

Generalization: As discussed above, in many real-world situations, the agents will often be faced with many environments, and experiences from some environments should allow these agents to perform well in other ones. Such a generalization allows us to tackle new environments with minimal or no replanning, and also gives us a methodology for obtaining good strategies for extremely large environments that could not be solved even with our factored techniques. However, it is not clear how policies for one MDP could be mapped to other ones. Each MDP is different, having a different number of states, actions, a different reward function, transition model, etc. To address this problem we propose the framework of *relational MDPs*, based on the *probabilistic relational model* (PRM) framework of Koller and Pfeffer [1998]. In a relational MDP, an environment is represented by a sets of related objects of different classes, where the transition model of one objects depends only on the states of related objects. For example, a modern factory is often organized in terms of many manufacturing cells. Each cell is of one of a few "types", or classes in our relational model, *e.g.*, lathes, painting, etc. The flow plan of parts between these cells defines the relations between these objects, and specifies the dynamics of the overall factory.

Such a relational representation allows us to represent a whole class of similar environments very compactly. Specifically, we can instantiate a particular environment by specifying the number of objects of each class, and the relations between them. For each particular environment we can apply our factored planning algorithms and distributed multiagent coordination approach to perform planning and action selection very efficiently. However, if we need to replan in every new environment, we have not achieved the any type of generalization. To address this issue, we propose a relational representation for the factored value function. Here our basis functions are represented in terms of classes of objects. We present a new LP formulation that allows us to find the weights of this class-level approximate value function by only considering a small set of sampled environments. We prove that, by optimizing over a polynomial number of sampled "small" environments, we obtain a class-level value function that is close to the one we would obtain had we considered all, possibly unboundedly-large, environments. Once we have obtained these weights, we can instantiate our class-level value function in any new environment, thus allowing us to generalize the results from a few sampled environments to new ones without any replanning.

1.5 Examples

There are many real-world problems that require the solution of sequential decision making problems. Puterman [1994], Bertsekas and Tsitsiklis [1996], and Sutton and Barto [1998] present many case studies. In particular, Puterman, [1994, Chapter 1], describes several problems in industry and science that were solved by automated algorithms, along with estimates of the large sums of money that were saved in the long-run. In this section, we present some intuitions about the types of structure we address and about the scope of this thesis by describing some abstractions of complex practical problems.

Manufacturing system: Optimization procedures have been applied to many areas of manufacturing. Consider, for example, the problem scheduling maintenance in sections of a large factory. As described in this chapter, we could model the dynamics of such a factory, albeit in an abstracted fashion, using a factored representation. Here we would also have an action variable for each section of the factory indicating whether this section should undergo maintenance in the next time step. The reward function will be factored additively according to sections of the factory indicating the production of each section minus its maintenance cost. The complexity of the representation of the factored value function depends on the particular problem at hand. We could, for example, include basis functions over pairs of connected sections in the factory. Using such a model our algorithms will give us a policy for scheduling maintenance that attempts to maximize the global reward of the entire factory. As described above, we can also use a relational representation for this type of problem. This representation would allow us to generalize from a few small factories to significantly larger ones.

Queueing networks: Queueing problems are a special type of stochastic dynamic system, where an agent who manages a set of queues of jobs must decide which one to serve at every time step. These problems have been widely studied in the literature, as they provide abstractions of many practical problems in industry. Queueing networks [Bolch *et al.*, 1998] are an extension of this model to problems involving many agents (servers) simultaneously. The network defines a process where jobs that are served in one queue are then assigned to another one. We can view this process as a factored MDP with a state variable to represent the length of each queue, and an action variable representing the action of each server. The network defines the interactions in the factored MDP, as the state of a particular queue in the next time step depends only on the queues that feed jobs into this one. The reward function is also factored, defined, for example, as the number of jobs that terminate in each queue. Given this factored representation, our framework will give us efficient algorithms for obtaining policies that coordinate the actions of these servers in order to approximately minimize the overall wait in the system. Again, a relational representation could be effective in this setting. Here large complex networks can often be composed of similar subnetworks, where each subnet is chosen from a few classes of subnets.

Computer games: In recent years, there has been a significant increase in interest in the applications of AI techniques to computer games [Laird & Van Lent, 2001]. In Chapter 13, we present an application of our factored techniques to a strategic war game called *Freecraft* [Freecraft, 2003], an open-source version of the popular Warcraft®game. The objective of this game is to coordinate the actions of a set of units with different skills in order to defeat an enemy force. Here, we use relational MDPs to represent possible Freecraft scenarios. Each unit is an instance of one of a few classes, including peasants, footmen, enemies, etc. Our agents, the units we control, receive a reward for each dead enemy, thus the reward function is additive, with one term for each enemy. The transition model also decomposes according to the interactions in the game, *e.g.*, the state of an enemy depends on the footmen that are attacking it. A relational representation for this domain would include a class of objects for each type of unit in the game. Our class-level value function could include terms between objects of class footman and those of class enemy. In Section 13.5.2, we show that this type of class-level value function allows us to generalize solutions effectively to very large Freecraft scenarios that could not be solved even by our factored planning techniques.

Networking: There are many possible applications of planning algorithms in networking tasks (*e.g.*, packet routing [Boyan & Littman, 1993]). An interesting potential application is the routing of queries in peer-to-peer systems [Crespo & Garcia-Molina, 2002], such as the popular music sharing software *Gnutella*. We can consider each node as an agent that can decide to fulfill a query, or forward it to one of the neighboring nodes. The state of this system is specified by the information (*e.g.,* songs) stored in each node and the query. A node cannot observe (or even store) the state of every node in the network, and should not flood the entire system with queries. Using our approach, we could tackle such problems effectively, requiring only limited observability and limited communication between the nodes.

Elevator scheduling: A building with a large number of floors requires an effective elevator scheduling policy to avoid long waits. This optimization problem requires long-term planning under uncertainty [Crites & Barto, 1996]. We can view the number of passengers waiting at each floor and in each elevator, along with the current floor of each elevator and the requested stops, as our state variables. There is one action variable for each elevator in this model, indicating the elevator's next destination. One could imagine a formulation of this problem as a factored MDP, where the state of each elevator only depends on its actions and the number of passengers in the current floor. The reward function can be represented additively as the number of people waiting on each floor and in each elevator. In most practical systems, however, rather than observing the number of people waiting on each floor, the elevators receive a signal of whether there are any passengers waiting at each floor. The problem is thus no longer fully observable, breaking the assumptions of our models. Typically, this partial observability issue is addressed by assuming full observability of the number of passengers on each floor, or by considering that an "average number" of people are waiting every time the elevator is called. Under these assumptions, our algorithms could also be applied to the elevator scheduling task.

Sensor networks: Estrin *et al.* [1999] define networked sensors as "those that coordinate among themselves to achieve a larger sensing task". We can even extend this definition to include actuation, leading to large-scale distributed planning problems, such as the ones addressed in this thesis. Actuation in such systems encompasses not only standard effectors in the environment, but also decisions over when to communicate information to other sensors, and when to sense the environment, thus maximizing the amount of information gathered, while bounding power consumption. We could use the factored MDP framework to model such systems, where the interactions between sensors would be fixed in static environments, and varying in environments where the location of sensors changes over time. Our relational framework could be particularly useful in such tasks, allowing us to generalize from environments with a manageable number of sensors, to very complex environments involving a very large set of devices.

1.6 The Thesis

The remainder of this thesis is organized as follows:

Chapter 2: We first present a brief review of the MDP model, and some exact and approximate solution algorithms, including *LP-based approximation* and *approximate policy iteration*. We also extend approximate policy iteration to utilize projections in max-norm that are compatible with existing theoretical analyzes.

Chapter 3: We review the factored MDP model, along with the factored value function approximation architecture and some initial basic operations required by our algorithms.

Chapter 4: We describe our novel factored LP decomposition technique, which allows us to exploit problem structure to solve LPs with exponentially-large constraint set very efficiently.

Chapter 5: We present our efficient approximate planning algorithms for single agent problems. By building on our factored LP algorithm, we design factored versions of the LP-based approximation algorithm and of approximate policy iteration with projections in max-norm. We also present an empirical evaluation of the scaling properties, and of the quality of the policies generated by these two approaches.

Chapter 6: We consider the dual formulation of our LP-based approximation algorithm for factored MDPs. This new formulation allows us find approximate solutions in highly connected problems that could not be solved by our factored LP decomposition technique.

Chapter 7: We extend our approximate solution algorithms to problems with context-specific structure, thus allowing us to exploit both additive structure and CSI. The empirical evaluation in this chapter includes comparisons to existing state-of-the-art methods of Boutilier *et al.* [1995] and Hoey *et al.* [1999].

Chapter 8: We review the basic extension of factored MDPs to problems involving multiple collaborating agents. We then present a straightforward extension of the basic factored value function representation to such problems.

Chapter 9: We present our new distributed multiagent coordination algorithm, which allows agents with limited observability and communication to select a maximizing joint action for each state. We also extend our basic factored LP-based planning algorithm to the multiagent setting. We present empirical evaluations demonstrating the polynomial scaling property for problems with fixed induced width, and comparing our algorithms to other state-of-the-art methods.

Chapter 10: We show that, by extending our factored multiagent approach to problems with context-specific structure, we obtain a new coordination algorithm, where the coordination structure naturally changes with state of the system. We also empirically verify that this algorithm yields highly dynamic coordination structures and effective policies.

Chapter 11: We describe *coordinated reinforcement learning*, a framework that leverages on our multiagent coordination algorithm to allow us to extend many existing RL solution methods to collaborative multiagent settings. We present empirical comparisons of our coordinated RL method and some existing state-of-the-art approaches.

Chapter 12: We introduce the new framework of relational MDPs, where both the MDP model and the factored value function of domain are represented in terms of related objects of various classes.

Chapter 13: We describe a new algorithm for optimizing the weights of the class-level value function over a set of environments. We also prove that by sampling a polynomial number of "small" environments we obtain a class-based value function that is close to the one we would obtain had we considered all worlds in our optimization. We present empirical evaluations of our generalization algorithm for relational MDPs, both on simulated environments, and on a real strategic computer war game.

Chapter 14: We summarize the algorithms and main contributions of this thesis. We finally conclude a discussion of future directions and open problems.

Our factored LP algorithm was first presented by Guestrin, Koller and Parr in [Guestrin *et al.*, 2001a], along with the factored approximate iteration algorithm using max-norm projections. Guestrin, Koller and Parr describe the multiagent coordination algorithm along with the factored version of the LP-based approximation algorithm for both single and multiagent problems, in [Guestrin *et al.*, 2001b]. The dual factorization method in Chapter 6 is new, and has not yet been published in the literature. The extension of our algorithm to exploit both additive and context-specific structure was presented by Guestrin, Venkataraman and Koller in [Guestrin *et al.*, 2002d], who also describe the resulting variable coordination structure in multiagent problems. Guestrin, Koller, Parr and Venkataraman describe all of our single agent methods in an unified presentation, in [Guestrin *et al.*, 2002a]. Guestrin, Lagoudakis, and Parr present the coordinated reinforcement learning framework, in [Guestrin *et al.*, 2002b]. Finally, the relational MDP representation, the generalization algorithm to new unseen problems, and the experimental results on the real strategic computer war game were described by Guestrin, Koller, Gearhart and Kanodia in [Guestrin *et al.*, 2003].

Part I

Basic models and tools

Chapter 2

Planning under uncertainty

A Markov decision process (MDP) is a mathematical framework for sequential decision making problems in stochastic domains. MDPs thus provide underlying semantics for the task of planning under uncertainty. We present only a concise overview of the MDP framework here, referring the reader to the books by Bertsekas and Tsitsiklis [1996], Puterman [1994], or Sutton and Barto [1998] for a more in-depth review.

2.1 Markov decision processes

A *Markov decision process (MDP)* $\mathcal{M}$ is defined as a 4-tuple $\mathcal{M} = (\mathbf{X}, A, R, P)$ where: $\mathbf{X}$ is a finite set of $|\mathbf{X}| = N$ states; A is a finite set of actions; R is a *reward function* $R : \mathbf{X} \times A \mapsto \mathbb{R}$, such that $R(\mathbf{x}, a)$ represents the reward obtained by the agent in state $\mathbf{x}$ after taking action a; and P is a *Markovian transition model* where $P(\mathbf{x}' \mid \mathbf{x}, a)$ represents the probability of going from state $\mathbf{x}$ to state $\mathbf{x}'$ after taking action a. We assume that the rewards are bounded, that is, there exists R_{max} such that $R_{max} \geq |R(\mathbf{x}, a)|, \forall \mathbf{x}, a$.

Example 2.1.1 *Consider the problem of optimizing the behavior of a system administrator (SysAdmin) maintaining a network of m computers. In this network, each machine is connected to some subset of the other machines. Various possible network topologies can be defined in this manner (see Figure 2.1 for some examples). In one simple network, we might connect the machines in a ring, with machine i connected to machines $i+1$ and $i-1$. (In this example, we assume addition and subtraction are performed modulo m.)*

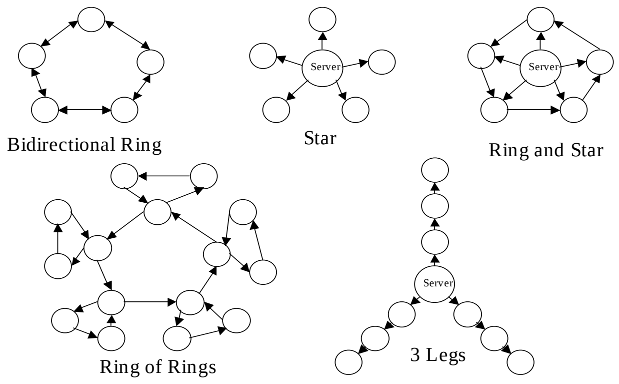


Figure 2.1: Network topologies tested; the status of a machine is influence by the status of its parent in the network.

Each machine is associated with a binary random variable X_i, representing whether it is working or has failed. At every time step, the SysAdmin receives a certain amount of money (reward) for each working machine. The job of the SysAdmin is to decide which machine to reboot; thus, there are $m + 1$ possible actions at each time step: reboot one of the m machines or do nothing (only one machine can be rebooted per time step). If a machine is rebooted, it will be working with high probability at the next time step. Every machine has a small probability of failing at each time step. However, if a neighboring machine fails, this probability increases dramatically. These failure probabilities define the transition model $P(\mathbf{x}' \mid \mathbf{x}, a)$, where $\mathbf{x}$ is a particular assignment describing which machines are working or have failed in the current time step, a is the SysAdmin's choice of machine to reboot and $\mathbf{x}'$ is the resulting state in the next time step. ▮

A *stationary (deterministic) policy* π for an MDP is a mapping $\pi : \mathbf{X} \mapsto A$, where $\pi(\mathbf{x})$ is the action the agent takes at state $\mathbf{x}$. In the SysAdmin problem, for each possible configuration of working and failing machines, the policy would tell the SysAdmin which machine to reboot. A *stationary randomized policy*, also known as a stochastic policy, ρ is a mapping from a state $\mathbf{x}$ to a probability distribution over the actions the agent may take at this state. We denote the probability of taking action a at state $\mathbf{x}$ by $\rho(a \mid \mathbf{x})$. For all MDPs, there exists at least one optimal policy which is stationary and deterministic [Puterman, 1994].

In this thesis, we assume that the MDP has an infinite horizon and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$.[1] Each policy is associated with a *value function* $\mathcal{V}_\pi \in \mathbb{R}^N$, where $\mathcal{V}_\pi(\mathbf{x})$ is the discounted cumulative value that the agent gets if it starts at state $\mathbf{x}$ and follows policy π. More precisely, the value $\mathcal{V}_\pi$ of a state $\mathbf{x}$ under policy π is given by:

$$\mathcal{V}_\pi(\mathbf{x}) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R\left(\mathbf{X}^{(t)}, \pi(\mathbf{X}^{(t)})\right) \middle| \mathbf{x}^{(0)} = \mathbf{x} \right],$$

where $\mathbf{X}^{(t)}$ is a random variable representing the state of the system after t steps. In our running example, the value function represents how much money the SysAdmin expects to collect if she starts acting according to π when the network is at state $\mathbf{x}$.

The value function for a fixed policy is the fixed point of a set of linear equations that define the value of a state in terms of the value of its possible successor states. More formally, we define:

Definition 2.1.2 (DP operator) *The* DP operator, $\mathcal{T}_\pi$, *for a stationary policy* π *is:*

$$\mathcal{T}_\pi \mathcal{V}(\mathbf{x}) = R_\pi(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} P_\pi(\mathbf{x}' \mid \mathbf{x}) \mathcal{V}(\mathbf{x}'),$$

where $R_\pi(\mathbf{x}) = R(\mathbf{x}, \pi(\mathbf{x}))$ *and* $P_\pi(\mathbf{x}' \mid \mathbf{x}) = P(\mathbf{x}' \mid \mathbf{x}, \pi(\mathbf{x}))$. *The value function of policy* π, $\mathcal{V}_\pi$, *is the fixed point of the* $\mathcal{T}_\pi$ *operator:* $\mathcal{V}_\pi = \mathcal{T}_\pi \mathcal{V}_\pi$. ▮

The optimal value function $\mathcal{V}^*$ describes the optimal value the agent can achieve for each starting state. $\mathcal{V}^*$ is defined by a set of *non-linear* equations. In this case, the value of a state must be the maximal expected value achievable by any policy starting at that state. More precisely, we define:

Definition 2.1.3 (Bellman operator) *The* Bellman operator, $\mathcal{T}^*$, *is:*

$$\mathcal{T}^* \mathcal{V}(\mathbf{x}) = \max_a [R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \mathcal{V}(\mathbf{x}')].$$

[1]Most of the results and algorithms we present have straightforward generalizations to other optimality criteria, such as long-term average reward [Puterman, 1994].

The optimal value function $\mathcal{V}^*$ is the fixed point of $\mathcal{T}^*$: $\mathcal{V}^* = \mathcal{T}^*\mathcal{V}^*$. ∎

For any value function $\mathcal{V}$, we can define the policy obtained by acting greedily relative to $\mathcal{V}$. In other words, at each state, the agent takes the action that maximizes the one-step utility, assuming that $\mathcal{V}$ represents our long-term utility achieved at the next state. More precisely, we define:

$$\mathsf{Greedy}[\mathcal{V}](\mathbf{x}) = \arg\max_a [R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a)\mathcal{V}(\mathbf{x}')]. \tag{2.1}$$

It is useful to define a Q-function, $Q_a(\mathbf{x})$, which represents the expected value the agent obtains after taking action a at the current time step and receiving a long-term value $\mathcal{V}$ thereafter. This Q function can be computed by:

$$Q_a(\mathbf{x}) = R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a)\mathcal{V}(\mathbf{x}). \tag{2.2}$$

That is, $Q_a(\mathbf{x})$ is given by the current reward plus the discounted expected future value. Using this notation, we can express the greedy policy as: $\mathsf{Greedy}[\mathcal{V}](\mathbf{x}) = \max_a Q_a(\mathbf{x})$.

The greedy policy relative to the optimal value function $\mathcal{V}^*$ is the optimal policy:

$$\pi^* = \mathsf{Greedy}[\mathcal{V}^*]. \tag{2.3}$$

Often, we can only obtain an approximation $\widehat{\mathcal{V}}$ of the optimal value function $\mathcal{V}^*$. In this case, our policy will be the suboptimal $\widehat{\pi} = \mathsf{Greedy}[\widehat{\mathcal{V}}]$, rather than the optimal one π^*. Williams and Baird [1993] present a bound on the loss of acting according to $\widehat{\pi}$ from a bound on the approximation quality of $\widehat{\mathcal{V}}$ called the *Bellman error*:

Definition 2.1.4 (Bellman error) *The* Bellman error *of a value function $\mathcal{V}$ is defined as:*

$$\mathrm{BellmanErr}(\mathcal{V}) = \left\|\mathcal{T}^*\mathcal{V} - \mathcal{V}\right\|_\infty ,$$

where for any vector $\mathcal{V}$, the max-norm is given by: $\left\|\mathcal{V}\right\|_\infty = \max_{\mathbf{x}} |\mathcal{V}(\mathbf{x})|$. ∎

Using this measure, Williams and Baird obtain the bound:

Theorem 2.1.5 (Williams & Baird, 1993) *For any value function estimate $\widehat{\mathcal{V}}$, with a greedy policy $\widehat{\pi} = \textsf{Greedy}[\widehat{\mathcal{V}}]$, the loss of acting according to $\widehat{\pi}$ instead of the optimal policy π^* is bounded by:*

$$\mathcal{V}^*(\mathbf{x}) - \mathcal{V}_{\widehat{\pi}}(\mathbf{x}) \leq \frac{2\gamma \mathrm{BellmanErr}(\widehat{\mathcal{V}})}{1-\gamma} \ , \ \forall \mathbf{x} \ ,$$

where $\mathcal{V}^*$ is the value of the optimal policy π^* and $\mathcal{V}_{\widehat{\pi}}$ is the actual value of acting according to the suboptimal policy $\widehat{\pi}$. ▮

2.2 Solving MDPs

There are several algorithms to compute the optimal policy in an MDP. The three most commonly used are linear programming, value iteration, and policy iteration. A key component in all three algorithms is the computation of value functions, as defined in Section 2.1. Recall that a value function defines a value for each state $\mathbf{x}$ in the state space. With an explicit representation of value functions as a vector of values for the different states, the solution algorithms all can be implemented as a series of simple algebraic steps. Once the optimal value function $\mathcal{V}^*$ is computed, the optimal policy π^* is simply the greedy policy with respect to $\mathcal{V}^*$ as defined in Equation (2.3).

2.2.1 Linear programming

Linear programming (LP) provides a simple and effective solution method for finding the optimal value function for an MDP. In the formulation first proposed by Manne [1960], the LP variables are $V(\mathbf{x})$ for each state $\mathbf{x}$, where $V(\mathbf{x})$ represents the value of starting at state $\mathbf{x}$, *i.e.*, $\mathcal{V}(\mathbf{x})$. The LP is given by:

$$\begin{array}{ll} \text{Variables:} & V(\mathbf{x}) \ , \ \forall \mathbf{x} \ ; \\ \text{Minimize:} & \sum_{\mathbf{x}} \alpha(\mathbf{x}) \ V(\mathbf{x}) \ ; \\ \text{Subject to:} & V(\mathbf{x}) \geq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) V(\mathbf{x}') \ , \quad \forall \mathbf{x} \in \mathbf{X}, a \in A \ ; \end{array} \tag{2.4}$$

where the *state relevance weights* α are positive ($\alpha(\mathbf{x}) > 0, \forall \mathbf{x}$), and, usually, normalized to sum to one ($\sum_{\mathbf{x}} \alpha(\mathbf{x}) = 1$). Interestingly, the optimal solution obtained by this LP is the same for any positive weight vector. Intuitively, the constraints enforce that $V(\mathbf{x})$ is greater than or equal to $\max_a R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) V(\mathbf{x}')$. By minimizing $\sum_{\mathbf{x}} \alpha(\mathbf{x})\, V(\mathbf{x})$, the LP forces equality for the maximum value of the righthand side, thus enforcing the Bellman equations.

It is useful to understand the dual of the LP in (2.4).

$$\begin{array}{lll} \text{Variables:} & \phi_a(\mathbf{x})\ ,\ \forall \mathbf{x}\, \forall a\ ; & \\ \text{Maximize:} & \sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a)\ ; & \\ \text{Subject to:} & \sum_a \phi_a(\mathbf{x}) = \alpha(\mathbf{x}) + \gamma \sum_a \sum_{\mathbf{x}'} P(\mathbf{x} \mid \mathbf{x}', a) \phi_a(\mathbf{x}')\ , & \forall \mathbf{x} \in \mathbf{X}\ ; \\ & \phi_a(\mathbf{x}) \geq 0\ , & \forall \mathbf{x} \in \mathbf{X}, a \in A\ . \end{array} \tag{2.5}$$

In this dual LP, the variable $\phi_a(\mathbf{x})$, called the *visitation frequency* for state $\mathbf{x}$ and action a, can be interpreted as the expected number of times that $\mathbf{x}$ will be visited and action a executed in this state (discounted so that future visits count less than present ones), where α is the starting state distribution. The constraints in (2.5) are thus analogous to the definition of a stationary distribution in a Markov chain (except that our frequency is now discounted).[2] Specifically, a constraint for a state $\mathbf{x}$ forces the total visitation frequency for this state, $\sum_a \phi_a(\mathbf{x})$, to be equal to the probability of starting at this state, $\alpha(\mathbf{x})$, plus the discounted expected flow from all other states $\mathbf{x}'$ to this state $\mathbf{x}$ times the respective visitation frequencies of the origin states, $\gamma \sum_a \sum_{\mathbf{x}'} P(\mathbf{x} \mid \mathbf{x}', a) \phi_a(\mathbf{x}')$.

There is a one to one correspondence between feasible solutions to this dual LP and policies in the MDP. Specifically, there is well-defined mapping between every feasible solution and a (randomized) policy in the underlying MDP. More formally:

Theorem 2.2.1

1. *Let ρ be any stationary randomized policy, then if:*

$$\phi_a^{\rho}(\mathbf{x}) = \sum_{t=0}^{\infty} \sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}) P_{\rho}(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}'),\ \forall \mathbf{x}, a\ , \tag{2.6}$$

[2] This relationship becomes very precise if (rather than discounted) the average reward optimality criteria is used. In this case, the constraints become exactly the stationary distribution constraints [Puterman, 1994].

where $P_\rho(\mathbf{x}' \mid \mathbf{x}) = \sum_a P(\mathbf{x}' \mid \mathbf{x}, a)\rho(a \mid \mathbf{x})$*, then* ϕ_a^ρ *is a feasible solution to the dual LP in (2.5).*

2. *If* ϕ_a *is a feasible solution to the dual LP in (2.5), then for all states* $\mathbf{x}$*,* $\sum_a \phi_a(\mathbf{x}) > 0$*. Furthermore, define a randomized policy* ρ *by:*

$$\rho(a \mid \mathbf{x}) = \frac{\phi_a(\mathbf{x})}{\sum_a \phi_a(\mathbf{x})}. \tag{2.7}$$

Then the dual solution defined by $\phi_a^\rho(\mathbf{x})$ *as in Equation (2.6) is a feasible solution to the dual LP in (2.5), and* $\phi_a^\rho(\mathbf{x}) = \phi_a(\mathbf{x})$ *for all* $\mathbf{x}$ *and* a*.*

3. *A deterministic policy* π^* *is optimal if and only if* $\phi_a^{\pi^*}$ *is an optimal basic feasible solution to the dual LP in (2.5).*

4. *The dual linear program has the same optimal basis for any positive weight vector* α*. Thus, both* $\phi_a^{\pi^*}$ *and* π^* *do not depend on* α*.*

Proof: *see, for example, the book by Puterman [1994].* ▮

Now consider the objective function of the dual LP in (2.5). By substituting the result in Equation (2.6), we obtain:

$$\begin{aligned}\sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a) &= \sum_a \sum_{\mathbf{x}} \sum_{t=0}^{\infty} \sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}) P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}') R(\mathbf{x}, a); \\ &= \sum_{\mathbf{x}'} \alpha(\mathbf{x}') E_\rho \left[\sum_{t=0}^{\infty} \gamma^t R_\rho\left(\mathbf{x}^{(t)}\right) \middle| \mathbf{x}^{(0)} = \mathbf{x}' \right].\end{aligned}$$

That is, the objective of the dual LP in (2.5) is to maximize total reward for all actions executed, and the state relevance weights α represent the starting state distribution. It is again surprising that the solution does not depend on the value of α. This property will not hold for the approximate version of this algorithm.

2.2.2 Value iteration

Value iteration is a commonly used alternative approach for solving MDPs [Bellman, 1957]. This algorithm, shown in Figure 2.2, starts from any initial estimate $\mathcal{V}^{(0)}$ of the

```
VALUEITERATION ($P$, $R$, $\gamma$, $\mathcal{V}^{(0)}$, $\varepsilon$, $t_{max}$)
        // $P$ – transition model.
        // $R$ – reward function.
        // $\gamma$ – discount factor.
        // $\mathcal{V}^{(0)}$ – any initial estimate of the value function.
        // $\varepsilon$ – Bellman error precision.
        // $t_{max}$ – maximum number of iterations.
        // Return near-optimal value function.
    LET $t = 0$.
    REPEAT
        LET:
            $\mathcal{V}^{(t+1)}(\mathbf{x}) = \mathcal{T}^* \mathcal{V}^{(t)}(\mathbf{x}) = \max_a \left[ R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \mathcal{V}(\mathbf{x}') \right], \ \forall \mathbf{x}$ .
        LET $t = t + 1$.
    UNTIL $\mathrm{BellmanErr}(\mathcal{V}^{(t+1)}) \leq \varepsilon$ OR $t \geq t_{max}$.
    RETURN $\mathcal{V}^{(t+1)}$.
```

Figure 2.2: Value iteration algorithm.

value function. This estimate is iteratively improved through repeated applications of the Bellman operator. The convergence of this algorithm relies on the max-norm *contraction* property of the Bellman operator:

Definition 2.2.2 (contraction mapping) *An operator $\mathcal{T}$ is said to be a* contraction mapping *in norm $\|\cdot\|$, with factor $\gamma > 0$, if for any two vectors $\mathcal{V}_1$ and $\mathcal{V}_2$:*

$$\|\mathcal{T}\mathcal{V}_1 - \mathcal{T}\mathcal{V}_2\| \leq \gamma \|\mathcal{V}_1 - \mathcal{V}_2\|_\infty . \quad ▮$$

The Bellman operator is a max-norm contraction:

Theorem 2.2.3 *The Bellman operator $\mathcal{T}^*$ and the DP operator $\mathcal{T}_\pi$ are max-norm contraction mappings with factor γ.*
Proof: *see, for example, the book by Puterman [1994].* ▮

A corollary of this theorem is the convergence of value iteration:

Corollary 2.2.4

1. *The Bellman operator has an unique fixed point, i.e., $\mathcal{V}^* = \mathcal{T}^*\mathcal{V}^*$.*

```
POLICYITERATION ($P$, $R$, $\gamma$, $\mathcal{V}^{(0)}$, $\varepsilon$, $t_{max}$)
        // $P$ – transition model.
        // $R$ – reward function.
        // $\gamma$ – discount factor.
        // $\pi^{(0)}$ – any initial policy.
        // $\varepsilon$ – Bellman error precision.
        // $t_{max}$ – maximum number of iterations.
        // Return (near-)optimal policy.
    LET $t = 0$.
    REPEAT
        // Value determination step.
        COMPUTE VALUE OF POLICY $\pi^{(t)}$ BY A SOLVING LINEAR SYSTEM OF EQUATIONS:
            $\mathcal{V}_{\pi^{(t)}}(\mathbf{x}) = R_{\pi^{(t)}}(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} P_{\pi^{(t)}}(\mathbf{x}' \mid \mathbf{x}) \mathcal{V}_{\pi^{(t)}}(\mathbf{x}), \ \forall \mathbf{x}$ .
        // Policy improvement step.
        LET $\pi^{(t+1)} = \mathsf{GREEDY}[\mathcal{V}_{\pi^{(t)}}]$.
        LET $t = t + 1$.
    UNTIL $\pi^{(t)} = \pi^{(t+1)}$ OR $\text{BellmanErr}(\mathcal{V}^{(t+1)}) \leq \varepsilon$ OR $t \geq t_{max}$.
    RETURN $\pi^{(t+1)}$.
```

Figure 2.3: Policy iteration algorithm.

2. *For any* $\mathcal{V}$, $(\mathcal{T}^*)^\infty \mathcal{V} = \mathcal{V}^*$.

3. *Value iteration converges to* $\mathcal{V}^*$. ▮

Note that, as $\mathcal{T}_\pi$ is equivalent to $\mathcal{T}^*$ in an MDP with only one possible policy, these results also apply to the DP operator $\mathcal{T}_\pi$. In this case, value iteration would converge to $\mathcal{V}_\pi$.

2.2.3 Policy iteration

Policy iteration is a very effective algorithm for solving MDPs [Howard, 1960]. This algorithm, shown in Figure 2.3, iterates over policies, producing an improved policy at each iteration. Starting with some initial policy $\pi^{(0)}$, each iteration consists of two phases. *Value determination* computes, for a policy $\pi^{(t)}$, the value function $\mathcal{V}_{\pi^{(t)}}$. The *policy improvement* step defines the next policy as $\pi^{(t+1)} = \mathsf{Greedy}[\mathcal{V}_{\pi^{(t)}}]$.

Policy iteration is monotonic:

Theorem 2.2.5 *Let* $\pi^{(t)}$ *and* $\pi^{(t+1)}$ *be any two successive policies generated by policy iteration, then:*

$$\mathcal{V}_{\pi^{(t+1)}}(\mathbf{x}) \geq \mathcal{V}_{\pi^{(t)}}(\mathbf{x}),\ \forall \mathbf{x}\ .$$

Furthermore, either $\pi^{(t)}$ is the optimal policy π^*, or there exists at least one state $\mathbf{x}'$ such that:

$$\mathcal{V}_{\pi^{(t+1)}}(\mathbf{x}') > \mathcal{V}_{\pi^{(t)}}(\mathbf{x}').$$

Proof: *see, for example, the book by Puterman [1994].* ▮

A corollary of this theorem is the convergence of policy iteration:

Corollary 2.2.6 *Policy iteration converges to the optimal policy π^*.* ▮

Note that the algorithm in Figure 2.3 may terminate with a suboptimal policy if the maximum number of iterations is reached or the Bellman error tolerance is set to a value greater than zero.

It is interesting to note that steps of the simplex algorithm when applied to solving the dual linear programming formulation in Section 2.2.1 correspond to policy changes at single states. On the other hand, steps of policy iteration can involve policy changes at multiple states. Thus, in practice, policy iteration tends to be faster than the linear programming approach [Puterman, 1994].

Policy iteration converges in at most as many iterations as value iteration [Puterman, 1994]. In practice, policy iteration tends to find the optimal policy in many fewer iterations, though each iteration is more costly computationally. Obtaining a tight bound on the number of iterations required for policy iteration to converge is still an open problem. However, in practice, the convergence to the optimal policy is usually very quick.

2.3 Approximate solution algorithms

In the previous section, we presented three algorithms for find optimal solutions to MDPs. The linear programming approach, for example, is guaranteed to yield a solution in time polynomial in the number of states and actions. Unfortunately, the number of states in most practical applications is too large for these methods to be feasible. In the *SysAdmin* problem, for example, the state $\mathbf{x}$ of the system is an assignment describing which machines are working or have failed; that is, a state $\mathbf{x}$ is an assignment to each random variable X_i.

Thus, the number of states is exponential in the number m of machines in the network ($|\mathbf{X}| = N = 2^m$). Hence, even representing an explicit value function in problems with more than about ten machines is infeasible.

In this section, we discuss the use of an *approximate* value function, which admits a compact representation. We also describe approximate versions of these exact algorithms that use approximate value functions. Our description in this section is somewhat abstract, and does not specify how the basic operations required by the algorithms can be performed explicitly. In later chapters, we elaborate on these issues, and describe the algorithms in detail.

2.3.1 Linear Value Functions

A very popular choice for approximating value functions is by using *linear regression*, as first proposed by Bellman *et al.* [1963]. Here, we define our space of allowable value functions $\mathcal{V} \in \mathcal{H} \subseteq \mathbb{R}^N$ via a set of *basis functions*:

Definition 2.3.1 (linear value function) *A* linear value function *over a set of basis functions $H = \{h_1, \ldots, h_k\}$ is a function $\mathcal{V}$ that can be written as $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{k} w_j \; h_j(\mathbf{x})$ for some coefficients $\mathbf{w} = (w_1, \ldots, w_k)'$.* ∎

We can now define $\mathcal{H}$ to be the linear subspace of $\mathbb{R}^N$ spanned by the basis functions H. It is useful to define an $N \times k$ matrix $\mathbf{H}$ whose columns are the k basis functions viewed as vectors. Specifically, the jth column of $\mathbf{H}$ corresponds to h_j, while the ith row of this column corresponds to the assignment to h_j in the ith state, $h_j(\mathbf{x}_i)$. In a more compact notation, our approximate value function is then represented by $\mathbf{H}\mathbf{w}$.

The expressive power of this linear representation is equivalent, for example, to that of a single layer neural network with features corresponding to the basis functions defining $\mathcal{H}$. Once the features are defined, we must optimize the coefficients $\mathbf{w}$ in order to obtain a good approximation for the true value function. We can view this approach as separating the problem of defining a reasonable space of features and the induced space $\mathcal{H}$, from the problem of searching within the space. The former problem is typically the purview of domain experts, while the latter is the focus of analysis and algorithmic design. Clearly, feature selection is an important issue for essentially all areas of learning and approximation. We offer some simple methods for selecting good features for MDPs in Section 14.2.1, but it is not our goal to address this large and important topic in this thesis.

Once we have a chosen a linear value function representation and a set of basis functions, the problem becomes one of finding values for the weights $\mathbf{w}$ such that $\mathbf{H}\mathbf{w}$ will yield a good approximation of the true value function. In this section, we consider two such approaches: approximate dynamic programming using policy iteration, and linear programming-based approximation.[3] In remainder of this thesis, we show how we can exploit problem structure to transform these approaches into practical algorithms that can deal with exponentially-large state spaces.

2.3.2 Linear programming-based approximation

The simplest approximation algorithm is based on the LP-based solution in Section 2.2.1. The approximate formulation for the LP approach, first proposed by Schweitzer and Seidmann [1985], restricts the space of allowable value functions to the linear space spanned by our basis functions. In this approximate formulation, the variables are $w_1, \ldots, w_k$: the weights for our basis functions. The LP is given by:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k \ ; \\ \text{Minimize:} & \sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \ h_i(\mathbf{x}) \ ; \\ \text{Subject to:} & \sum_i w_i \ h_i(\mathbf{x}) \geq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i \ h_i(\mathbf{x}') \quad \forall \mathbf{x} \in \mathbf{X}, \forall a \in A. \end{array} \tag{2.8}$$

In other words, this formulation takes the LP in (2.4) and substitutes the explicit state value function by a linear value function representation $\sum_i w_i \ h_i(\mathbf{x})$, or, in our more compact notation, $\mathcal{V}$ is replaced by $\mathbf{H}\mathbf{w}$. This linear program is guaranteed to be feasible if a constant function — a function with the same constant value for all states — is included in the set of basis functions. To simplify our presentation, we assume that this basis function is included:

Assumption 2.3.2 (constant basis function) *The constant function is included in our set of basis function. We will denote this basis function by h_0:*

[3] Our techniques easily extend to approximate versions of value iteration.

$$h_0(\mathbf{x}) = 1 \ , \ \forall \mathbf{x} \ . \ \blacksquare$$

In this linear programming-based approximation, the choice of state relevance weights, α, becomes important. Intuitively, not all constraints in this LP are binding; that is, the constraints are tighter for some states than for others. For each state $\mathbf{x}$, the relevance weight $\alpha(\mathbf{x})$ indicates the relative importance of a tight constraint. Therefore, unlike the exact case, the solution obtained may differ for different choices of the positive weight vector α; de Farias and Van Roy [2001a] provide an example of this effect.

The recent work of de Farias and Van Roy [2001a] provides some analysis of the quality of the approximation obtained by this approach relative to that of the best possible approximation in the subspace, and some guidance as to selecting α so as to improve the quality of the approximation. In particular, their analysis shows that this LP provides the best approximation (in a weighted $\mathcal{L}_1$-norm sense) $\mathbf{H}\mathbf{w}^*$ of the optimal value function $\mathcal{V}^*$ subject to the constraint that $\mathbf{H}\mathbf{w}^* \geq \mathcal{T}^*\mathbf{H}\mathbf{w}^*$, where the weights in the $\mathcal{L}_1$ norm are the state relevance weights α. Additionally, de Farias and Van Roy provide an analysis of the quality of the greedy policy generated from the approximation $\mathbf{H}\mathbf{w}$ obtained from this LP-based approach.

The transformation from an exact to an approximate problem formulation has the effect of reducing the number of free variables in the LP to k (one for each basis function coefficient), but the number of constraints remains $N \times |A|$. In our *SysAdmin* problem, for example, the number of constraints in the LP in (2.8) is $(m+1) \cdot 2^m$, where m is the number of machines in the network. Thus, the process of generating the constraints and solving the LP still seems unmanageable for more than a few machines. de Farias and Van Roy [2001b] analyze the error introduced by an algorithm, where the LP is solved with a sampled subset of the $N \times |A|$. To obtain these theoretical guarantees, the constraints must be sampled according to a particular, often unattainable, distribution. In Chapter 5, we discuss how we can exploit structure in an MDP to provide for a compact closed-form representation and an efficient solution to this LP.

2.3.3 Approximate policy iteration

Projections

The steps in the policy iteration algorithm require a manipulation of both value functions and policies, both of which often cannot be represented explicitly in large MDPs. To define a version of the policy iteration algorithm that uses approximate value functions, we use the following basic idea: We restrict the algorithm to using only value functions within the provided linear subspace $\mathcal{H}$; whenever the algorithm takes a step that results in a value function $\mathcal{V}$ that is outside this space, we *project* the result back into the space by finding the value function within the space which is closest to $\mathcal{V}$. More precisely:

Definition 2.3.3 (projection operator) *A* projection operator Π *is a mapping* $\Pi : \mathbb{R}^N \rightarrow \mathcal{H}$. Π *is said to be a* projection w.r.t. a norm $\|\cdot\|$ *if* $\Pi\mathcal{V} = \mathbf{H}\mathbf{w}^*$ *such that* $\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|\mathbf{H}\mathbf{w} - \mathcal{V}\|$. ▮

That is, $\Pi\mathcal{V}$ is the linear combination of the basis functions that is closest to $\mathcal{V}$ with respect to the chosen norm.

Our approximate policy iteration algorithm performs the policy improvement step exactly. In the value determination step, the value function — the value of acting according to the current policy $\pi^{(t)}$ — is approximated through a linear combination of basis functions.

We now consider the problem of value determination for a policy $\pi^{(t)}$ in detail. We can rewrite the value determination step in terms of matrices and vectors. If we view $\mathcal{V}_{\pi^{(t)}}$ and $R_{\pi^{(t)}}$ as N-vectors, and $P_{\pi^{(t)}}$ as an $N \times N$ matrix, we have the equations:

$$\mathcal{V}_{\pi^{(t)}} = R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathcal{V}_{\pi^{(t)}}.$$

This is a system of linear equations with one equation for each state, which can only be solved exactly for relatively small N. Our goal is to provide an approximate solution, within $\mathcal{H}$. More precisely, we want to find:

$$\begin{aligned} \mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} \left\|\mathbf{H}\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathbf{H}\mathbf{w})\right\| ; \\ &= \arg\min_{\mathbf{w}} \left\|\left(\mathbf{H} - \gamma P_{\pi^{(t)}} \mathbf{H}\right) \mathbf{w}^{(t)} - R_{\pi^{(t)}}\right\| . \end{aligned}$$

Thus, our *approximate policy iteration* algorithm alternates between two steps:

$$\begin{aligned} \mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} \left\| \mathbf{H}\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathbf{H}\mathbf{w}) \right\|; & (2.9) \\ \pi^{(t+1)} &= \textsf{Greedy}[\mathbf{H}\mathbf{w}^{(t)}]. & (2.10) \end{aligned}$$

Max-norm projection

An approach along these lines has been used in various papers, with several recent theoretical and algorithmic results [Schweitzer & Seidmann, 1985; Tsitsiklis & Van Roy, 1996a; Van Roy, 1998; Koller & Parr, 1999; Koller & Parr, 2000]. However, these approaches suffer from a problem that we might call "norm incompatibility." When computing the projection, they utilize the standard Euclidean projection operator with respect to the $\mathcal{L}_2$ norm or a *weighted* $\mathcal{L}_2$ norm. [4] On the other hand, most of the convergence and error analyses for MDP algorithms utilize max-norm ($\mathcal{L}_\infty$). This incompatibility has made it difficult to provide error guarantees.

We can tie the projection operator more closely to the error bounds through the use of a projection operator in $\mathcal{L}_\infty$ norm. The problem of minimizing the $\mathcal{L}_\infty$ norm has been studied in the optimization literature as the problem of finding the Chebyshev solution[5] to an overdetermined linear system of equations [Cheney, 1982]. The problem is defined as finding $\mathbf{w}^*$ such that:

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \left\| C\mathbf{w} - \mathbf{b} \right\|_\infty . \qquad (2.11)$$

We use an algorithm due to Stiefel [1960], that solves this problem by linear programming:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k, \phi \,; \\ \text{Minimize:} & \phi \,; \\ \text{Subject to:} & \phi \geq \sum_{j=1}^{k} c_{ij} w_j - b_i \,, \quad \text{and} \\ & \phi \geq b_i - \sum_{j=1}^{k} c_{ij} w_j \,, \quad i = 1 \ldots N. \end{array} \qquad (2.12)$$

[4]Weighted $\mathcal{L}_2$ norm projections are stable and have meaningful error bounds when the weights correspond to the stationary distribution of a fixed policy under evaluation (value determination) [Van Roy, 1998], but they are not stable when combined with $\mathcal{T}^*$. Averagers [Gordon, 1995] are stable and non-expansive in $\mathcal{L}_\infty$, but require that the mixture weights be determined *a priori*. Thus, they do not, in general, minimize $\mathcal{L}_\infty$ error.

[5]The Chebyshev norm is also referred to as max, supremum and $\mathcal{L}_\infty$ norms and the minimax solution.

The constraints in this linear program imply that $\phi \geq \left|\sum_{j=1}^{k} c_{ij} w_j - b_i\right|$ for each i, or equivalently, that $\phi \geq \|C\mathbf{w} - \mathbf{b}\|_\infty$. The objective of the LP is to minimize ϕ. Thus, at the solution $(\mathbf{w}^*, \phi^*)$ of this linear program, $\mathbf{w}^*$ is the solution of Equation (2.11) and ϕ is the $\mathcal{L}_\infty$ projection error.

We can use the $\mathcal{L}_\infty$ projection in the context of the approximate policy iteration in the obvious way. When implementing the projection operation of Equation (2.9), we can use the $\mathcal{L}_\infty$ projection (as in Equation (2.11)), where $C = (\mathbf{H} - \gamma P_{\pi^{(t)}} \mathbf{H})$ and $\mathbf{b} = R_{\pi^{(t)}}$. This minimization can be solved using the linear program of (2.12).

A key point is that this LP only has $k+1$ variables. However, there are $2N$ constraints, which makes it impractical for large state spaces. In the *SysAdmin* problem, for example, the number of constraints in this LP is exponential in the number of machines in the network (a total of $2 \cdot 2^m$ constraints for m machines). In future chapters, we show that, in *factored* MDPs with linear value functions, all the $2N$ constraints can be represented efficiently, leading to a tractable algorithm.

Error analysis

We motivated our use of the max-norm projection within the approximate policy iteration algorithm via its compatibility with standard error analysis techniques for MDP algorithms. We now provide a careful analysis of the impact of the $\mathcal{L}_\infty$ error introduced by the projection step. The analysis provides motivation for the use of a projection step that directly minimizes this quantity. We acknowledge, however, that the main impact of this analysis is motivational. In practice, we cannot provide *a priori* guarantees that an $\mathcal{L}_\infty$ projection will outperform other methods.

Our goal is to analyze approximate policy iteration in terms of the amount of error introduced at each step by the projection operation. If the error is zero, then we are performing exact value determination, and no error should accrue. If the error is small, we should get an approximation that is accurate. This result follows from the analysis below. More precisely, we define the *max-norm projection error* as the error resulting from the approximate value determination step:

$$\beta^{(t)} = \left\|\mathbf{H}\mathbf{w}^{(t)} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathbf{H}\mathbf{w}^{(t)}\right)\right\|_\infty .$$

Note that, by using our max-norm projection, we are finding the set of weights $\mathbf{w}^{(t)}$ that exactly minimizes the one-step projection error $\beta^{(t)}$. That is, we are choosing the best possible weights with respect to this error measure. Furthermore, this is exactly the error measure that is going to appear in the bounds of our theorem. Thus, we can now make the bounds for each step as tight as possible.

We first show that the projection error accrued in each step is bounded:

Lemma 2.3.4 *The value determination error is bounded: There exists a constant $\beta_P \leq R_{max}$ such that $\beta_P \geq \beta^{(t)}$ for all iterations t of the algorithm.*
Proof: *See Appendix A.1.1.* ▮

Due to the contraction property of the Bellman operator, the overall accumulated error is a decaying average of the projection error incurred throughout all iterations:

Definition 2.3.5 (discounted value determination error) *The* discounted value determination error *at iteration t is defined as: $\overline{\beta}^{(t)} = \beta^{(t)} + \gamma\overline{\beta}^{(t-1)}$; $\overline{\beta}^{(0)} = 0$.* ▮

Lemma 2.3.4 implies that the accumulated error remains bounded in approximate policy iteration: $\overline{\beta}^{(t)} \leq \frac{\beta_P(1-\gamma^t)}{1-\gamma}$. We can now bound the loss incurred when acting according to the policy generated by our approximate policy iteration algorithm, as opposed to the optimal policy:

Theorem 2.3.6 *In the approximate policy iteration algorithm, let $\pi^{(t)}$ be the policy generated at iteration t. Furthermore, let $\mathcal{V}_{\pi^{(t)}}$ be the* actual *value of acting according to this policy. The loss incurred by using policy $\pi^{(t)}$ as opposed to the optimal policy π^* with value $\mathcal{V}^*$ is bounded by:*

$$\|\mathcal{V}^* - \mathcal{V}_{\pi^{(t)}}\|_\infty \leq \gamma^t \|\mathcal{V}^* - \mathcal{V}_{\pi^{(0)}}\|_\infty + \frac{2\gamma\overline{\beta}^{(t)}}{(1-\gamma)^2}. \qquad (2.13)$$

Proof: *See Appendix A.1.2.* ▮

In words, Equation (2.13) shows that the difference between our approximation at iteration t and the optimal value function is bounded by the sum of two terms. The first term is present in standard policy iteration and goes to zero exponentially fast. The second is the discounted accumulated projection error and, as Lemma 2.3.4 shows, is bounded. This second term can be minimized by choosing $\mathbf{w}^{(t)}$ as the one that minimizes:

$$\left\|\mathbf{H}\mathbf{w}^{(t)} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathbf{H}\mathbf{w}^{(t)}\right)\right\|_\infty ,$$

which is exactly the computation performed by the max-norm projection. Therefore, this theorem motivates the use of max-norm projections to minimize the error term that appears in our bound.

The bounds we have provided so far may seem fairly trivial, as we have not provided a strong *a priori* bound on $\beta^{(t)}$. Fortunately, several factors make these bounds interesting despite the lack of *a priori* guarantees. If approximate policy iteration converges, as occurred in all of our experiments, we can obtain a much tighter bound: If $\widehat{\pi}$ is the policy after convergence, then

$$\|\mathcal{V}^* - \mathcal{V}_{\widehat{\pi}}\|_\infty \leq \frac{2\gamma\beta_{\widehat{\pi}}}{(1-\gamma)},$$

where $\beta_{\widehat{\pi}}$ is the one-step max-norm projection error associated with estimating the value of $\widehat{\pi}$. Since the max-norm projection operation provides $\beta_{\widehat{\pi}}$, we can easily obtain an *a posteriori* bound as part of the policy iteration procedure. More details are provided in Section 5.3.

If approximate policy iteration gets stuck in a cycle, one could rewrite the bound in Theorem 2.3.6 in terms of the worst case projection error β_P, or the worst projection error in a cycle of policies. These formulations would be closer to the analysis of Bertsekas and Tsitsiklis, [1996, Proposition 6.2, p.276]. However, consider the case where most policies (or most policies in the final cycle) have a low projection error, but there are a few policies that cannot be approximated well using the projection operation, so that they have a large one-step projection error. A worst-case bound would be very loose, because it would be dictated by the error of the most difficult policy to approximate. On the other hand, using our discounted accumulated error formulation, errors introduced by policies that are hard to approximate decay very rapidly. Thus, the error bound represents an “average” case analysis: a decaying average of the projection errors for policies encountered at the successive iterations of the algorithm. As in the convergent case, this bound can be computed easily as part of the policy iteration procedure when max-norm projection is used.

The practical benefit of *a posteriori* bounds is that they can give meaningful feedback on the impact of the choice of the value function approximation architecture. While we are not explicitly addressing the difficult and general problem of feature selection in this thesis, our error bounds motivate algorithms that aim to minimize the error *given* an approximation architecture and provide feedback that could be useful in future efforts to automatically discover or improve approximation architectures.

2.4 Discussion and related work

This chapter presents Markov decision processes, the basic mathematical framework for representing planning problems in the presence of uncertainty. The field of MDPs, as it is popularly known, was formalized by Bellman [1957] in the 1950's. The importance of value function approximation was recognized at an early stage by Bellman himself [1963]. In the early 1990's, the MDP framework was recognized by AI researchers as a formal framework that could be used to address the problem of planning under uncertainty [Dean *et al.*, 1993].

Within the AI community, value function approximation developed concomitantly with the notion of value function representations for Markov chains. Sutton's seminal paper on temporal difference learning [1988], which addressed the use of value functions for prediction but not planning, assumed a very general representation of the value function and noted the connection to general function approximators such as neural networks. However, the stability of this combination was not directly addressed at that time.

Several important developments gave the AI community deeper insight into the relationship between function approximation and dynamic programming. Tsitsiklis and Van Roy [1996b] and, independently, Gordon [1995] popularized the analysis of approximate MDP methods via the contraction properties of the dynamic programming operator and function approximator. Tsitsiklis and Van Roy [1996a] later established a general convergence result for linear value function approximators and $TD(\lambda)$. Bertsekas and Tsitsiklis [1996] unified a large body of work on approximate dynamic programming under the name of *Neuro-dynamic Programming*, also providing many novel and general error analyses. The analysis of the novel max-norm projection version of approximate policy iteration, which we present in this chapter, builds on some of these techniques. The max-norm projection property of our algorithm directly minimizes a bound on the quality of the resulting policy obtained from this analysis.

Approximate linear programming for MDPs using linear value function approximation was introduced by Schweitzer and Seidmann [1985], though the approach was somewhat underappreciated until fairly recently due to the lack of compelling error analyses and the lack of an effective method for handling the large number of constraints. Recent work by de Farias and Van Roy [2001a] has started to address some of these concerns with new error bounds on the quality of the greedy policy with respect to the approximate value function generated by the linear programming approach.

Chapter 3

Factored Markov decision processes

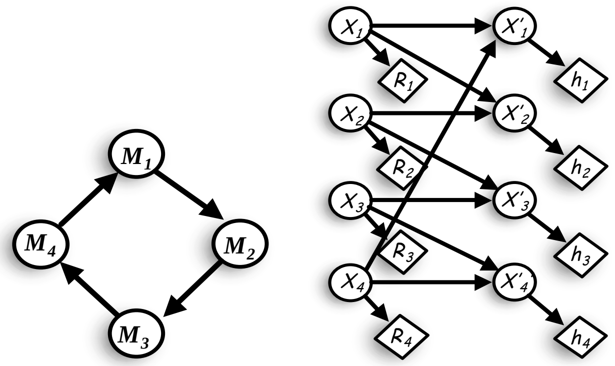
Factored MDPs are a representation language that allows us to exploit problem structure to represent exponentially-large MDPs very compactly. In this chapter, we review this representation as it is a central element for our efficient algorithms. We also present a structured representation for an approximate value function, which will allow us to design very efficient approximate solution algorithms for exponentially-large MDPs.

3.1 Representation

In a factored MDP, the set of states is described via a set of *random (state) variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $\mathrm{Dom}(X_i)$. A state $\mathbf{x}$ defines a value $x_i \in \mathrm{Dom}(X_i)$ for each variable X_i. In general, we use upper case letters (*e.g.*, X) to denote random variables, and lower case (*e.g.*, x) to denote their values. We use boldface to denote vectors of variables (*e.g.*, $\mathbf{X}$) or their values ($\mathbf{x}$). For an instantiation $\mathbf{y} \in \mathrm{Dom}(\mathbf{Y})$ and a subset of these variables $\mathbf{Z} \subseteq \mathbf{Y}$, we use $\mathbf{y}[\mathbf{Z}]$ to denote the value of the variables $\mathbf{Z}$ in the instantiation $\mathbf{y}$.

3.1.1 Factored transition model

In a standard MDP as presented in Section 2.1, the representation of the transition model is exponentially large in the number of state variables. However, the global state transition



(a) **(b)**

$P(X'_i = \textit{Working} \mid X_i, X_{i-1}, A)$:

	Action is reboot:	
	machine i	other machine
$X_{i-1} = D \wedge X_i = D$	1	0.05
$X_{i-1} = D \wedge X_i = W$	1	0.5
$X_{i-1} = W \wedge X_i = D$	1	0.09
$X_{i-1} = W \wedge X_i = W$	1	0.9

(c)

Figure 3.1: Factored MDP example: from a network topology (a) we obtain the factored MDP representation (b) with the CPDs described in (c).

model τ can often be represented compactly as the product of local factors by using a *dynamic Bayesian network (DBN)* [Dean & Kanazawa, 1989]. Such a model is thus called a *factored MDP*. The idea of representing a large MDP using a factored model was first proposed by Boutilier *et al.* [1995].

Let X_i denote the variable X_i at the current time and X'_i, the same variable at the next step. The *transition graph* of a DBN is a two-layer directed acyclic graph G_τ whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$. We denote the parents of X'_i in the graph by $\mathsf{Parents}_\tau(X'_i)$. For simplicity of exposition, we assume that $\mathsf{Parents}_\tau(X'_i) \subseteq \mathbf{X}$; thus, all arcs in the DBN are between variables in consecutive time slices. (This assumption is used for expository purposes only; intra-time-slice arcs are handled by a small modification presented in Section 3.3.) Each node X'_i is associated with a *conditional probability distribution (CPD)* $P_\tau(X'_i \mid \mathsf{Parents}_\tau(X'_i))$. The transition probability $P_\tau(\mathbf{x}' \mid \mathbf{x})$ is then defined to be:

$$P_\tau(\mathbf{x}' \mid \mathbf{x}) = \prod_i P_\tau(x'_i \mid \mathbf{x}[\mathsf{Parents}_\tau(X'_i)]) \ ,$$

where $\mathbf{x}[\mathsf{Parents}_\tau(X'_i)]$ is the value in $\mathbf{x}$ to the variables in $\mathsf{Parents}_\tau(X'_i)$. The complexity of this representation is now linear in the number of state variables (the number of factors in our DBN), and, in the worst case, only exponential in the number of variables in the largest factor. In Chapter 7, we present a representation that can further reduce this complexity.

Example 3.1.1 *Consider, for example, an instance of the SysAdmin problem with four computers, $M_1, \ldots, M_4$ in an unidirectional ring topology as shown in Figure 3.1(a). Our first task in modelling this problem as a factored MDP is to define the state space* $\mathbf{X}$*. Each machine is associated with a binary random variable X_i, representing whether it is working or has failed. Thus, our state space is represented by four random variables: $\{X_1, X_2, X_3, X_4\}$, where the domain of each state variable is given by $\mathrm{Dom}[X_i] = \{Working, Dead\}$. The next task is to define the transition model, represented as a DBN. The parents of the next time step variables X_i' depend on the network topology. Specifically, the probability that machine i will fail at the next time step depends on whether it is working at the current time step and on the status of its direct neighbors (parents in the topology) in the network at the current time step. As shown in Figure 3.1(b), the parents of X_i' in this example are X_i and X_{i-1}. The CPD of X_i' is such that if $X_i =$ Dead, then $X_i' =$ Dead with high probability; that is, failures tend to persist. If $X_i =$ Working, then the distribution over possible values of X_i' is a function of the number of parents that are dead (in the unidirectional ring topology X_i' has only one other parent X_{i-1}); that is, a failure in any of its neighbors can increase the chance that machine i will fail.* ∎

We have described how to represent factored the Markovian transition dynamics arising from an MDP as a DBN, but we have not directly addressed the representation of actions. Generally, we can define the transition dynamics of an MDP by defining a separate DBN model $\tau_a = \langle G_a, P_a \rangle$ for each action a. In Chapter 8, we introduce an additional factorization of the action variables.

Example 3.1.2 *In our system administrator example, we have an action a_i for rebooting each one of the machines, and a default action d for doing nothing. The transition model described above corresponds to the "do nothing" action. The transition model for a_i is different from d only in the transition model for the variable X_i', which is now $X_i' =$ Working with probability one, regardless of the status of the neighboring machines. The table in Figure 3.1(c) shows the actual CPD for $P(X_i' =$ Working $\mid X_i, X_{i-1}, A)$, with one entry for each assignment to the state variables X_i and X_{i-1}, and to the action A.* ∎

3.1.2 Factored reward function

To fully specify an MDP, we also need to provide a compact representation of the reward function. We assume that the reward function is factored additively into a set of localized reward functions, each of which only depends on a small set of variables. In our example, we might have a reward function associated with each machine i, which depends on X_i. That is, the SysAdmin is paid on a per-machine basis: at every time step, she receives money for machine i only if it is working. We can formalize this concept of localized functions:

Definition 3.1.3 (scope) *A function f has a* scope *Scope*$[f] = \mathbf{C} \subseteq \mathbf{X}$ *if* $f : \mathrm{Dom}(\mathbf{C}) \mapsto \mathbb{R}$. ▮

If f has scope $\mathbf{Y}$ and $\mathbf{Y} \subseteq \mathbf{Z}$, we use $f(\mathbf{z})$ as shorthand for $f(\mathbf{z}[\mathbf{Y}])$, where $\mathbf{y}$ is the part of the instantiation $\mathbf{z}$ that corresponds to variables in $\mathbf{Y}$.

We can now characterize the concept of local rewards. Let $R_1^a, \ldots, R_r^a$ be a set of functions, where the scope of each R_i^a is restricted to variable cluster $\mathbf{W}_i^a \subset \{X_1, \ldots, X_n\}$. The reward for taking action a at state $\mathbf{x}$ is defined to be $R^a(\mathbf{x}) = \sum_{i=1}^r R_i^a(\mathbf{W}_i^a) \in \mathbb{R}$. In our example, we have a reward function R_i associated with each machine i, which depends only X_i, and does not depend on the action choice. These local rewards are represented by the diamonds in Figure 3.1(b), in the usual notation for influence diagrams [Howard & Matheson, 1984]. Although not every problem can be modelled compactly using such a factored representation of the reward function, we believe that such a representation is applicable in many large-scale problems, as discussed in Chapter 1.

3.2 Factored value functions

One might be tempted to believe that factored transition dynamics and rewards would result in a factored value function, which can thereby be represented compactly. Unfortunately, even in trivial factored MDPs, there is no guarantee that structure in the model is preserved in the value function [Koller & Parr, 1999], and exact solutions to these problems are intractable [Mundhenk *et al.*, 2000; Liberatore, 2002]. Thus, in general, we must resort to approximate solutions to these factored MDPs.

The linear value function approach, and the algorithms described in Section 2.3, apply to any choice of basis functions. In the context of factored MDPs, Koller and Parr [1999] suggest a specific type of basis function, which is particularly compatible with the structure of a factored MDP. They suggest that although the value function is typically not structured, there are many cases where it might be "close" to structured. That is, it might be well-approximated using a linear combination of functions each of which refers only to a small number of variables. More precisely, we define:

Definition 3.2.1 (factored value function) *A* factored (linear) value function *is a linear function over the basis $h_1, \dots, h_k$, where the scope of each h_i is restricted to some subset of variables $\mathbf{C}_i$.* ▮

Value functions of this type have a long history in the area of multi-attribute utility theory [Keeney & Raiffa, 1976]. In our example, we might have a basis function h_i for each machine, indicating whether it is working or not. Each basis function has scope restricted to X_i. These are represented as diamonds in the next time step in Figure 3.1(b).

Factored value functions provide the key to performing efficient computations over the exponential-sized state spaces we have in factored MDPs. The main insight is that restricted-scope functions (including our basis functions) allow for certain basic operations to be implemented very efficiently. In the remainder of this chapter, we show how structure in factored MDPs can be exploited to perform one such crucial operation very efficiently: one-step lookahead (backprojection). Then, in Chapter 4 we present a novel LP decomposition technique, which exploits problem structure to represent exponentially many LP constraints very compactly. These basic building blocks will allow us to formulate very efficient approximation algorithms for factored MDPs. For example, in Chapter 5, we present two such algorithms, each in its own self-contained section: the linear programming-based approximation algorithm for factored MDPs in Section 5.1, and approximate policy iteration with max-norm projection in Section 5.2.

3.3 One-step lookahead

A key step in all of our planning algorithms is the computation of the one-step lookahead value of some action a. This is necessary, for example, when computing the greedy policy, as in Equation (2.1). Let us consider the computation of a Q function, which is again given by:

$$Q_a(\mathbf{x}) = R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a)\mathcal{V}(\mathbf{x}'). \tag{3.1}$$

That is, $Q_a(\mathbf{x})$ is given by the current reward plus the discounted expected future value. If we compute the Q-function, we obtain the greedy policy simply by $\mathsf{Greedy}[\mathcal{V}](\mathbf{x}) = \max_a Q_a(\mathbf{x})$.

Recall that we are estimating the long-term value of our policy using a set of basis functions: $\mathcal{V}(\mathbf{x}) = \sum_i w_i\, h_i(\mathbf{x})$. Thus, we can rewrite Equation (3.1) as:

$$Q_a(\mathbf{x}) = R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i\, h_i(\mathbf{x}'). \tag{3.2}$$

The size of the state space is exponential, so that computing the expectation $\sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i\, h_i(\mathbf{x}')$ seems infeasible. Fortunately, as discussed by Koller and Parr [1999], this expectation operation, or backprojection, can be performed efficiently if the transition model and the value function are both factored appropriately. The linearity of the value function permits a linear decomposition, where each summand in the expectation can be viewed as an independent value function and updated in a manner similar to the value iteration procedure used by Boutilier *et al.* [2000]. We now recap the construction briefly, by first defining:

$$G^a(\mathbf{x}) = \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i\, h_i(\mathbf{x}') = \sum_i w_i \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) h_i(\mathbf{x}').$$

Thus, we can compute the expectation of each basis function separately:

$$g_i^a(\mathbf{x}) = \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) h_i(\mathbf{x}'),$$

```
$Backproj_a(h)$ — WHERE BASIS FUNCTION $h$ HAS SCOPE $\mathbf{C}$.
    DEFINE THE SCOPE OF THE BACKPROJECTION: $\Gamma_a(\mathbf{C}') = \cup_{X'_i \in \mathbf{C}'} \mathrm{PARENTS}_a(X'_i)$.
    FOR EACH ASSIGNMENT $\mathbf{y} \in \Gamma_a(\mathbf{C}')$:
        $g^a(\mathbf{y}) = \sum_{\mathbf{c}' \in \mathbf{C}'} \prod_{i | X'_i \in \mathbf{C}'} P_a(\mathbf{c}'[X'_i] \mid \mathbf{y}) h(\mathbf{c}')$.

    RETURN $g^a$.
```

Figure 3.2: Backprojection of basis function h.

and then weight them by w_i to obtain the total expectation $G^a(\mathbf{x}) = \sum_i w_i \; g_i^a(\mathbf{x})$. The intermediate function g_i^a is called the *backprojection* of the basis function h_i through the transition model P_a, which we denote by $g_i^a = P_a h_i$. Note that, in factored MDPs, the transition model P_a is factored (represented as a DBN) and the basis functions h_i have scope restricted to a small set of variables. These two important properties allow us to compute the backprojections very efficiently.

We now show how some restricted-scope function h (such as our basis functions) can be backprojected through some transition model P_τ represented as a DBN τ. Here h has scope restricted to $\mathbf{Y}$; our goal is to compute $g = P_\tau h$. We define the *backprojected scope of* $\mathbf{Y}$ *through* τ as the set of parents of $\mathbf{Y}'$ in the transition graph G_τ; $\Gamma_\tau(\mathbf{Y}') = \cup_{Y'_i \in \mathbf{Y}'} \mathrm{Parents}_\tau(Y'_i)$. If intra-time-slice arcs are included, so that

$$\mathrm{Parents}_\tau(X'_i) \in \{X_1, \ldots, X_n, X'_1, \ldots, X'_n\},$$

then the only change to our algorithm is in the definition of backprojected scope of $\mathbf{Y}$ through τ. The definition now includes not only direct parents of Y', but also all variables in $\{X_1, \ldots, X_n\}$ that are ancestors of Y':

$$\Gamma_\tau(\mathbf{Y}') = \{X_j \mid \text{there exist a directed path from } X_j \text{ to any } X'_i \in \mathbf{Y}'\}.$$

Thus, the backprojected scope may become larger, but the functions are still factored.

We can now show that, if h has scope restricted to $\mathbf{Y}$, then its backprojection g has scope restricted to the parents of $\mathbf{Y}'$, *i.e.*, $\Gamma_\tau(\mathbf{Y}')$. Furthermore, each backprojection can be computed by only enumerating settings of variables in $\Gamma_\tau(\mathbf{Y}')$, rather than settings of all variables $\mathbf{X}$:

$$\begin{aligned}
g(\mathbf{x}) &= (P_\tau h)(\mathbf{x}); \\
&= \sum_{\mathbf{x}'} P_\tau(\mathbf{x}' \mid \mathbf{x}) h(\mathbf{x}'); \\
&= \sum_{\mathbf{x}'} P_\tau(\mathbf{x}' \mid \mathbf{x}) h(\mathbf{y}'); \\
&= \sum_{\mathbf{y}'} P_\tau(\mathbf{y}' \mid \mathbf{x}) h(\mathbf{y}') \sum_{\mathbf{u}' \in (\mathbf{x}' - \mathbf{y}')} P_\tau(\mathbf{u}' \mid \mathbf{x}); \\
&= \sum_{\mathbf{y}'} P_\tau(\mathbf{y}' \mid \mathbf{z}) h(\mathbf{y}'); \\
&= g(\mathbf{z});
\end{aligned}$$

where $\mathbf{z}$ is the value of $\Gamma_\tau(\mathbf{Y}')$ in $\mathbf{x}$ and the term $\sum_{\mathbf{u}' \in (\mathbf{x}' - \mathbf{y}')} P_\tau(\mathbf{u}' \mid \mathbf{x}) = 1$ as it is the sum of a probability distribution over a complete domain. Therefore, we see that $(P_\tau h)$ is a function whose scope is restricted to $\Gamma_\tau(\mathbf{Y}')$. Note that the cost of the computation depends linearly on $|\mathrm{Dom}(\Gamma_\tau(\mathbf{Y}'))|$, which depends on $\mathbf{Y}$ (the scope of h) and on the complexity of the process dynamics. This backprojection procedure is summarized in Figure 3.2.

Returning to our example, consider a basis function h_i that is an indicator of variable X_i: it takes value 1 if the i^{th} machine is working and 0 otherwise. Each h_i has scope restricted to X_i', thus, its backprojection g_i has scope restricted to $\text{Parents}_\tau(X_i')$: $\Gamma_\tau(X_i') = \{X_{i-1}, X_i\}$.

3.4 Discussion and related work

This chapter describes the framework of factored MDPs, which allows the representation of exponentially-large planning problems very compactly. This model builds on a dynamic Bayesian network (DBN) [Dean & Kanazawa, 1989], which gives a compact representation for a complex transition model. The idea of applying a DBN to represent a large MDP was first proposed by Boutilier *et al.* [1995].

Although factored MDPs give us a very compact representation for large planning problems, computing exact solutions to these problems is known to be hard [Mundhenk *et al.*, 2000; Liberatore, 2002]. Furthermore, as shown by Allender *et al.* [2002], a compact approximate solution with theoretical guarantees generally does not exist.

However, as suggested by Koller and Parr [1999], in many practical cases, the value function may be close to structured, and can be well-approximated by a factored linear value function. This chapter describes this factored approximate representation of the value function. We also review an efficient method for performing one-step lookahead planning using a factored value function and a factored MDP, in a manner similar to the value iteration procedure used by Boutilier *et al.* [2000].

Chapter 4

Representing exponentially many constraints

Recall that both of the approximate solution algorithms presented in Chapter 2 use linear programs to obtain the value function coefficients. The number of constraints in both of these LPs is proportional to the number of states in the MDP, this number is exponential in the number of state variables in the factored MDP. In this chapter, we present a novel LP decomposition technique, which exploits problem structure, such as the one present in factored MDPs, to represent exponentially many LP constraints very compactly. This decomposition technique will be a central element in all of our factored planning algorithms.

4.1 Exponentially-large constraint sets

As seen in Section 2.3, both our approximation algorithms require the solution of linear programs: the LP in (2.8) for the linear programming-based approximation algorithm, and the LP in (2.12) for approximate policy iteration. These LPs have some common characteristics: they have a small number of free variables (for k basis functions there are $k + 1$ free variables in approximate policy iteration and k in linear programming-based approximation), but the number of constraints is still exponential in the number of state variables. However, in factored MDPs, these LP constraints have another very useful property: the functionals in the constraints have restricted scope. This key observation allows us to represent these constraints very compactly.

First, observe that the constraints in the linear programs are all of the form:

$$\phi \geq \sum_i w_i \, c_i(\mathbf{x}) - b(\mathbf{x}), \forall \mathbf{x}, \tag{4.1}$$

where only ϕ and $w_1, \ldots, w_k$ are free variables in the LP and $\mathbf{x}$ ranges over all states. This general form represents both the type of constraint in the max-norm projection LP in (2.12) and the linear programming-based approximation formulation in (2.8).[1]

The first insight in our construction is that we can replace the entire set of constraints in Equation (4.1) by one equivalent non-linear constraint:

$$\phi \geq \max_{\mathbf{x}} \sum_i w_i \, c_i(\mathbf{x}) - b(\mathbf{x}). \tag{4.2}$$

The second insight is that this new non-linear constraint can be implemented by a set of linear constraints using a construction that follows the structure of variable elimination in cost networks [Bertele & Brioschi, 1972]. This insight allows us to exploit structure in factored MDPs to represent this constraint compactly.

We tackle the problem of representing the constraint in Equation (4.2) in two steps: first, computing the maximum assignment for a fixed set of weights; then, representing the non-linear constraint by small set of linear constraints, using a construction we call the *factored LP*.

4.2 Maximizing over the state space

First consider a simpler problem: Given some *fixed* weights w_i, we would like to compute the maximization: $\phi^* = \max_{\mathbf{x}} \sum_i w_i \, c_i(\mathbf{x}) - b(\mathbf{x})$, that is, the state $\mathbf{x}$, such that the

[1]The complementary constraints in (2.12), $\phi \geq b(\mathbf{x}) - \sum_i w_i \, c_i(\mathbf{x})$, can be formulated using an analogous construction to the one we present in this section by changing the sign of $c_i(\mathbf{x})$ and $b(\mathbf{x})$. The linear programming-based approximation constraints of (2.8) can also be formulated in this form, as we show in Section 5.1.

difference between $\sum_i w_i \; c_i(\mathbf{x})$ and $b(\mathbf{x})$ is maximal. However, we cannot explicitly enumerate the exponential number of states and compute the difference. Fortunately, structure in factored MDPs allows us to compute this maximum efficiently.

In the case of factored MDPs, our state space is a set of vectors $\mathbf{x}$ which are assignments to the state variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. We can view both $C\mathbf{w}$ and $\mathbf{b}$ as functions of these state variables, and hence also their difference. Thus, we can define a function $F^{\mathbf{w}}(X_1, \ldots, X_n)$ such that $F^{\mathbf{w}}(\mathbf{x}) = \sum_i w_i \; c_i(\mathbf{x}) - b(\mathbf{x})$. Note that we have executed a representation shift; we are viewing $F^{\mathbf{w}}$ as a function of the variables $\mathbf{X}$, which is parameterized by $\mathbf{w}$. Recall that the size of the state space is exponential in the number of variables. Hence, our goal in this section is to compute $\max_{\mathbf{x}} F^{\mathbf{w}}(\mathbf{x})$ without explicitly considering each of the exponentially many states. The solution is to use the fact that $F^{\mathbf{w}}$ has a factored representation. More precisely, $C\mathbf{w}$ has the form $\sum_i w_i \; c_i(\mathbf{Z}_i)$, where $\mathbf{Z}_i$ is a subset of $\mathbf{X}$. For example, we might have $c_1(X_1, X_2)$ which takes value 1 in states where $X_1 =$ *true* and $X_2 =$ *false* and 0 otherwise. Similarly, the vector $\mathbf{b}$ in our case is also a sum of restricted-scope functions. Thus, we can express $F^{\mathbf{w}}$ as a sum $\sum_j f_j^{\mathbf{w}}(\mathbf{Z}_j)$, where $f_j^{\mathbf{w}}$ may or may not depend on $\mathbf{w}$. In the future, we sometimes drop the superscript $\mathbf{w}$ when it is clear from context.

Using our more compact notation, our goal here is simply to compute

$$\max_{\mathbf{x}} \sum_i w_i \; c_i(\mathbf{x}) - b(\mathbf{x}) = \max_{\mathbf{x}} F^{\mathbf{w}}(\mathbf{x}),$$

that is, to find the state $\mathbf{x}$ over which $F^{\mathbf{w}}$ is maximized. Recall that $F^{\mathbf{w}} = \sum_{j=1}^{m} f_j^{\mathbf{w}}(\mathbf{Z}_j)$. We can maximize such a function, $F^{\mathbf{w}}$, without enumerating every state using *non-serial dynamic programming* [Bertele & Brioschi, 1972]. The idea is virtually identical to *variable elimination* in a Bayesian network. We review this construction here, as it is a central component in our solution LP.

Our goal is to compute

$$\max_{x_1,\ldots,x_n} \sum_j f_j(\mathbf{x}[\mathbf{Z}_j]).$$

The main idea is that, rather than summing all functions and then doing the maximization, we maximize over variables one at a time. When maximizing over x_l, only summands involving x_l participate in the maximization.

Example 4.2.1 *Assume*

$$F = f_1(x_1, x_2) + f_2(x_1, x_3) + f_3(x_2, x_4) + f_4(x_3, x_4).$$

We therefore wish to compute:

$$\max_{x_1,x_2,x_3,x_4} f_1(x_1, x_2) + f_2(x_1, x_3) + f_3(x_2, x_4) + f_4(x_3, x_4).$$

We can first compute the maximum over x_4; the functions f_1 and f_2 are irrelevant, so we can push them out. We get

$$\max_{x_1,x_2,x_3} f_1(x_1, x_2) + f_2(x_1, x_3) + \max_{x_4}[f_3(x_2, x_4) + f_4(x_3, x_4)].$$

The result of the internal maximization depends on the values of x_2, x_3; thus, we can introduce a new function $e_1(X_2, X_3)$ whose value at the point x_2, x_3 is the value of the internal $\max$ expression. Our problem now reduces to computing

$$\max_{x_1,x_2,x_3} f_1(x_1, x_2) + f_2(x_1, x_3) + e_1(x_2, x_3),$$

having one fewer variable. Next, we eliminate another variable, say X_3, with the resulting expression reducing to:

$$\max_{x_1,x_2} f_1(x_1, x_2) + e_2(x_1, x_2),$$
$$\textit{where} \quad e_2(x_1, x_2) = \max_{x_3}[f_2(x_1, x_3) + e_1(x_2, x_3)].$$

Finally, we define

$$e_3 = \max_{x_1,x_2} f_1(x_1, x_2) + e_2(x_1, x_2).$$

The result at this point is a number, which is the desired maximum over $x_1, \ldots, x_4$. While the naive approach of enumerating all states requires 63 arithmetic operations if all variables are binary, using variable elimination we only need to perform 23 operations. ∎

The general variable elimination algorithm is described in Figure 4.1. The inputs to the algorithm are the functions to be maximized $\mathcal{F} = \{f_1, \ldots, f_m\}$, an elimination ordering $\mathcal{O}$ on the variables, where $\mathcal{O}(i)$ returns the ith variable to be eliminated, and ElimOperator $(\mathcal{E}, X_l)$ is the operation that will be performed on the set of functions $\mathcal{E}$ when variable X_l is eliminated. If we are maximizing over the state space we use the operator MaxOut defined in Figure 4.2. As in the example above, for each variable X_l to be eliminated, we select the relevant functions $e_1, \ldots, e_L$, those whose scope contains X_l. These functions are removed from the set $\mathcal{F}$ and we introduce a new function $e = \max_{x_l} \sum_{j=1}^{L} e_j$. At this point, the scope of the functions in $\mathcal{F}$ no longer depends on X_l, that is, X_l has been 'eliminated'. This procedure is repeated until all variables have been eliminated. The remaining functions in $\mathcal{F}$ thus have empty scope. The desired maximum is therefore given by the sum of these remaining functions.

The computational cost of this algorithm is linear in the number of new "function values" introduced in the elimination process. More precisely, consider the computation of a new function e whose scope is **Z**. To compute this function, we need to compute $|\mathrm{Dom}[\mathbf{Z}]|$ different values. The cost of the algorithm is linear in the overall number of these values, introduced throughout the execution. As shown by Dechter [1999], this cost is exponential in the induced width of the *cost network*, the undirected graph defined over the variables $X_1, \ldots, X_n$, with an edge between X_l and X_m if they appear together in one of the original functions f_j. The complexity of this algorithm is, of course, dependent on the variable elimination order and the problem structure. Computing the optimal elimination order is an NP-hard problem [Arnborg *et al.*, 1987] and elimination orders yielding low induced tree width do not exist for some problems. These issues have been confronted successfully for a large variety of practical problems in the Bayesian network community, which has benefited from a large variety of good heuristics which have been developed for the variable elimination ordering problem [Bertele & Brioschi, 1972; Kjaerulff, 1990; Reed, 1992; Becker & Geiger, 2001].

4.3 Factored LP

In this section, we present the centerpiece of our planning algorithms: a new, general approach for compactly representing exponentially-large sets of LP constraints in problems

```
VARIABLEELIMINATION ($\mathcal{F}$, $\mathcal{O}$, ELIMOPERATOR)
        // $\mathcal{F} = \{f_1, \ldots, f_m\}$ is the set of functions.
        // $\mathcal{O}$ stores the elimination order.
        // ELIMOPERATOR is the operation used when eliminating variables.
    FOR $i = 1$ TO NUMBER OF VARIABLES:
        // Select the next variable to be eliminated.
        LET $l = \mathcal{O}(i)$ .
        // Select the relevant functions.
        LET $\mathcal{E} = \{e_1, \ldots, e_L\}$ BE THE FUNCTIONS IN $\mathcal{F}$ WHOSE SCOPE CONTAINS $X_l$.
        // Eliminate current variable $X_l$.
        LET $e =$ ELIMOPERATOR $(\mathcal{E}, X_l)$.
        // Update set of functions.
        UPDATE THE SET OF FUNCTIONS $\mathcal{F} = \mathcal{F} \cup \{e\} \setminus \{e_1, \ldots, e_L\}$.
    // Now, all functions have empty scopes, and the last step eliminates the empty set.
    RETURN ELIMOPERATOR $(\mathcal{F}, \emptyset)$.
```

Figure 4.1: Variable elimination procedure, where ELIMOPERATOR is used when a variable is eliminated. To compute the maximum value of $f_1 + \cdots + f_m$, where each f_i is a restricted-scope function, we must substitute ELIMOPERATOR with MAXOUT.

```
MAXOUT $(\mathcal{E}, X_l)$
        // $\mathcal{E} = \{e_1, \ldots, e_m\}$ is the set of functions to be maximized.
        // $X_l$ variable to be maximized.
    LET $f = \sum_{j=1}^{L} e_j$.
    IF $X_l = \emptyset$:
        LET $e = f$.
    ELSE:
        DEFINE A NEW FUNCTION $e = \max_{x_l} f$; NOTE THAT
         $\mathrm{SCOPE}[e] = \cup_{j=1}^{L} \mathrm{SCOPE}[e_j] - \{X_l\}$.
    RETURN $e$.
```

Figure 4.2: MAXOUT operator for variable elimination, procedure that maximizes variable X_l from functions $e_1 + \cdots + e_m$.

with factored structure — those where the functions in the constraints can be decomposed as the sum of restricted-scope functions. Consider our original problem of representing the non-linear constraint in Equation (4.2) compactly. Recall that we wish to represent the non-linear constraint $\phi \geq \max_{\mathbf{x}} \sum_i w_i \; c_i(\mathbf{x}) - b(\mathbf{x})$, or equivalently, $\phi \geq \max_{\mathbf{x}} F^{\mathbf{w}}(\mathbf{x})$, without generating one constraint for each state as in Equation (4.1). The new, key insight is that this non-linear constraint can be implemented using a construction that follows the structure of variable elimination in cost networks.

Consider any function e used within $\mathcal{F}$ (including the original f_i's), and let $\mathbf{Z}$ be its scope. For any assignment $\mathbf{z}$ to $\mathbf{Z}$, we introduce a variable $u^e_{\mathbf{z}}$, whose value represents $e_{\mathbf{z}}$, into the linear program. For the initial functions $f_i^{\mathbf{w}}$, we include the constraint that $u^{f_i}_{\mathbf{z}} = f_i^{\mathbf{w}}(\mathbf{z})$. As $f_i^{\mathbf{w}}$ is linear in $\mathbf{w}$, this constraint is linear in the LP variables. Now, consider a new function e introduced into $\mathcal{F}$ by eliminating a variable X_l. Let $e_1, \ldots, e_L$ be the functions extracted from $\mathcal{F}$, where each e_j has scope restricted to $\mathbf{Z}_j$, and let $\mathbf{Z} = \bigcup_j \mathbf{Z}_j$ be the scope of the resulting e. We introduce a set of constraints:

$$u^e_{\mathbf{z}} \geq \sum_{j=1}^{L} u^{e_j}_{(\mathbf{z},x_l)[\mathbf{Z}_j]} \qquad \forall x_l. \tag{4.3}$$

Let e_n be the last (empty scope) function generated in the elimination, and recall that its scope is empty. Hence, we have only a single variable u^{e_n}. We introduce the additional constraint $\phi \geq u^{e_n}$.

The complete algorithm, presented in Figure 4.3, is divided into three parts: First, we generate equality constraints for functions that depend on the weights w_i (basis functions). In the second part, we add the equality constraints for functions that do not depend on the weights (target functions). These equality constraints let us abstract away the differences between these two types of functions and manage them in a unified fashion in the third part of the algorithm. This third part follows a procedure similar to variable elimination described in Figure 4.1. However, unlike standard variable elimination where we would introduce a new function e, such that $e = \max_{x_l} \sum_{j=1}^{L} e_j$, in our factored LP procedure we introduce new LP variables $u^e_{\mathbf{z}}$. To enforce the definition of e as the maximum over X_l of $\sum_{j=1}^{L} e_j$, we introduce the new LP constraints in Equation (4.3).

```
FACTOREDLP ($C$, $\mathbf{b}$,$\mathcal{O}$)
        // $C = \{c_1, \ldots, c_k\}$ is the set of basis functions.
        // $\mathbf{b} = \{b_1, \ldots, b_m\}$ is the set of target functions.
        // $\mathcal{O}$ stores the elimination order.
        // Return a (polynomial) set of constraints $\Omega$ equivalent to $\phi \geq \sum_i w_i c_i(\mathbf{x}) + \sum_j b_j(\mathbf{x}), \forall \mathbf{x}$ .
    // Data structure for the constraints in factored LP.
    LET $\Omega = \{\}$ .
    // Data structure for the intermediate functions generated in variable elimination.
    LET $\mathcal{F} = \{\}$ .
    // Generate equality constraint to abstract away basis functions.
    FOR EACH $c_i \in C$:
        LET $\mathbf{Z} = \mathrm{SCOPE}[c_i]$.
        FOR EACH ASSIGNMENT $\mathbf{z} \in \mathbf{Z}$, CREATE A NEW LP VARIABLE $u_{\mathbf{z}}^{f_i}$ AND ADD A CON-
          STRAINT TO $\Omega$:
                $$u_{\mathbf{z}}^{f_i} = w_i c_i(\mathbf{z}).$$
        STORE NEW FUNCTION $f_i$ TO USE IN VARIABLE ELIMINATION STEP: $\mathcal{F} = \mathcal{F} \cup \{f_i\}$.
    // Generate equality constraint to abstract away target functions.
    FOR EACH $b_j \in \mathbf{b}$:
        LET $\mathbf{Z} = \mathrm{SCOPE}[b_j]$.
        FOR EACH ASSIGNMENT $\mathbf{z} \in \mathbf{Z}$, CREATE A NEW LP VARIABLE $u_{\mathbf{z}}^{f_j}$ AND ADD A CON-
          STRAINT TO $\Omega$:
                $$u_{\mathbf{z}}^{f_j} = b_j(\mathbf{z}).$$
        STORE NEW FUNCTION $f_j$ TO USE IN VARIABLE ELIMINATION STEP: $\mathcal{F} = \mathcal{F} \cup \{f_j\}$.
    // Now, $\mathcal{F}$ contains all of the functions involved in the LP, our constraints become: $\phi \geq$
      $\sum_{e_i \in \mathcal{F}} e_i(\mathbf{x}), \forall \mathbf{x}$ , which we represent compactly using a variable elimination procedure.
    FOR $i = 1$ TO NUMBER OF VARIABLES:
        // Select the next variable to be eliminated.
        LET $l = \mathcal{O}(i)$ .
        // Select the relevant functions.
        LET $e_1, \ldots, e_L$ BE THE FUNCTIONS IN $\mathcal{F}$ WHOSE SCOPE CONTAINS $X_l$, AND LET $\mathbf{Z}_j =$
          $\mathrm{SCOPE}[e_j]$.
        // Introduce linear constraints for the maximum over current variable $X_l$.
        DEFINE A NEW FUNCTION $e$ WITH SCOPE $\mathbf{Z} = \cup_{j=1}^{L} \mathbf{Z}_j - \{X_l\}$ TO REPRESENT
          $\max_{x_l} \sum_{j=1}^{L} e_j$.
        ADD CONSTRAINTS TO $\Omega$ TO ENFORCE MAXIMUM: FOR EACH ASSIGNMENT $\mathbf{z} \in \mathbf{Z}$:
                $$u_{\mathbf{z}}^{e} \geq \sum_{j=1}^{L} u_{(\mathbf{z},x_l)[\mathbf{Z}_j]}^{e_j} \qquad \forall x_l.$$
        // Update set of functions.
        UPDATE THE SET OF FUNCTIONS $\mathcal{F} = \mathcal{F} \cup \{e\} \setminus \{e_1, \ldots, e_L\}$.
    // Now, all variables have been eliminated and all functions have empty scope.
    ADD LAST CONSTRAINT TO $\Omega$:    $\phi \geq \sum_{e_i \in \mathcal{F}} e_i$.
    RETURN $\Omega$.
```

Figure 4.3: Factored LP algorithm for the compact representation of the exponential set of constraints $\phi \geq \sum_i w_i c_i(\mathbf{x}) + \sum_j b_j(\mathbf{x}), \forall \mathbf{x}$.

Example 4.3.1 *To understand this construction, consider the LP formed when using the simple functions in Example 4.2.1 above, and assume we want to express the fact that* $\phi \geq \max_{\mathbf{x}} F^{\mathbf{w}}(\mathbf{x})$. *We first introduce a set of variables* $u^{f_1}_{x_1,x_2}$ *for every instantiation of values* x_1, x_2 *to the variables* X_1, X_2. *Thus, if* X_1 *and* X_2 *are both binary, we have four such variables. We then introduce equality constraints defining the value of* $u^{f_1}_{x_1,x_2}$ *appropriately. For example, if* f_1 *is an indicator weighted by* w_1 *that takes value* 1 *if* $X_1 = t$ *and* $X_2 = f$, *and* 0 *otherwise, we have* $u^{f_1}_{t,t} = 0$, $u^{f_1}_{t,f} = w_1$, *and so on. We have similar variables and constraints for each* f_j *and each value* $\mathbf{z}$ *in* $\mathbf{Z}_j$. *Note that each of the constraints is a simple equality constraint involving numerical constants and perhaps the weight variables* $\mathbf{w}$.

Next, we introduce variables for each of the intermediate expressions generated by variable elimination. For example, when eliminating X_4, *we introduce a set of LP variables* $u^{e_1}_{x_2,x_3}$; *for each of them, we have a set of constraints*

$$u^{e_1}_{x_2,x_3} \geq u^{f_3}_{x_2,x_4} + u^{f_4}_{x_3,x_4}$$

one for each value x_4 *of* X_4. *We have a similar set of constraint for* $u^{e_2}_{x_1,x_2}$ *in terms of* $u^{f_2}_{x_1,x_3}$ *and* $u^{e_1}_{x_2,x_3}$. *Note that each constraint is a simple linear inequality.* ▮

We can now prove that our factored LP construction represents the same constraint as the non-linear constraint in Equation (4.2):

Theorem 4.3.2 *The constraints generated by the factored LP construction are equivalent to the non-linear constraint in Equation (4.2). That is, an assignment to* $(\phi, \mathbf{w})$ *satisfies the factored LP constraints if and only if it satisfies the constraint in Equation (4.2).*
Proof: *See Appendix A.2.* ▮

Returning to our original formulation, we have that $\sum_j f^{\mathbf{w}}_j$ is $C\mathbf{w} - \mathbf{b}$ in the original set of constraints. Hence our new set of constraints is equivalent to the original set: $\phi \geq \max_{\mathbf{x}} \sum_i w_i \, c_i(\mathbf{x}) - b(\mathbf{x})$ in Equation (4.2), which in turn is equivalent to the exponential set of constraints $\phi \geq \sum_i w_i \, c_i(\mathbf{x}) - b(\mathbf{x}), \forall \mathbf{x}$ in Equation (4.1). Thus, we can represent this exponential set of constraints by a new set of constraints and LP variables. The size of this new set, as in variable elimination, is exponential only in the induced width of the cost network, rather than in the total number of variables.

4.4 Factored max-norm projection

We can now use our procedure for representing the exponential number of constraints in Equation (4.1) compactly to compute efficient max-norm projections, as in Equation (2.11):

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|C\mathbf{w} - \mathbf{b}\|_\infty .$$

The max-norm projection is computed by the linear program in (2.12). There are two sets of constraints in this LP: $\phi \geq \sum_{j=1}^{k} c_{ij} w_j - b_i, \forall i$ and $\phi \geq b_i - \sum_{j=1}^{k} c_{ij} w_j, \forall i$. Each of these sets is an instance of the constraints in Equation (4.1), which we have just addressed in the previous section. Thus, if each of the k basis functions in C is a restricted-scope function and the target function $\mathbf{b}$ is the sum of restricted-scope functions, then we can use our factored LP technique to represent the constraints in the max-norm projection LP compactly. The correctness of our algorithm is a corollary of Theorem 4.3.2:

Corollary 4.4.1 *The solution $(\phi^*, \mathbf{w}^*)$ of a linear program that minimizes ϕ subject to the constraints in* FACTOREDLP*(C, $-\mathbf{b}$,$\mathcal{O}$) and* FACTOREDLP*($-C$, $\mathbf{b}$,$\mathcal{O}$), for any elimination order $\mathcal{O}$ satisfies:*

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|C\mathbf{w} - \mathbf{b}\|_\infty , \quad \textit{and} \quad \phi^* = \min_{\mathbf{w}} \|C\mathbf{w} - \mathbf{b}\|_\infty . \quad \blacksquare$$

The original max-norm projection LP had $k+1$ variables and two constraints for each state $\mathbf{x}$; thus, the number of constraints is exponential in the number of state variables. On the other hand, our new factored max-norm projection LP has more variables, but exponentially fewer constraints. The number of variables and constraints in the new factored LP is exponential only in the number of state variables in the largest factor in the cost network, rather than exponential in the total number of state variables. As we show in Section 5.4.1, this exponential gain allows us to compute max-norm projections efficiently when solving very large factored MDPs.

4.5 Discussion and related work

Both of the approximate solution algorithms presented in Chapter 2 use linear programs to obtain the value function coefficients. These LPs contain one constraint for each joint assignment of the state variables. In this chapter, we present factored LPs, a novel LP decomposition technique, which allows us to represent an LP with an exponentially-large set of constraints by a provably equivalent, polynomially-sized LP. This decomposition relies on the assumption that each constraint is defined by the sum of functions whose scope is restricted to a subset of the state variables. The complexity of our decomposition technique is exponential only in the induced width of a cost network defined by the local functions in the constraints.

Many algorithms have been proposed for tackling exponentially-large constraint sets. The book by Bertsimas and Tsitsiklis [1997] presents many typical approaches. An interesting option is the use of the delayed constraint generation, or cutting planes, method. Schuurmans and Patrascu [2001], building on our factored LP approach, propose one such algorithm, where variable elimination cost network is used to find violated constraints. As they use this approach in the context of the SIMPLEX algorithm, their method does not offer our polynomial complexity guarantees. However, in light of the extension of Schuurmans and Patrascu [2001], we can view variable elimination as a polynomial time *separation oracle* for finding violated constraints. Such an oracle guarantees polynomial time complexity of the ellipsoid method for solving LPs [Bertsimas & Tsitsiklis, 1997, Theorem 8.5]. Thus, such cutting planes method can also yield a polynomial implementation of our exponentially-large LPs. We present further discussion in Section 7.8.

The closest approach to our factored LP is the LP transformation method of Yannakakis [1991]. He tackles the problem of optimizing a linear function over a polytope that may contain exponentially many facets. Yannakakis shows that, for some examples, this exponentially-large polytope can be described as a reduced, polynomially-sized, LP by adding a new set of variables and constraints, as we do in our approach. He also proves that if the underlying polytope represents a travelling salesman problem, then the reduced LP requires exponentially many constraints, unless P=NP. Maximization in a cost network is obviously an NP-complete problem, thus, the reduced polytope will also require an exponential description, in general. Our factored LP method focuses on exploiting local structure in the constraints to generate an analogous decomposition with a polynomial description, in problems that have fixed induced width.

We believe that the LP decomposition technique presented in this chapter allows the compact representation of many practical optimization problems. In the next part of this thesis, we will apply this technique to optimize the weights of our factored value function approximation very efficiently.

Part II

Approximate planning for structured single-agent systems

Chapter 5

Efficient planning algorithms

Recall that, as described in Chapter 3, we seek to find linear approximations to the value function of the form:

$$\mathcal{V}^{\mathbf{w}}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}),$$

where each h_i is a restricted scope function. Once these weights $\mathbf{w}$ are obtained (by any approach), the agent can select its action in some state $\mathbf{x}$ by simply computing the greedy action with respect to this approximate value function, which is again given by:

$$\text{Greedy}[\mathcal{V}^{\mathbf{w}}](\mathbf{x}) = \arg\max_a Q_a^{\mathbf{w}}(\mathbf{x}) = \arg\max_a R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i h_i(\mathbf{x}').$$

The Q_a function for each action can be computed efficiently, in single agent problems with factored value functions, as described in Section 3.3. Thus, the greedy policy can always be represent implicitly by the Q-function, given $\mathbf{w}$. Therefore, this part of the thesis focuses on designing efficient planning algorithms for optimizing such weights $\mathbf{w}$.

In this chapter, we present two planning algorithms, which exploit structure in a factored MDP to compute approximate solutions very efficiently: factored linear programming-based approximation, and factored approximate policy iteration with max-norm projection. Each algorithm is presented in a self-contained section, which can thus be read independently. Finally, we present an efficient algorithm for computing a bound on the quality of the greedy policies obtained from factored value functions.

5.1 Factored linear programming-based approximation

We begin with the simplest of our approximate MDP solution algorithms, based on the linear programming-based approximation formulation in Section 2.3.2. Using the LP decomposition technique in Chapter 4, we can formulate an algorithm, which is both simple and efficient.

5.1.1 The algorithm

As discussed in Section 2.3.2, the linear programming-based approximation formulation is based on the exact linear programming approach to solving MDPs presented in Section 2.2.1. However, in this approximate version, we restrict the space of value functions to the linear space defined by our basis functions. More precisely, in this approximate LP formulation, the variables are $w_1, \ldots, w_k$ — the weights for our basis functions. The LP is given by:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k \; ; \\ \text{Minimize:} & \sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \; h_i(\mathbf{x}) \; ; \\ \text{Subject to:} & \sum_i w_i \; h_i(\mathbf{x}) \geq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i \; h_i(\mathbf{x}') \quad \forall \mathbf{x} \in \mathbf{X}, \; a \in A. \end{array} \tag{5.1}$$

In other words, this formulation takes the LP in (2.4) and substitutes the explicit state value function with a linear value function representation $\sum_i w_i \; h_i(\mathbf{x})$. This transformation from an exact to an approximate problem formulation has the effect of reducing the number of free variables in the LP to k (one for each basis function coefficient), but the number of constraints remains $|\mathbf{X}| \times |A|$. In our *SysAdmin* problem in Example 2.1.1, for example, the number of constraints in the LP in (5.1) is $(m+1) \cdot 2^m$, where m is the number of machines in the network. However, using our algorithm for representing exponentially-large constraint sets compactly we are able to compute the solution to this linear programming-based approximation algorithm in *closed form* with an exponentially smaller LP, as in Chapter 4.

First, consider the objective function $\sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \; h_i(\mathbf{x})$ of the LP (5.1). Naively representing this objective function requires a summation over an exponentially-large state

```
FACTOREDLPA ($P$, $R$, $\gamma$, $H$, $\mathcal{O}$, $\alpha$)
        // $P$ is the factored transition model.
        // $R$ is the set of factored reward functions.
        // $\gamma$ is the discount factor.
        // $H$ is the set of basis functions $H = \{h_1, \ldots, h_k\}$.
        // $\mathcal{O}$ stores the elimination order.
        // $\alpha$ are the state relevance weights.
        // Return the basis function weights $\mathbf{w}$ computed by linear programming-based approximation.
    // Cache the backprojections of the basis functions.
    FOR EACH BASIS FUNCTION $h_i \in H$; FOR EACH ACTION $a$:
        LET $g_i^a = Backproj_a(h_i)$.
    // Compute factored state relevance weights.
    FOR EACH BASIS FUNCTION $h_i$, COMPUTE THE FACTORED STATE RELEVANCE WEIGHTS $\alpha_i$
      AS IN EQUATION (5.2) .
    // Generate linear programming-based approximation constraints.
    LET $\Omega = \{\}$.
    FOR EACH ACTION $a$:
        LET $\Omega = \Omega \cup$ FACTOREDLP$(\{\gamma g_1^a - h_1, \ldots, \gamma g_k^a - h_k\}, R^a, \mathcal{O})$.
    // So far, our constraints guarantee that $\phi \geq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i\ h_i(\mathbf{x}') -$
      $\sum_i w_i\ h_i(\mathbf{x})$; to satisfy the linear programming-approximation solution in (5.1) we must add
      a final constraint.
    LET $\Omega = \Omega \cup \{\phi = 0\}$.
    // We can now obtain the solution weights by solving an LP.
    LET $\mathbf{w}$ BE THE SOLUTION OF THE LINEAR PROGRAM: MINIMIZE $\sum_i \alpha_i w_i$, SUBJECT TO THE
      CONSTRAINTS $\Omega$.
    RETURN $\mathbf{w}$.
```

Figure 5.1: Factored linear programming-based approximation algorithm.

space. However, we can rewrite the objective and obtain a compact representation. We first reorder the terms:

$$\sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \, h_i(\mathbf{x}) = \sum_i w_i \sum_{\mathbf{x}} \alpha(\mathbf{x}) \, h_i(\mathbf{x}).$$

Now, consider the state relevance weights $\alpha(\mathbf{x})$ as a distribution over states, so that $\alpha(\mathbf{x}) > 0$ and $\sum_{\mathbf{x}} \alpha(\mathbf{x}) = 1$. As with the backprojections in Section 3.3, we can now write:

$$\alpha_i = \sum_{\mathbf{x}} \alpha(\mathbf{x}) \, h_i(\mathbf{x}) = \sum_{\mathbf{c}_i \in \mathbf{C}_i} \alpha(\mathbf{c}_i) \, h_i(\mathbf{c}_i), \tag{5.2}$$

where $\alpha(\mathbf{c}_i)$ represents the marginal of the state relevance weights α over the domain $\mathrm{Dom}[\mathbf{C}_i]$ of the basis function h_i. For example, if we use uniform state relevance weights as in our experiments — $\alpha(\mathbf{x}) = \frac{1}{|\mathbf{X}|}$ — then the marginals become $\alpha(\mathbf{c}_i) = \frac{1}{|\mathbf{C}_i|}$. Thus, we can rewrite the objective function as $\sum_i w_i \, \alpha_i$, where each basis weight α_i is computed as shown in Equation (5.2). If the state relevance weights are represented by marginals, then the cost of computing each α_i depends exponentially on the size of the scope of $\mathbf{C}_i$ only, rather than exponentially on the number of state variables. On the other hand, if the state relevance weights are represented by arbitrary distributions, we need to obtain the marginals over the $\mathbf{C}_i$'s, which may not be an efficient computation. Thus, best results are achieved by using a compact representation, such as a Bayesian network, for the state relevance weights.

Second, note that the right side of the constraints in the LP (5.1) correspond to the Q_a functions:

$$Q_a(\mathbf{x}) = R^a(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i \, h_i(\mathbf{x}').$$

Using the efficient backprojection operation in factored MDPs described in Section 3.3 we can rewrite the Q_a functions as:

$$Q_a(\mathbf{x}) = R^a(\mathbf{x}) + \gamma \sum_i w_i \, g_i^a(\mathbf{x}) \, ,$$

where g_i^a is the backprojection of basis function h_i through the transition model P_a. As we discussed, if h_i has scope restricted to $\mathbf{C}_i$, then g_i^a is a restricted scope function of $\Gamma_a(\mathbf{C}'_i)$.

We can precompute the backprojections g_i^a and the basis relevance weights α_i. The linear programming-based approximation LP of (5.1) can then be written as:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k \ ; \\ \text{Minimize:} & \sum_i \alpha_i \, w_i \ ; \\ \text{Subject to:} & \sum_i w_i \, h_i(\mathbf{x}) \geq R^a(\mathbf{x}) + \gamma \sum_i w_i \, g_i^a(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbf{X}, \forall a \in A. \end{array} \tag{5.3}$$

Finally, we can rewrite this LP to use constraints of the same form as the one in Equation (4.2):

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k \ ; \\ \text{Minimize:} & \sum_i \alpha_i \, w_i \ ; \\ \text{Subject to:} & 0 \geq \max_{\mathbf{x}} \left\{ R^a(\mathbf{x}) + \sum_i w_i \ \left[\gamma g_i^a(\mathbf{x}) - h_i(\mathbf{x}) \right] \right\} \quad \forall a \in A. \end{array} \tag{5.4}$$

We can now use our factored LP construction in Chapter 4 to represent these non-linear constraints compactly. Basically, there is one set of factored LP constraints for each action a. Specifically, we can write the non-linear constraint in the same form as those in Equation (4.2) by expressing the functions C as: $c_i(\mathbf{x}) = h_i(\mathbf{x}) - \gamma g_i^a(\mathbf{x})$. Each $c_i(\mathbf{x})$ is a restricted-scope function; that is, if $h_i(\mathbf{x})$ has scope restricted to $\mathbf{C}_i$, then $g_i^a(\mathbf{x})$ has scope restricted to $\Gamma_a(\mathbf{C}'_i)$, which means that $c_i(\mathbf{x})$ has scope restricted to $\mathbf{C}_i \cup \Gamma_a(\mathbf{C}'_i)$. Next, the target function $\mathbf{b}$ becomes the reward function $R^a(\mathbf{x})$ which, by assumption, is factored. Finally, in the constraint in Equation (4.2), ϕ is a free variable. On the other hand, in the LP in (5.4) the maximum in the right hand side must be less than zero. This final condition can be achieved by adding a constraint $\phi = 0$. Thus, our algorithm generates a set of factored LP constraints, one for each action. The total number of constraints and variables in this new LP is linear in the number of actions $|A|$ and only exponential in the induced width of each cost network, rather than in the total number of variables. The complete factored linear programming-based approximation algorithm is outlined in Figure 5.1.

5.1.2 An example

We now present a complete example of the operations required by the approximate LP algorithm to solve the factored MDP shown in Figure 3.1(a). Our presentation follows four steps: problem representation, basis function selection, backprojections and LP construction.

Problem representation: First, we must fully specify the factored MDP model for the problem. The structure of the DBN is shown in Figure 3.1(b). This structure is maintained for all action choices. Next, we must define the transition probabilities for each action. There are 5 actions in this problem: do nothing, or reboot one of the 4 machines in the network. The CPDs for these actions are shown in Figure 3.1(c). Finally, we must define the reward function. We decompose the global reward as the sum of 4 local reward functions, one for each machine, such that there is a reward if the machine is working. Specifically, $R_i(X_i = \textit{W}) = 1$ and $R_i(X_i = \textit{D}) = 0$, breaking symmetry by setting $R_4(X_4 = \textit{true}) = 2$. We use a discount factor of $\gamma = 0.9$.

Basis function selection: In this simple example, we use five simple basis functions. First, we include the constant function $h_0 = 1$. Next, we add indicators for each machine which take value 1 if the machine is working: $h_i(X_i = \textit{W}) = 1$ and $h_i(X_i = \textit{D}) = 0$.

Backprojections: The first algorithmic step is computing the backprojection of the basis functions, as defined in Section 3.3. The backprojection of the constant basis is simple:

$$\begin{aligned} g_0^a &= \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) h_0 \; ; \\ &= \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) \, 1 \; ; \\ &= 1 \; . \end{aligned}$$

Next, we must backproject of each indicator basis functions h_i. We repeat the derivation of this computation for completeness:

$$
\begin{aligned}
g_i^a &= \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) h_i(x_i') \; ; \\
&= \sum_{x_1',x_2',x_3',x_4'} \prod_j P_a(x_j' \mid x_{j-1}, x_j) h_i(x_i') \; ; \\
&= \sum_{x_i'} P_a(x_i' \mid x_{i-1}, x_i) h_i(x_i') \sum_{\mathbf{x}'[\mathbf{X}'-\{X_i'\}]} \prod_{j \neq i} P_a(x_j' \mid x_{j-1}, x_j) \; ; \\
&= \sum_{x_i'} P_a(x_i' \mid x_{i-1}, x_i) h_i(x_i') \; ; \\
&= P_a(X_i' = \mathit{W} \mid x_{i-1}, x_i)\, 1 + P_a(X_i' = \mathit{D} \mid x_{i-1}, x_i)\, 0 \; ; \\
&= P_a(X_i' = \mathit{W} \mid x_{i-1}, x_i) \; .
\end{aligned}
$$

Thus, g_i^a is a restricted-scope function of $\{X_{i-1}, X_i\}$. We can now use the CPDs in Figure 3.1(c) to specify g_i^a:

$g_i^{\mathit{reboot} = i}(X_{i-1}, X_i) =$

	$X_i = \mathit{W}$	$X_i = \mathit{D}$
$X_{i-1} = \mathit{W}$	1	1
$X_{i-1} = \mathit{D}$	1	1

;

$g_i^{\mathit{reboot} \neq i}(X_{i-1}, X_i) =$

	$X_i = \mathit{W}$	$X_i = \mathit{D}$
$X_{i-1} = \mathit{W}$	0.9	0.09
$X_{i-1} = \mathit{D}$	0.5	0.05

.

LP construction: To illustrate the factored LPs constructed by our algorithms, we define the constraints for the linear programming-based approximation approach presented above. First, we define the functions $c_i^a = \gamma g_i^a - h_i$, as shown in Equation (5.4). In our example, these functions are $c_0^a = \gamma - 1 = -0.1$ for the constant basis, and for the indicator bases:

$$c_i^{\textit{reboot}\,=\,i}(X_{i-1},X_i) \;=\; \begin{array}{|c|c|c|}\hline & X_i = W & X_i = D \\ \hline X_{i-1} = W & -0.1 & 0.9 \\ \hline X_{i-1} = D & -0.1 & 0.9 \\ \hline \end{array}\;;$$

$$c_i^{\textit{reboot}\,\neq\,i}(X_{i-1},X_i) \;=\; \begin{array}{|c|c|c|}\hline & X_i = W & X_i = D \\ \hline X_{i-1} = W & -0.19 & 0.081 \\ \hline X_{i-1} = D & -0.55 & 0.045 \\ \hline \end{array}\;.$$

Using this definition of c_i^a, the linear programming-based approximation constraints are given by:

$$0 \geq \max_{\mathbf{x}} \sum_i R_i + \sum_j w_j c_j^a \;\;, \;\forall a \;. \tag{5.5}$$

We present the LP construction for one of the 5 actions: *reboot* $= 1$. Analogous constructions can be made for the other actions.

In the first set of constraints, we abstract away the difference between rewards and basis functions by introducing LP variables u and equality constraints. We begin with the reward functions:

$$\begin{array}{ll} u_{x_1}^{R_1} = 1 \;,\; u_{\bar{x}_1}^{R_1} = 0 \;; & \quad u_{x_2}^{R_2} = 1 \;,\; u_{\bar{x}_2}^{R_2} = 0 \;; \\ u_{x_3}^{R_3} = 1 \;,\; u_{\bar{x}_3}^{R_3} = 0 \;; & \quad u_{x_4}^{R_4} = 2 \;,\; u_{\bar{x}_4}^{R_4} = 0 \;. \end{array}$$

We now represent the equality constraints for the c_j^a functions for the *reboot* $= 1$ action. Note that the appropriate basis function weight from Equation (5.5) appears in these constraints:

$$\begin{array}{rclrclrclrcl}
u^{c_0} & = & -0.1\,w_0 \;; & & & & & & & & & \\
u^{c_1}_{x_1,x_4} & = & -0.1\,w_1 \;, & u^{c_1}_{\bar{x}_1,x_4} & = & 0.9\,w_1 \;, & u^{c_1}_{x_1,\bar{x}_4} & = & -0.1\,w_1 \;, & u^{c_1}_{\bar{x}_1,\bar{x}_4} & = & 0.9\,w_1 \;; \\
u^{c_2}_{x_1,x_2} & = & -0.19\,w_2 \;, & u^{c_2}_{\bar{x}_1,x_2} & = & -0.55\,w_2 \;, & u^{c_2}_{x_1,\bar{x}_2} & = & 0.081\,w_2 \;, & u^{c_2}_{\bar{x}_1,\bar{x}_2} & = & 0.045\,w_2 \;; \\
u^{c_3}_{x_2,x_3} & = & -0.19\,w_3 \;, & u^{c_3}_{\bar{x}_2,x_3} & = & -0.55\,w_3 \;, & u^{c_3}_{x_2,\bar{x}_3} & = & 0.081\,w_3 \;, & u^{c_3}_{\bar{x}_2,\bar{x}_3} & = & 0.045\,w_3 \;; \\
u^{c_4}_{x_3,x_4} & = & -0.19\,w_4 \;, & u^{c_4}_{\bar{x}_3,x_4} & = & -0.55\,w_4 \;, & u^{c_4}_{x_3,\bar{x}_4} & = & 0.081\,w_4 \;, & u^{c_4}_{\bar{x}_3,\bar{x}_4} & = & 0.045\,w_4 \;.
\end{array}$$

Using these new LP variables, our LP constraint from Equation (5.5) for the *reboot* $= 1$ action becomes:

$$0 \geq \max_{x_1,x_2,x_3,x_4} \sum_{i=1}^{4} u^{R_i}_{X_i} + u^{c_0} + \sum_{j=1}^{4} u^{c_j}_{X_{j-1},X_j} .$$

We are now ready for the variable elimination process. We illustrate the elimination of variable X_4:

$$0 \geq \max_{x_1,x_2,x_3} \sum_{i=1}^{3} u^{R_i}_{X_i} + u^{c_0} + \sum_{j=2}^{3} u^{c_j}_{X_{j-1},X_j} + \max_{x_4} \left[u^{R_4}_{X_4} + u^{c_1}_{X_1,X_4} + u^{c_4}_{X_3,X_4} \right] .$$

We can represent the term $\max_{X_4} \left[u^{R_4}_{X_4} + u^{c_1}_{X_1,X_4} + u^{c_4}_{X_3,X_4} \right]$ by a set of linear constraints, one for each assignment of X_1 and X_3, using the new LP variables $u^{e_1}_{X_1,X_3}$ to represent this maximum:

$$\begin{array}{rcl}
u^{e_1}_{x_1,x_3} & \geq & u^{R_4}_{x_4} + u^{c_1}_{x_1,x_4} + u^{c_4}_{x_3,x_4} \; ; \\
u^{e_1}_{x_1,x_3} & \geq & u^{R_4}_{\bar{x}_4} + u^{c_1}_{x_1,\bar{x}_4} + u^{c_4}_{x_3,\bar{x}_4} \; ; \\
u^{e_1}_{\bar{x}_1,x_3} & \geq & u^{R_4}_{x_4} + u^{c_1}_{\bar{x}_1,x_4} + u^{c_4}_{x_3,x_4} \; ; \\
u^{e_1}_{\bar{x}_1,x_3} & \geq & u^{R_4}_{\bar{x}_4} + u^{c_1}_{\bar{x}_1,\bar{x}_4} + u^{c_4}_{x_3,\bar{x}_4} \; ; \\
u^{e_1}_{x_1,\bar{x}_3} & \geq & u^{R_4}_{x_4} + u^{c_1}_{x_1,x_4} + u^{c_4}_{\bar{x}_3,x_4} \; ; \\
u^{e_1}_{x_1,\bar{x}_3} & \geq & u^{R_4}_{\bar{x}_4} + u^{c_1}_{x_1,\bar{x}_4} + u^{c_4}_{\bar{x}_3,\bar{x}_4} \; ; \\
u^{e_1}_{\bar{x}_1,\bar{x}_3} & \geq & u^{R_4}_{x_4} + u^{c_1}_{\bar{x}_1,x_4} + u^{c_4}_{\bar{x}_3,x_4} \; ; \\
u^{e_1}_{\bar{x}_1,\bar{x}_3} & \geq & u^{R_4}_{\bar{x}_4} + u^{c_1}_{\bar{x}_1,\bar{x}_4} + u^{c_4}_{\bar{x}_3,\bar{x}_4} \; .
\end{array}$$

We have now eliminated variable X_4 and our global non-linear constraint becomes:

$$0 \geq \max_{X_1,X_2,X_3} \sum_{i=1}^{3} u^{R_i}_{X_i} + u^{c_0} + \sum_{j=2}^{3} u^{c_j}_{X_{j-1},X_j} + u^{e_1}_{X_1,X_3} .$$

Next, we eliminate variable X_3. The new LP constraints and variables have the form:

$$u^{e_2}_{X_1,X_2} \geq u^{R_3}_{X_3} + u^{c_3}_{X_2,X_3} + u^{e_1}_{X_1,X_3} \; , \; \forall \; X_1, X_2, X_3 \; ;$$

thus removing X_3 from the global non-linear constraint:

$$0 \geq \max_{X_1,X_2} \sum_{i=1}^{2} u_{X_i}^{R_i} + u^{c_0} + u_{X_1,X_2}^{c_2} + u_{X_1,X_2}^{e_2} \; .$$

We can now eliminate X_2, generating the linear constraints:

$$u_{X_1}^{e_3} \geq u_{X_2}^{R_2} + u_{X_1,X_2}^{c_2} + u_{X_1,X_2}^{e_2} \; , \; \forall \, X_1, X_2 \; .$$

Now, our global non-linear constraint involves only X_1:

$$0 \geq \max_{X_1} u_{X_1}^{R_1} + u^{c_0} + u_{X_1}^{e_3} \; .$$

As X_1 is the last variable to be eliminated, the scope of the new LP variable is empty and the linear constraints are given by:

$$u^{e_4} \geq u_{X_1}^{R_1} + u_{X_1}^{e_3} \; , \; \forall \, X_1 \; .$$

All of the state variables have now been eliminated, turning our global non-linear constraint into a simple linear constraint:

$$0 \geq u^{c_0} + u^{e_4} \; ,$$

which completes the LP description for the linear programming-based approximation solution to the problem in Figure 3.1.

In this small example with only four state variables, our factored LP technique generates a total of 89 equality constraints, 115 inequality constraints and 149 LP variables, while the explicit state representation in Equation (2.8) generates only 80 inequality constraints and 5 LP variables. However, as the problem size increases, the number of constraints and LP variables in our factored LP approach grow as $O(n^2)$, while the explicit state approach grows exponentially, at $O(n2^n)$. This scaling effect is illustrated in Figure 5.2.

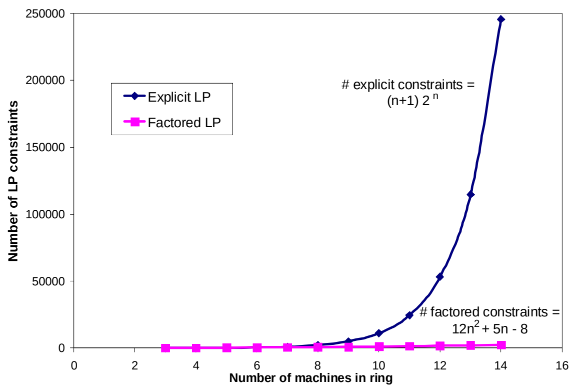


Figure 5.2: Number of constraints in the LP generated by the explicit state representation versus the factored LP-based approximation algorithm.

5.2 Factored approximate policy iteration with max-norm projection

The factored LP-based approximation approach described in the previous section is both elegant and easy to implement. However, we cannot, in general, provide strong guarantees about the error it achieves. An alternative is to use the approximate policy iteration described in Section 2.3.3, which does offer certain bounds on the error. However, as we shall see, this algorithm is significantly more complicated, and requires that we place additional restrictions on the factored MDP.

In particular, approximate policy iteration requires a representation of the policy at each iteration. In order to obtain a compact policy representation, we must make an additional assumption: each action only affects a small number of state variables. We first state this assumption formally. Then, we show how to obtain a compact representation of the greedy policy with respect to a factored value function, under this assumption. Finally, we describe our factored approximate policy iteration algorithm using max-norm projections.

5.2.1 Default action model

In Chapter 3, we presented the factored MDP model, where each action is associated with its own factored transition model represented as a DBN and with its own factored reward function. However, different actions often have very similar transition dynamics, only differing in their effect on some small set of variables. In particular, in many cases a variable has a default evolution model, which only changes if an action affects it directly [Boutilier *et al.*, 2000].

This type of structure turns out to be useful for compactly representing policies, a property which is important in our approximate policy iteration algorithm. Thus, in this section of the thesis, we restrict attention to factored MDPs that are defined using a *default transition model* $\tau_d = \langle G_d, P_d \rangle$ [Koller & Parr, 2000]. For each action a, we define $\mathit{Effects}[a] \subseteq \mathbf{X}'$ to be the variables in the next state whose local probability model is different from τ_d, *i.e.*, those variables X'_i such that $P_a(X'_i \mid \mathsf{Parents}_a(X'_i)) \neq P_d(X'_i \mid \mathsf{Parents}_d(X'_i))$.

Example 5.2.1 *In our system administrator example, we have an action a_i for rebooting each one of the machines, and a default action d for doing nothing. The transition model described above corresponds to the "do nothing" action, which is also the default transition model. The transition model for a_i is different from d only in the transition model for the variable X'_i, which is now $X'_i =$ W with probability one, regardless of the status of the neighboring machines. Thus, in this example, $\mathit{Effects}[a_i] = X'_i$.* ▮

As in the transition dynamics, we can also define the notion of *default reward model*. In this case, there is a set of reward functions $\sum_{i=1}^{r} R_i(\mathbf{W}_i)$ associated with the default action d. In addition, each action a can have a reward function $R^a(\mathbf{W}^a)$. Here, the extra reward of action a has scope restricted to $\mathit{Rewards}[a] = \mathbf{W}_i^a \subset \{X_1, \ldots, X_n\}$. Thus, the total reward associated with action a is given by $R^a + \sum_{i=1}^{r} R_i$. Note that R^a can also be factored as a linear combination of smaller terms for an even more compact representation.

5.2.2 Computing greedy policies

We can now build on this additional assumption to define the complete algorithm. Recall that the approximate policy iteration algorithm iterates through two steps: policy improvement and approximate value determination. We now discuss each of these steps.

The policy improvement step computes the greedy policy relative to a value function $\mathcal{V}^{(t-1)}$: $\pi^{(t)} = \mathsf{Greedy}[\mathcal{V}^{(t-1)}]$. Recall that our value function estimates have the linear form $\mathbf{Hw}$. As we described in Section 3.3, the greedy policy for this type of value function is given by:

$$\text{Greedy}[\mathbf{Hw}](\mathbf{x}) = \arg\max_a Q_a(\mathbf{x}),$$

where each Q_a can be represented by:

$$Q_a(\mathbf{x}) = R(\mathbf{x}, a) + \sum_i w_i \, g_i^a(\mathbf{x}).$$

If we attempt to represent this policy naively, we are again faced with the problem of exponentially-large state spaces. Fortunately, as shown by Koller and Parr [2000], the greedy policy relative to a factored value function has the form of a *decision list*. More precisely, the policy can be written in the form $\langle \mathbf{t}_1, a_1 \rangle, \langle \mathbf{t}_2, a_2 \rangle, \ldots, \langle \mathbf{t}_L, a_L \rangle$, where each $\mathbf{t}_i$ is an assignment of values to some small subset $\mathbf{T}_i$ of variables, and each a_i is an action. The greedy action to take in state $\mathbf{x}$ is the action a_j corresponding to the first event $\mathbf{t}_j$ in the list with which $\mathbf{x}$ is consistent. For completeness, we now review the construction of this decision-list policy.

The critical assumption that allows us to represent the policy as a compact decision list is the default action assumption described in Section 5.2.1. Under this assumption, the Q_a functions can be written as:

$$Q_a(\mathbf{x}) = R^a(\mathbf{x}) + \sum_{i=1}^{r} R_i(\mathbf{x}) + \sum_i w_i \, g_i^a(\mathbf{x}),$$

where R^a has scope restricted to $\mathbf{W}^a$. The Q function for the default action d is just: $Q_d(\mathbf{x}) = \sum_{i=1}^{r} R_i(\mathbf{x}) + \sum_i w_i \, g_i^d(\mathbf{x})$.

We now have a set of linear Q-functions which implicitly describes a policy π. It is not immediately obvious that these Q functions result in a compactly expressible policy. An important insight is that most of the components in the weighted combination are identical, so that g_i^a is equal to g_i^d for most i. Intuitively, a component g_i^a corresponding to the backprojection of basis function $h_i(\mathbf{C}_i)$ is only different if the action a influences one of the variables in $\mathbf{C}_i$. More formally, assume that $\mathit{Effects}[a] \cap \mathbf{C}_i = \emptyset$. In this case, all of the variables in $\mathbf{C}_i$ have the same transition model in τ_a and τ_d. Thus, we have that $g_i^a(\mathbf{x}) = g_i^d(\mathbf{x})$; in other words, the ith component of the Q_a function is irrelevant when deciding whether action a is better than the default action d. We can define which components are actually relevant: let I_a be the set of indices i such that $\mathit{Effects}[a] \cap \mathbf{C}_i \neq \emptyset$. These are the indices of those basis functions whose backprojection differs in P_a and P_d. In our example SysAdmin DBN of Figure 3.1, action a_i reboots machine i, thus a_i only affects the CPD of X_i'. As only the basis function h_i depends on X_i, we have that $I_{a_i} = i$.

Let us now consider the impact of taking action a over the default action d. We can define the impact — the difference in value — as:

$$\begin{aligned} \delta_a(\mathbf{x}) &= Q_a(\mathbf{x}) - Q_d(\mathbf{x}); \\ &= R^a(\mathbf{x}) + \sum_{i \in I_a} w_i \left[g_i^a(\mathbf{x}) - g_i^d(\mathbf{x}) \right]. \end{aligned} \tag{5.6}$$

This analysis shows that $\delta_a(\mathbf{x})$ is a function whose scope is restricted to

$$\mathbf{T}_a = \mathbf{W}^a \cup \left[\cup_{i \in I_a} \Gamma_a(\mathbf{C}_i') \right]. \tag{5.7}$$

In our example DBN, $\mathbf{T}_{a_2} = \{X_1, X_2\}$.

Intuitively, we now have a situation where we have a "baseline" value function $Q_d(\mathbf{x})$ which defines a value for each state $\mathbf{x}$. Each action a changes that baseline by adding or subtracting an amount from each state. The point is that this amount depends only on $\mathbf{T}_a$, so that it is the same for all states in which the variables in $\mathbf{T}_a$ take the same values.

We can now define the greedy policy relative to our Q functions. For each action a, define a set of *conditionals* $\langle \mathbf{t}, a, \delta \rangle$, where each $\mathbf{t}$ is some assignment of values to the variables $\mathbf{T}_a$, and δ is $\delta_a(\mathbf{t})$. Now, sort all of the conditionals for all of the actions by order of decreasing δ:

$$\langle \mathbf{t}_1, a_1, \delta_1 \rangle, \langle \mathbf{t}_2, a_2, \delta_2 \rangle, \ldots, \langle \mathbf{t}_L, a_L, \delta_L \rangle$$

Consider our optimal action in a state $\mathbf{x}$. We would like to get the largest possible "bonus" over the default value. If $\mathbf{x}$ is consistent with $\mathbf{t}_1$, we should clearly take action a_1, as it gives us bonus δ_1. If not, then we should try to get δ_2; thus, we should check if $\mathbf{x}$ is consistent with $\mathbf{t}_2$, and if so, take a_2. Using this procedure, we can compute the decision-list policy associated with our linear estimate of the value function. The complete algorithm

```
DECISIONLISTPOLICY ($Q_a$)
        // $Q_a$ is the set of Q-functions, one for each action;
        // Return the decision-list policy $\Delta$.
    LET $\Delta = \{\}$.
    // Compute the bonus functions.
    FOR EACH ACTION $a$, OTHER THAN THE DEFAULT ACTION $d$:
        COMPUTE THE BONUS FOR TAKING ACTION $a$,

                $\delta_a(\mathbf{x}) = Q_a(\mathbf{x}) - Q_d(\mathbf{x})$;

         AS IN EQUATION (5.6). NOTE THAT $\delta_a$ HAS SCOPE RESTRICTED TO $\mathbf{T}_a$, AS IN EQUATION (5.7).
        // Add states with positive bonuses to the (unsorted) decision list.
        FOR EACH ASSIGNMENT $\mathbf{t} \in \mathbf{T}_a$:
            IF $\delta_a(\mathbf{t}) > 0$, ADD BRANCH TO DECISION LIST:

                $\Delta = \Delta \cup \{\langle \mathbf{t}, a, \delta_a(\mathbf{t}) \rangle\}$.

    // Add the default action to the (unsorted) decision list.
    LET $\Delta = \Delta \cup \{\langle \emptyset, d, 0 \rangle\}$.
    // Sort decision list to obtain final policy.
    SORT THE DECISION LIST $\Delta$ IN DECREASING ORDER ON THE $\delta$ ELEMENT OF $\langle \mathbf{t}, a, \delta \rangle$.
    RETURN $\Delta$.
```

Figure 5.3: Method for computing the decision-list policy Δ from the factored representation of the Q_a functions.

for computing the decision-list policy is summarized in Figure 5.3.

Note that the number of conditionals in the list is $\sum_a |\mathrm{Dom}(\mathbf{T}_a)|$; $\mathbf{T}_a$, in turn, depends on the set of basis function clusters that intersect with the effects of a. Thus, the size of the policy depends in a natural way on the interaction between the structure of our process description and the structure of our basis functions. In problems where the actions modify a large number of variables, the policy representation could become unwieldy. The linear programming-based approximation approach in Section 5.1 is more appropriate in such cases, as requires an independent factored LP construction for the DBN of each action, and not for a particular policy. Thus, no explicit representation of the policy is necessary.

5.2.3 Value determination

In the approximate value determination step our algorithm computes:

```
FactoredAPI ($P$, $R$, $\gamma$, $H$, $\mathcal{O}$, $\varepsilon$, $t_{max}$)
        // $P$ is the factored transition model.
        // $R$ is the set of factored reward functions.
        // $\gamma$ is the discount factor.
        // $H$ is the set of basis functions $H = \{h_1, \ldots, h_k\}$.
        // $\mathcal{O}$ stores the elimination order.
        // $\varepsilon$ is the Bellman error precision.
        // $t_{max}$ is the maximum number of iterations.
        // Return the basis function weights $\mathbf{w}$ computed by approximate policy iteration.
    // Initialize weights
    LET $\mathbf{w}^{(0)} = \mathbf{0}$.
    // Cache the backprojections of the basis functions.
    FOR EACH BASIS FUNCTION $h_i \in H$; FOR EACH ACTION $a$:
        LET $g_i^a = Backproj_a(h_i)$.
    // Main approximate policy iteration loop.
    LET $t = 0$.
    REPEAT
      // Policy improvement part of the loop.
        // Compute decision list policy for iteration $t$ weights.
        LET $\Delta^{(t)} = $ DecisionListPolicy$(R^a + \gamma \sum_i w_i^{(t)} g_i^a)$.
      // Value determination part of the loop.
        // Initialize constraints for max-norm projection LP, and indicators.
        LET $\Omega^+ = \{\}$, $\Omega^- = \{\}$, AND $\mathcal{I} = \{\}$.
        // For every branch of the decision list policy, generate the relevant set of constraints, and
          update the indicators to constraint the state space for future branches.
        FOR EACH BRANCH $\langle \mathbf{t}_j, a_j \rangle$ IN THE DECISION LIST POLICY $\Delta^{(t)}$:
            // Instantiate the variables in $\mathbf{T}_j$ to the assignment given in $\mathbf{t}_j$.
            INSTANTIATE THE SET OF FUNCTIONS $\{h_1 - \gamma g_1^{a_j}, \ldots, h_k - \gamma g_k^{a_j}\}$ WITH THE
              PARTIAL STATE ASSIGNMENT $\mathbf{t}_j$ AND STORE IN $C$.
            INSTANTIATE THE TARGET FUNCTIONS $R^{a_j}$ WITH THE PARTIAL STATE ASSIGN-
              MENT $\mathbf{t}_j$ AND STORE IN $\mathbf{b}$.
            INSTANTIATE THE INDICATOR FUNCTIONS $\mathcal{I}$ WITH THE PARTIAL STATE ASSIGN-
              MENT $\mathbf{t}_j$ AND STORE IN $\mathcal{I}'$.
            // Generate the factored LP constraints for the current decision list branch.
            LET $\Omega^+ = \Omega^+ \cup$ FactoredLP$(C, -\mathbf{b} + \mathcal{I}', \mathcal{O})$.
            LET $\Omega^- = \Omega^- \cup$ FactoredLP$(-C, \mathbf{b} + \mathcal{I}', \mathcal{O})$.
            // Update the indicator functions.
            LET $\mathcal{I}_j(\mathbf{x}) = -\infty \mathbb{1}(\mathbf{x} = \mathbf{t}_j)$ AND UPDATE THE INDICATORS $\mathcal{I} = \mathcal{I} \cup \mathcal{I}_j$.
        // We can now obtain the new set of weights by solving an LP, which corresponds to the
          max-norm projection.
        LET $\mathbf{w}^{(t+1)}$ BE THE SOLUTION OF THE LINEAR PROGRAM: MINIMIZE $\phi$, SUBJECT TO
          THE CONSTRAINTS $\{\Omega^+, \Omega^-\}$.
        LET $t = t + 1$.
    UNTIL BellmanErr$(\mathbf{H}\mathbf{w}^{(t)}) \leq \varepsilon$ OR $t \geq t_{max}$ OR $\mathbf{w}^{(t-1)} = \mathbf{w}^{(t)}$.
    RETURN $\mathbf{w}^{(t)}$.
```

Figure 5.4: Factored approximate policy iteration with max-norm projection algorithm.

$$\mathbf{w}^{(t)} = \arg\min_{\mathbf{w}} \left\| \mathbf{H}\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathbf{H}\mathbf{w}) \right\|_{\infty}.$$

By rearranging the expression, we get:

$$\mathbf{w}^{(t)} = \arg\min_{\mathbf{w}} \left\| \left(\mathbf{H} - \gamma P_{\pi^{(t)}} \mathbf{H}\right) \mathbf{w} - R_{\pi^{(t)}} \right\|_{\infty}.$$

This equation is an instance of the optimization in Equation (2.11). If $P_{\pi^{(t)}}$ is factored, we can conclude that $C = (\mathbf{H} - \gamma P_{\pi^{(t)}} \mathbf{H})$ is also a matrix whose columns correspond to restricted-scope functions. More specifically:

$$c_i(\mathbf{x}) = h_i(\mathbf{x}) - \gamma g_i^{\pi^{(t)}}(\mathbf{x});$$

where $g_i^{\pi^{(t)}}$ is the backprojection of the basis function h_i through the transition model $P_{\pi^{(t)}}$, as described in Section 3.3. The target $\mathbf{b} = R_{\pi^{(t)}}$ corresponds to the reward function, which for the moment is assumed to be factored. Thus, we can again apply our factored LP in Section 4.4 to estimate the value of the policy $\pi^{(t)}$.

Unfortunately, the transition model $P_{\pi^{(t)}}$ is not factored, as a decision list representation for the policy $\pi^{(t)}$ will, in general, induce a transition model $P_{\pi^{(t)}}$ which cannot be represented by a compact DBN. Nonetheless, we can still generate a compact LP by exploiting the decision list structure of the policy. The basic idea is to introduce cost networks corresponding to each branch in the decision list, ensuring, additionally, that only states consistent with this branch are considered in the cost network maximization. Specifically, we have a factored LP construction for each branch $\langle \mathbf{t}_i, a_i \rangle$. The ith cost network only considers a subset of the states that is consistent with the ith branch of the decision list. Let S_i be the set of states $\mathbf{x}$ such that $\mathbf{t}_i$ is the first event in the decision list for which $\mathbf{x}$ is consistent. That is, for each state $\mathbf{x} \in S_i$, $\mathbf{x}$ is consistent with $\mathbf{t}_i$, but it is *not* consistent with any $\mathbf{t}_j$ with $j < i$.

Recall that, as in Equation (4.1), our LP construction defines a set of constraints, which imply that $\phi \geq \sum_i w_i \; c_i(\mathbf{x}) - b(\mathbf{x})$ for each state $\mathbf{x}$. Instead, we have a separate set of constraints for the states in each subset S_i. For each state in S_i, we know that action a_i is taken. Hence, we can apply our construction above using P_{a_i} — a transition model which is factored by assumption — in place of the non-factored $P_{\pi^{(t)}}$. Similarly, the reward function becomes $R^{a_i}(\mathbf{x}) + \sum_{i=1}^{r} R_i(\mathbf{x})$ for this subset of states.

The only issue is to guarantee that the cost network constraints derived from this transition model are applied only to states in S_i. Specifically, we must guarantee that they are applied only to states consistent with $\mathbf{t}_i$, but not to states that are consistent with some $\mathbf{t}_j$ for $j < i$. To guarantee the first condition, we simply instantiate the variables in $\mathbf{T}_i$ to take the values specified in $\mathbf{t}_i$. That is, our cost network now considers only the variables in $\{X_1, \ldots, X_n\} - \mathbf{T}_i$, and computes the maximum only over the states consistent with $\mathbf{T}_i = \mathbf{t}_i$. To guarantee the second condition, we ensure that we do not impose any constraints on states associated with previous decisions. This is achieved by adding indicators $\mathcal{I}_j$ for each previous decision $\mathbf{t}_j$, with weight $-\infty$. More specifically, $\mathcal{I}_j$ is a function that takes value $-\infty$ for states consistent with $\mathbf{t}_j$ and zero for other all assignments of $\mathbf{T}_j$. The constraints for the ith branch will be of the form:

$$\phi \geq R(\mathbf{x}, a_i) + \sum_{l} w_l \left(\gamma g_l(\mathbf{x}, a_i) - h(\mathbf{x})\right) + \sum_{j<i} -\infty \mathbb{1}(\mathbf{x} = \mathbf{t}_j), \qquad \forall \mathbf{x} \sim [\mathbf{t}_i], \quad (5.8)$$

where $\mathbf{x} \sim [\mathbf{t}_i]$ defines the assignments of $\mathbf{X}$ consistent with $\mathbf{t}_i$. The introduction of these indicators causes the constraints associated with $\mathbf{t}_i$ to be trivially satisfied by states in S_j for $j < i$. Note that each of these indicators is a restricted-scope function of $\mathbf{T}_j$ and can be handled in the same fashion as all other terms in the factored LP. Thus, for a decision list of size L, our factored LP contains constraints from $2L$ cost networks. The complete approximate policy iteration with max-norm projection algorithm is outlined in Figure 5.4.

5.3 Computing bounds on policy quality

We have presented two algorithms for computing approximate solutions to factored MDPs. All these algorithms generate linear value functions which can be denoted by $\mathbf{H}\widehat{\mathbf{w}}$, where $\widehat{\mathbf{w}}$ are the resulting basis function weights. In practice, the agent will define its behavior by acting according to the greedy policy $\widehat{\pi} = \mathsf{Greedy}[\mathbf{H}\widehat{\mathbf{w}}]$. One issue that remains is how this policy $\widehat{\pi}$ compares to the true optimal policy π^*; that is, how the *actual* value $\mathcal{V}_{\widehat{\pi}}$ of policy $\widehat{\pi}$ compares to $\mathcal{V}^*$.

In Section 2.3, we showed some *a priori* bounds for the quality of the policy. Another

```
FactoredBellmanErr ($P$, $R$, $\gamma$, $H$, $\mathcal{O}$, $\widehat{\mathbf{w}}$)
        // $P$ is the factored transition model.
        // $R$ is the set of factored reward functions.
        // $\gamma$ is the discount factor.
        // $H$ is the set of basis functions $H = \{h_1, \ldots, h_k\}$.
        // $\mathcal{O}$ stores the elimination order.
        // $\widehat{\mathbf{w}}$ are the weights for the linear value function.
        // Return the Bellman error for the value function $\mathbf{H}\widehat{\mathbf{w}}$.
    // Cache the backprojections of the basis functions.
    For each basis function $h_i \in H$; for each action $a$:
        Let $g_i^a = Backproj_a(h_i)$.
    // Compute decision list policy for value function $\mathbf{H}\widehat{\mathbf{w}}$.
    Let $\widehat{\Delta} = $ DecisionListPolicy$(R^a + \gamma \sum_i \widehat{w}_i g_i^a)$.
    // Initialize indicators.
    Let $\mathcal{I} = \{\}$.
    // Initialize Bellman error.
    Let $\varepsilon = 0$.
    // For every branch of the decision list policy, generate the relevant cost networks, solve it with
      variable elimination, and update the indicators to constraint the state space for future branches.
    For each branch $\langle \mathbf{t}_j, a_j \rangle$ in the decision list policy $\widehat{\Delta}$:
        // Instantiate the variables in $\mathbf{T}_j$ to the assignment given in $\mathbf{t}_j$.
        Instantiate the set of functions $\{\widehat{w}_1(h_1 - \gamma g_1^{a_j}), \ldots, \widehat{w}_k(h_k - \gamma g_k^{a_j})\}$ with the
          partial state assignment $\mathbf{t}_j$ and store in $C$.
        Instantiate the target functions $R^{a_j}$ with the partial state assignment
          $\mathbf{t}_j$ and store in $\mathbf{b}$.
        Instantiate the indicator functions $\mathcal{I}$ with the partial state assignment
          $\mathbf{t}_j$ and store in $\mathcal{I}'$.
        // Use variable elimination to solve first cost network, and update Bellman error, if error for
          this branch is larger.
        Let $\varepsilon = \max(\varepsilon,$ VariableElimination$(C - \mathbf{b} + \mathcal{I}', \mathcal{O}))$.
        // Use variable elimination to solve second cost network, and update Bellman error, if error
          for this branch is larger.
        Let $\varepsilon = \max(\varepsilon,$ VariableElimination$(-C + \mathbf{b} + \mathcal{I}', \mathcal{O}))$.
        // Update the indicator functions.
        Let $\mathcal{I}_j(\mathbf{x}) = -\infty \mathbb{1}(\mathbf{x} = \mathbf{t}_j)$ and update the indicators $\mathcal{I} = \mathcal{I} \cup \mathcal{I}_j$.
    Return $\varepsilon$.
```

Figure 5.5: Algorithm for computing Bellman error for factored value function $\mathbf{H}\widehat{\mathbf{w}}$.

possible procedure is to compute an *a posteriori* bound. That is, given our resulting weights $\widehat{\mathbf{w}}$, we compute a bound on the loss of acting according to the greedy policy $\widehat{\pi}$ rather than the optimal policy. This can be achieved by using the *Bellman error* analysis of Williams and Baird [1993].

The *Bellman error* is defined as $\mathrm{BellmanErr}(\mathcal{V}) = \|\mathcal{T}^*\mathcal{V} - \mathcal{V}\|_\infty$. Given the greedy policy $\widehat{\pi} = \textsf{Greedy}[\mathcal{V}]$, their analysis provides the bound of Theorem 2.1.5:

$$\|\mathcal{V}^* - \mathcal{V}_{\widehat{\pi}}\|_\infty \leq \frac{2\gamma \mathrm{BellmanErr}(\mathcal{V})}{1-\gamma}. \tag{5.9}$$

Thus, we can use the Bellman error $\mathrm{BellmanErr}(\mathbf{H}\widehat{\mathbf{w}})$ to evaluate the quality of our resulting greedy policy.

Note that computing the Bellman error involves a maximization over the state space. Thus, the complexity of this computation grows exponentially with the number of state variables. Koller and Parr [2000] suggested that structure in the factored MDP can be exploited to compute the Bellman error efficiently. Here, we show how this error bound can be computed by a set of cost networks using a similar construction to the one in our max-norm projection algorithms. This technique can be used for any $\widehat{\pi}$ that can be represented as a decision list and does not depend on the algorithm used to determine the policy. Thus, we can apply this technique to solutions determined by the linear programming-based approximation algorithm if the action descriptions permit a decision list representation of the policy.

For some set of weights $\widehat{\mathbf{w}}$, the Bellman error is given by:

$$\begin{aligned} \mathrm{BellmanErr}(\mathbf{H}\widehat{\mathbf{w}}) &= \|\mathcal{T}^*\mathbf{H}\widehat{\mathbf{w}} - \mathbf{H}\widehat{\mathbf{w}}\|_\infty ; \\ &= \max \begin{pmatrix} \max_{\mathbf{x}} \sum_i w_i h_i(\mathbf{x}) - R_{\widehat{\pi}}(\mathbf{x}) - \gamma \sum_{\mathbf{x}'} P_{\widehat{\pi}}(\mathbf{x}' \mid \mathbf{x}) \sum_j w_j h_j(\mathbf{x}') \,, \\ \max_{\mathbf{x}} R_{\widehat{\pi}}(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} P_{\widehat{\pi}}(\mathbf{x}' \mid \mathbf{x}) \sum_j w_j h_j(\mathbf{x}') - \sum_i w_i h_i(\mathbf{x}) \end{pmatrix} . \end{aligned}$$

If the rewards $R_{\widehat{\pi}}$ and the transition model $P_{\widehat{\pi}}$ are factored appropriately, then we can compute each one of these two maximizations ($\max_{\mathbf{x}}$) using variable elimination in a cost network as described in Section 4.2. However, $\widehat{\pi}$ is a decision list policy and it does not induce a factored transition model. Fortunately, as in the approximate policy iteration algorithm in Section 5.2, we can exploit the structure in the decision list to perform such a maximization efficiently. In particular, as in approximate policy iteration, we will generate two cost networks for each branch in the decision list. To guarantee that our maximization is performed only over states where this branch is relevant, we include the same type of indicator functions, which will force irrelevant states to have a value of $-\infty$, thus guaranteeing that at each point of the decision list policy we obtain the corresponding state with the maximum error. The state with the overall largest Bellman error will be the maximum over the ones generated for each point the in the decision list policy. The complete factored algorithm for computing the Bellman error is outlined in Figure 5.5.

One last interesting note concerns our approximate policy iteration algorithm with max-norm projection of Section 5.2. In all our experiments, this algorithm converged, so that $\mathbf{w}^{(t)} = \mathbf{w}^{(t+1)}$ after some iterations. If such convergence occurs, then the objective function $\phi^{(t+1)}$ of the linear program in our last iteration is equal to the Bellman error of the final policy:

Lemma 5.3.1 *If approximate policy iteration with max-norm projection converges, so that $\mathbf{w}^{(t)} = \mathbf{w}^{(t+1)}$ for some iteration t, then the max-norm projection error $\phi^{(t+1)}$ of the last iteration is equal to the Bellman error for the final value function estimate $\mathbf{H}\widehat{\mathbf{w}} = \mathbf{H}\mathbf{w}^{(t)}$:*

$$\mathrm{BellmanErr}(\mathbf{H}\widehat{\mathbf{w}}) = \phi^{(t+1)}.$$

Proof: *See Appendix A.3.* ▮

Thus, we can bound the loss of acting according to the final policy $\pi^{(t+1)}$ by substituting $\phi^{(t+1)}$ into the Bellman error bound:

Corollary 5.3.2 *If approximate policy iteration with max-norm projection converges after t iterations to a final value function estimate $\mathbf{H}\widehat{\mathbf{w}}$ associated with a greedy policy $\widehat{\pi} =$ Greedy$[\mathbf{H}\widehat{\mathbf{w}}]$, then the loss of acting according to $\widehat{\pi}$ instead of the optimal policy π^* is bounded by:*

$$\|\mathcal{V}^* - \mathcal{V}_{\widehat{\pi}}\|_\infty \leq \frac{2\gamma\phi^{(t+1)}}{1-\gamma},$$

where $\mathcal{V}_{\widehat{\pi}}$ is the actual *value of the policy $\widehat{\pi}$.* ▮

Therefore, when approximate policy iteration converges we obtain a bound on the quality of the resulting policy without a special purpose computation of the Bellman error.

5.4 Empirical evaluation

The factored representation of a value function is most appropriate in certain types of systems: Systems that involve many variables, but where the strong interactions between the variables are fairly sparse, so that the decoupling of the influence between variables does not induce an unacceptable loss in accuracy. As discussed in Chapter 1 and argued by Simon [1981], many complex systems have a nearly decomposable, hierarchical structure, with the subsystems interacting only weakly between themselves. Throughout this thesis, to evaluate our algorithms, we selected problems, which we believe to exhibit this type of structure.

5.4.1 Scaling properties

In order to evaluate the scaling properties of our factored algorithms, we tested our approaches the SysAdmin problem described in detail in Chapter 7. This problem relates to a system administrator who has to maintain a network of computers; we experimented with various network architectures, shown in Figure 2.1. Machines fail randomly, and a faulty machine increases the probability that its neighboring machines will fail. At every time step, the SysAdmin can go to one machine and reboot it, causing it to be working in the next time step with high probability. Recall that the state space in this problem grows exponentially in the number of machines in the network, that is, a problem with m machines has 2^m states. Each machine receives a reward of 1 when working (except in the ring, where one machine receives a reward of 2, to introduce some asymmetry), a zero reward is given to faulty machines, and the discount factor is $\gamma = 0.95$. The optimal strategy for rebooting machines will depend upon the topology, the discount factor, and the status of the machines in the network. If machine i and machine j are both faulty, the benefit of rebooting i must be weighed against the expected discounted impact of delaying rebooting j on j's successors. For many network topologies, this policy may be a function of the status of every single machine in the network.

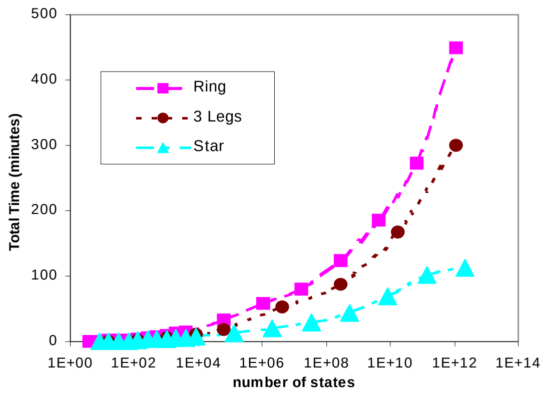
The basis functions we used include independent indicators for each machine, with value 1 if it is working and zero otherwise (*i.e.*, each one is a restricted-scope function of a single variable), and the constant basis, whose value is 1 for all states. We selected straightforward variable elimination orders: for the “Star” and “Three Legs” topologies, we first eliminated the variables corresponding to computers in the legs, and the center computer (server) was eliminated last; for “Ring”, we started with an arbitrary computer and followed the ring order; for “Ring and Star”, the ring machines were eliminated first and then the center one; finally, for the “Ring of Rings” topology, we eliminated the computers in the outer rings first and then the ones in the inner ring.

We implemented the factored policy iteration and linear programming algorithms in Matlab, using CPLEX as the LP solver. Experiments were performed on a Sun UltraSPARC-II, 359 MHz with 256MB of RAM.

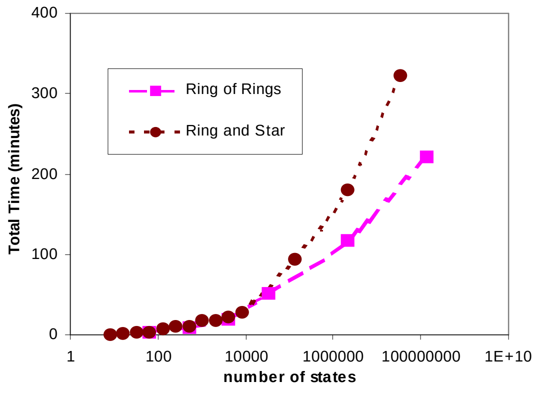
We first evaluated the complexity of our algorithms, tests were performed with increasing the number of states, that is, increasing number of machines on the network. Figure 5.6 shows the running time for increasing problem sizes, for various architectures. The simplest one is the “Star”, where the backprojection of each basis function has scope restricted to two variables and the largest factor in the cost network has scope restricted to two variables. The most difficult one was the “Bidirectional Ring”, where factors contain five variables.

Note that the number of states grows exponentially (indicated by the log scale in Figure 5.6), but running times increase only logarithmically in the number of states, or polynomially in the number of variables. We illustrate this behavior in Figure 5.6(d), where we fit a 3rd order polynomial to the running times for the “unidirectional ring”, where the factors generated by variable elimination included up to 3 variables at a time. Note that the size of the problem description grows quadratically with the number of variables: adding a machine to the network also adds the possible action of fixing that machine. For this problem, the computation cost of our factored algorithm empirically grows approximately as $O\left((n \cdot |A|)^{1.5}\right)$, for a problem with n variables, as opposed to the exponential complexity — $poly\left(2^n, |A|\right)$ — of the explicit algorithm.

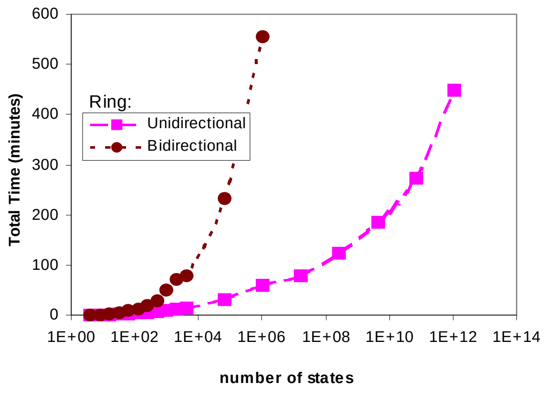
Next, we measured the error in our approximate value function relative to the true optimal value function $\mathcal{V}^*$. Note that it is only possible to compute $\mathcal{V}^*$ for small problems; in our case, we were only able to go up to 10 machines. Here, we used two types of basis functions: the same single variable functions, and pairwise basis functions. The pairwise basis functions contain indicators for neighboring pairs of machines (*i.e.*, functions of two



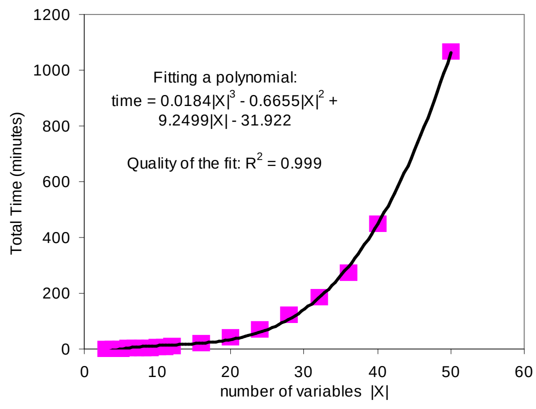
(a)



(b)



(c)



(d)

Figure 5.6: Results of approximate policy iteration with max-norm projection on variants of the SysAdmin problem: (a)–(c) Running times; (d) Fitting a polynomial to the running time for the “Ring” topology.

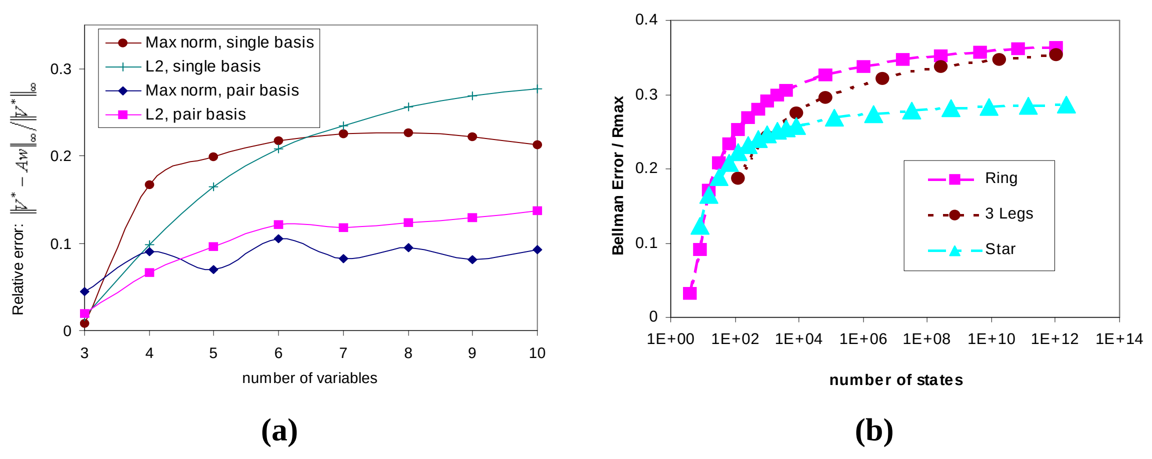


Figure 5.7: Quality of the solutions of approximate policy iteration with max-norm projection: (a) Relative error to optimal value function $\mathcal{V}^*$ and comparison to $\mathcal{L}_2$ projection for "Ring"; (b) For large models, measuring Bellman error after convergence.

variables). As expected, the use of pairwise basis functions resulted in better approximations. For comparison, we also evaluated the error in the approximate value function produced by the $\mathcal{L}_2$-projection algorithm of Koller and Parr [2000]. As we discussed in Section 5.5.1, the $\mathcal{L}_2$ projections in factored MDPs by Koller and Parr are difficult and time consuming; hence, we were only able to compare the two algorithms for smaller problems, where an equivalent $\mathcal{L}_2$-projection can be implemented using an explicit state space formulation. Results for both algorithms are presented in Figure 5.7(a), showing the relative error of the approximate solutions to the true value function for increasing problem sizes. The results indicate that, for larger problems, the max-norm formulation generates a better approximation of the true optimal value function $\mathcal{V}^*$ than the $\mathcal{L}_2$-projection.

For these small problems, we can also compare the actual value of the policy generated by our algorithm to the value of the optimal policy. Here, the value of the policy generated by our algorithm is much closer to the value of the optimal policy than the error implied by the difference between our approximate value function and $\mathcal{V}^*$. For example, for the "Star" architecture with one server and up to 6 clients, our approximation with single variable basis functions had relative error of 12%, but the policy we generated had the same value as the optimal policy. In this case, the same was true for the policy generated by the $\mathcal{L}_2$

projection. In a “Unidirectional Ring” with 8 machines and pairwise basis, the relative error between our approximation and $\mathcal{V}^*$ was about 10%, but the resulting policy only had a 6% loss over the optimal policy. For the same problem, the $\mathcal{L}_2$ approximation has a value function error of 12%, and a true policy loss was 9%. In other words, both methods induce policies that have lower errors than the errors in the approximate value function (at least for small problems). However, our algorithm continues to outperform the $\mathcal{L}_2$ algorithm, even with respect to actual policy loss.

For large models, we can no longer compute the correct value function, so we cannot evaluate our results by computing $\|\mathcal{V}^* - Aw\|_\infty$. Fortunately, as discussed in Section 5.3, the Bellman error can be used to provide a bound on the approximation error and can be computed efficiently by exploiting problem-specific structure. Figure 5.7(b) shows that the Bellman error increases very slowly with the number of states.

It is also valuable to look at the actual decision-list policies generated in our experiments. First, we noted that the lists tended to be short, the length of the final decision list policy grew approximately linearly with the number of machines. Furthermore, the policy itself is often fairly intuitive. In the “Ring and Star” architecture, for example, the decision list says: If the server is faulty, fix the server; else, if another machine is faulty, fix it.

5.4.2 LP-based approximation and approximate PI

Thus far, we have presented scaling results for running times and approximation error for our approximate PI approach. We now compare this algorithm to the simpler approximate LP approach of Section 5.1. As shown in Figure 5.8(a), the approximate LP algorithm for factored MDPs is significantly faster than the approximate PI algorithm. In fact, approximate PI with single-variable basis functions variables is more costly computationally than the LP approach using basis functions over consecutive triples of variables. As shown in Figure 5.8(b), for singleton basis functions, the approximate PI policy obtains slightly better performance for some problem sizes. However, as we increase the number of basis functions for the approximate LP formulation, the value of the resulting policy is much better. Thus, in this problem, our factored linear programming-based approximation formulation allows us to use more basis functions and to obtain a resulting policy of higher

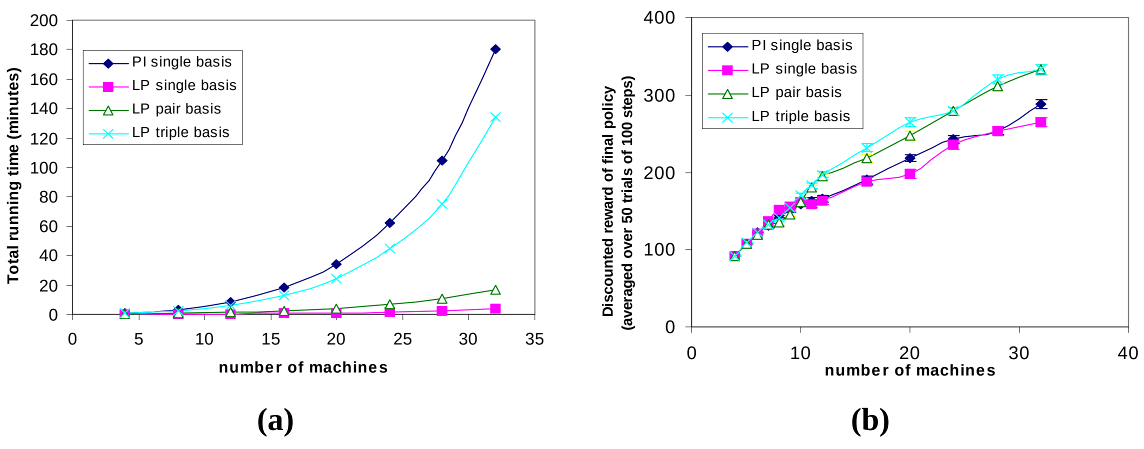


Figure 5.8: Comparing LP-based approximation versus approximate policy iteration on the SysAdmin problem with a "Ring" topology: (a) running time; (b) value of policy estimated by 50 monte carlo runs of 100 steps.

value, while still maintaining a faster running time. These results, along with the simpler implementation, suggest that in practice one may first try to apply the linear programming-based approximation algorithm before deciding to move to the more elaborate approximate policy iteration approach.

5.5 Discussion and related work

In this chapter, we present new algorithms for approximate linear programming and approximate dynamic programming (value and policy iteration) for factored MDPs. Both of these algorithms leverage on the novel LP decomposition technique presented in the previous chapter.

This chapter also presents an efficient factored algorithm for computing the Bellman error. This measure can be used to bound the quality of a greedy policy relative to an approximate value function. Koller and Parr [2000] first suggested that structure in a factored MDP can be exploited to compute the Bellman error efficiently. In this chapter, we present a correct and novel algorithm for computing this bound.

5.5.1 Comparing max-norm and $\mathcal{L}_2$ projections

It is instructive to compare our max-norm policy iteration algorithm to the $\mathcal{L}_2$-projection policy iteration algorithm of Koller and Parr [2000] in terms of computational costs per iteration and implementation complexity. Computing the $\mathcal{L}_2$ projection requires (among other things) a series of dot product operations between basis functions and backprojected basis functions $\langle h_i \bullet g_j^\pi \rangle$. These expressions are easy to compute if P_π refers to the transition model of a particular action a. However, if the policy π is represented as a decision list, as is the result of the factored policy improvement step, then this step becomes much more complicated. In particular, for every branch of the decision list, for every pair of basis functions i and j, and for each assignment to the variables in $\mathsf{Scope}[h_i] \cup \mathsf{Scope}[g_j^a]$, it requires the solution of a counting problem which is $\sharp P$-complete in general. Although Koller and Parr show that this computation can be performed using a Bayesian network (BN) inference, the algorithm still requires a BN inference for each one of those assignments at each branch of the decision list. This makes the algorithm very difficult to implement efficiently in practice.

The max-norm projection, on the other hand, relies on solving a linear program at every iteration. The size of the linear program depends on the cost networks generated. As we discuss, two cost networks are needed for each point in the decision list. The complexity of each of these cost networks is approximately the same as only one of the BN inferences in the counting problem for the $\mathcal{L}_2$ projection. Overall, for each branch in the decision list, we have a total of two of these "inferences", as opposed to one for each assignment of $\mathsf{Scope}[h_i] \cup \mathsf{Scope}[g_j^a]$ for every pair of basis functions i and j. Thus, the max-norm policy iteration algorithm is substantially less complex computationally than the approach based on $\mathcal{L}_2$-projection. Furthermore, the use of linear programming allows us to rely on existing LP packages (such as CPLEX), which are very highly optimized.

In this chapter, we present empirical evaluations demonstrating that, as expected, the running time of our factored algorithms grows polynomially with the number of state variables, for problems with fixed induced width in the underlying cost network. Additionally, we empirically compare our max-norm projection method to the $\mathcal{L}_2$-projection algorithm, demonstrating that the max-norm projection approach seems to generate better policies, in addition to the computational advantages described above.

5.5.2 Comparing linear programming and policy iteration

It is also interesting to compare the approximate policy iteration algorithm and the approximate linear programming algorithm. In the approximate linear programming algorithm, we never need to compute the decision list policy. The policy can always be represented implicitly by the Q_a functions, as discussed in the beginning of this chapter. Thus, this algorithm does not require explicit computation or manipulation of the greedy policy. This difference has two important consequences: one computational and the other in terms of generality.

First, not having to compute or consider the decision lists makes approximate linear programming faster and easier to implement. In this algorithm, we generate a single LP with one cost network for each action and never need to compute a decision list policy. On the other hand, in each iteration, approximate policy iteration needs to generate two LPs for every branch of the decision list of size L, which is usually significantly longer than $|A|$, with a total of $2L$ cost networks. In terms of representation, we do not require the policies to be compact; thus, we do not need to make the default action assumption. Therefore, the approximate linear programming algorithm can deal with a more general class of problems, where each action can have its own independent DBN transition model. On the other hand, as described in Section 2.3.3, approximate policy iteration has stronger guarantees in terms of error bounds.

These differences are further highlighted in our experimental results comparing the two algorithms: empirically, the LP-based approximation algorithm seems to be a favorable option. Our experiments suggest that approximate policy iteration tends to generate better policies for the same set of basis functions. However, due to the computational advantages, we can add more basis functions to the approximate linear programming algorithm, obtaining a better policy and still maintaining a much faster running time than approximate policy iteration.

5.5.3 Summary

Our approximate dynamic programming algorithms are motivated by error analyses in Section 2.3.3 showing the importance of minimizing $\mathcal{L}_\infty$ error. These algorithms are more efficient and substantially easier to implement than previous algorithms based on the $\mathcal{L}_2$-projection. Our experimental results also suggest that max-norm projection performs better in practice.

Our approximate linear programming algorithm for factored MDPs is simpler, easier to implement and more general than the dynamic programming approaches. Unlike our policy iteration algorithm, it does not rely on the default action assumption, which states that actions only affect a small number of state variables. Although this algorithm does not have the same theoretical guarantees as max-norm projection approaches, empirically it seems to be a favorable option. Our experiments suggest that approximate policy iteration tends to generate better policies for the same set of basis functions.

Chapter 6

Factored dual linear programming-based approximation

In this chapter, we describe the formulation and interpretation of both the dual of the linear programming-based approximation algorithm, and of the dual of our factored version of this algorithm. This presentation will yield a very natural interpretation of the factorized dual LP, a new bound on the quality of the solutions obtained by the LP-based approximation approach, and a novel algorithm for approximating problems with large induced width that cannot be solved by our standard LP decomposition technique.

6.1 The approximate dual LP

In Section 2.2.1, we presented an interpretation of the dual of the exact linear programming solution algorithm for MDPs.[1] This exact formulation is, again, given by:

[1]In this thesis, we call the LP formulation in terms of the value function, presented in Equation (2.4), the "primal" formulation, while we refer to the one involving the visitation frequencies, in Equation (2.5), the "dual" formulation. In some presentations by other authors, the latter formulation is called the "primal", as it maximizes the rewards directly.

$$\begin{array}{ll}\text{Variables:} & \phi_a(\mathbf{x}) \ , \ \forall\mathbf{x}, \ \forall a \ ; \\ \text{Maximize:} & \sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a) \ ; \\ \text{Subject to:} & \forall \mathbf{x} \in \mathbf{X}, a \in A \ : \\ & \quad \sum_a \phi_a(\mathbf{x}) = \alpha(\mathbf{x}) + \gamma \sum_{\mathbf{x}', a'} \phi_{a'}(\mathbf{x}') P(\mathbf{x} \mid \mathbf{x}', a') \ ; \\ & \forall \mathbf{x} \in \mathbf{X}, a \in A \ : \\ & \quad \phi_a(\mathbf{x}) \geq 0. \end{array} \tag{6.1}$$

In this section, we present the formulation and interpretation of the dual of the LP-based approximation algorithm, and a new bound on the quality of the policies obtained by the LP-based approximation approach.

6.1.1 Interpretation

We present an interpretation of the dual of the LP-based approximation formulation in (2.8). Similar interpretations have been described in more general settings involving constrained optimizations over visitation frequencies [Derman, 1970]. This section will, however, build the foundation for our bound and novel algorithm.

First, note that the dual of the LP-based approximation formulation in (2.8) is given by:

$$\begin{array}{ll}\text{Variables:} & \phi_a(\mathbf{x}) \ , \ \forall\mathbf{x}, \ \forall a \ ; \\ \text{Maximize:} & \sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a) \ ; \\ \text{Subject to:} & \forall i = 1, \ldots, k \ : \\ & \quad \sum_{\mathbf{x},a} \phi_a(\mathbf{x}) h_i(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_i(\mathbf{x}) + \gamma \sum_{\mathbf{x}', a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_i(\mathbf{x}) \ ; \\ & \forall \mathbf{x} \in \mathbf{X}, a \in A \ : \\ & \quad \phi_a(\mathbf{x}) \geq 0. \end{array} \tag{6.2}$$

At the optimum, the weights w_i of the ith basis function h_i in the primal formulation will be the Lagrange multiplier of the *flow constraint*:

$$\sum_{\mathbf{x},a} \phi_a(\mathbf{x}) h_i(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_i(\mathbf{x}) + \gamma \sum_{\mathbf{x}', a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_i(\mathbf{x}) \tag{6.3}$$

induced by h_i.

Note that both the variables and the objective function of this LP are the same as those in the exact dual LP in (6.1). In this section, we show that the constraints in the approximate version are a relaxation of the constraints in the exact dual LP in (6.1). To understand this property, consider a basis function h_j that takes value 1 for state $\mathbf{x}_j$ and zero for all other states, *i.e.*, $h_j(\mathbf{x}) = \mathbb{1}(\mathbf{x} = \mathbf{x}_j)$. The constraint corresponding to this basis function in the approximate dual LP in (6.2) becomes:

$$\sum_{\mathbf{x},a} \phi_a(\mathbf{x})\mathbb{1}(\mathbf{x} = \mathbf{x}_j) = \sum_{\mathbf{x}} \alpha(\mathbf{x})\mathbb{1}(\mathbf{x} = \mathbf{x}_j) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a')\mathbb{1}(\mathbf{x} = \mathbf{x}_j) \; ;$$

this constraint is equivalent to:

$$\sum_{a} \phi_a(\mathbf{x}_j) = \alpha(\mathbf{x}_j) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}')P(\mathbf{x}_j \mid \mathbf{x}', a') \; .$$

This is exactly the constraint corresponding to state $\mathbf{x}_j$ in the exact dual LP in (6.1). If we had an indicator basis function for every state $\mathbf{x}$, then the approximate dual LP in (6.2) will be exactly equivalent to the exact one in (6.1), as they would have the same set of constraints. Equivalently, the linear subspace formed by our basis functions would include any possible value function, and the approximate LP approach would be exact. In practice, our basis function subspace will not span the whole space of value functions, and, as we will now prove, the constraints in (6.2) will be a relaxation of those in the exact dual LP in (6.1).

First, it is useful to interpret the dual variables ϕ_a as a density function:

Lemma 6.1.1 *Any feasible set of visitation frequencies $\phi_a(\mathbf{x})$ in the approximate dual LP in (6.2) forms a density function over $\mathbf{X} \times A$, that is:*

$$\begin{aligned} \forall \mathbf{x}, a : \;\; \phi_a(\mathbf{x}) &\geq 0 \, , \qquad \textit{and} && (6.4) \\ \sum_{\mathbf{x},a} \phi_a(\mathbf{x}) &= \frac{1}{1-\gamma} \, . && (6.5) \end{aligned}$$

Proof: *See Appendix A.4.1.* ▮

In the exact case in Section 2.2.1, the density represented by the visitation frequencies

$\phi_a(\mathbf{x})$ has a one to one correspondence to policies in the MDP, as shown in Theorem 2.2.1. This correspondence is guaranteed by the constraints in the dual LP in (6.1). Although, in the approximate case, the visitation frequencies $\phi_a(\mathbf{x})$ still form a density as shown by Lemma 6.1.1, we now prove that the constraints have been relaxed, and the one to one correspondence between $\phi_a(\mathbf{x})$ and policies no longer holds:

Theorem 6.1.2

1. *Let ρ be any stationary randomized policy; then if:*
$$\phi_a^\rho(\mathbf{x}) = \sum_{t=0}^{\infty}\sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}) P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}')\alpha(\mathbf{x}'),\ \forall \mathbf{x}, a\ , \tag{6.6}$$
where $P_\rho(\mathbf{x}' \mid \mathbf{x}) = \sum_{a'} P(\mathbf{x}' \mid \mathbf{x}, a')\rho(a' \mid \mathbf{x})$, then ϕ_a^ρ is a feasible solution to the approximate dual LP in (6.2).
2. *There may exist a feasible solution $\phi_a(\mathbf{x})$ to the approximate dual LP in (6.2) such that, for some state $\mathbf{x}$, $\sum_a \phi_a(\mathbf{x}) = 0$.*
3. *There may even exist a set of $\phi_a(\mathbf{x})$ that is a feasible solution to the approximate dual LP in (6.2), and such that, for all states $\mathbf{x}$, $\sum_a \phi_a(\mathbf{x}) > 0$, but if we define a randomized policy ρ by:*
$$\rho(a \mid \mathbf{x}) = \frac{\phi_a(\mathbf{x})}{\sum_a \phi_a(\mathbf{x})}; \tag{6.7}$$
then the dual solution defined by $\phi_a^\rho(\mathbf{x})$ as in Equation (6.6) is such that $\phi_a^\rho(\mathbf{x}) \neq \phi_a(\mathbf{x})$ for at least some $\mathbf{x}$ and a.

Proof: *See Appendix A.4.2.* ▮

Comparing Theorem 6.1.2 for the approximate dual formulation with the corresponding Theorem 2.2.1 for the exact case, we formally prove two characteristics of the approximate dual LP:

Relaxation: Theorem 2.2.1 proves that every solution to the exact dual LP corresponds to a (randomized) policy. Theorem 6.1.2 Item 1 indicates that all (randomized) policies yield feasible solutions to the approximate dual LP in (6.2).

Non-policy solutions: The one to one correspondence between policies and dual solutions that is present in the exact formulation no longer holds. Specifically, Theorem 6.1.2 Items 2 and 3 prove that not all feasible solutions to the approximate dual LP in (6.2) necessarily correspond to policies.

Therefore, rather than approximating the space of policies, the approximate dual LP in (6.2) is finding the approximation to the state visitation frequencies $\phi_a(\mathbf{x})$ that has maximum value. To understand the nature of this approximation, examine again the constraint introduced by an arbitrary basis function $h_i(\mathbf{x})$:

$$\sum_{\mathbf{x},a} \phi_a(\mathbf{x}) h_i(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_i(\mathbf{x}) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_i(\mathbf{x}).$$

As $\phi_a(\mathbf{x})$ can be interpreted as a density, we can express this constraint using expectations:

$$\mathbf{E}_{\phi_a}\left[h_i(\mathbf{x})\right] = \mathbf{E}_{\alpha}\left[h_i(\mathbf{x})\right] + \gamma \mathbf{E}_{\phi_a P_a}\left[h_i(\mathbf{x})\right] \ . \qquad (6.8)$$

Thus, rather than enforcing the flow constraints described in Section 2.2.1 for all states, we are now enforcing flow constraints for features of the states (basis functions). That is, a set of visitation frequencies $\phi_a(\mathbf{x})$ in our approximate LP will be feasible if, for each feature or basis function h_i, the total expected value of this feature under $\phi_a(\mathbf{x})$, given by $\mathbf{E}_{\phi_a}\left[h_i(\mathbf{x})\right]$, is equal to the expected value of this feature under the starting distribution (represented by the state relevance weights $\alpha(\mathbf{x})$), $\mathbf{E}_{\alpha}\left[h_i(\mathbf{x})\right]$, plus the total discounted expected value of this feature under the flow from all other states $\mathbf{x}'$ to this state $\mathbf{x}$ times the respective visitation frequencies of the origin states, $\gamma \mathbf{E}_{\phi_a P_a}\left[h_i(\mathbf{x})\right]$. In other words, we are enforcing the flow constraints in terms of features of the states, rather than individually for each state.[2]

[2] As in the exact case, this relationship becomes more intuitive in the average reward case, where our relaxed constraints now become relaxed conditions on a stationary distribution.

6.1.2 Theoretical analysis of the LP-based approximation policies

Theorem 2.2.1 shows that there exists a one to one correspondence between every feasible solution to the exact dual LP in (6.1) and a (randomized) policy in the MDP. In Theorem 6.1.2, we proved that every policy corresponds to a feasible solution to the approximate dual formulation in (6.2), but that the one to one correspondence no longer holds. We will now define a correspondence between feasible solutions to the dual LP in (6.2) and policies. This correspondence leads to a new bound and intuition on the quality of the solutions obtained by the LP-based approximation approach, both in the dual form and in the primal form in (2.8).

Definition 6.1.3 (approximate dual solution policy set) *Let $\widetilde{\phi}_a$ be any feasible solution to the approximate dual LP in (6.2). We define the* approximate dual solution policy set, $\mathsf{PoliciesOf}[\widetilde{\phi}_a]$, *to include every (randomized) policy ρ such that:*

$$\rho(a \mid \mathbf{x}) = \begin{cases} \frac{\widetilde{\phi}_a(\mathbf{x})}{\sum_{a'} \widetilde{\phi}_{a'}(\mathbf{x})}, & \textit{if } \sum_{a'} \widetilde{\phi}_{a'}(\mathbf{x}) > 0; \\ \rho_a^{\mathbf{x}}, & \textit{otherwise} ; \end{cases}$$

where $\rho_a^{\mathbf{x}}$ is any *probability distribution over actions such that $\sum_{a'} \rho_{a'}^{\mathbf{x}} = 1$.* ∎

In other words, we define every feasible solution $\widetilde{\phi}_a$ to the dual LP to correspond to a set of randomized policies, where for states such that $\sum_{a'} \widetilde{\phi}_{a'}(\mathbf{x}) > 0$ we define the policy in the usual manner, and in states where $\sum_{a'} \widetilde{\phi}_{a'}(\mathbf{x}) = 0$ we can select any distribution over actions. Note that by Theorem 2.2.1, any feasible solution $\phi_a(\mathbf{x})$ to the exact dual LP in (6.1) has $\sum_a \phi_a(\mathbf{x}) > 0$ for all states. In this case, $\mathsf{PoliciesOf}[\phi_a]$ contain exactly one policy, as defined by the one to one correspondence in Theorem 2.2.1.

To understand the set of policies in $\mathsf{PoliciesOf}[\widetilde{\phi}_a]$, let us consider the greedy policy with respect to the solution of the primal LP-based approximation formulation in (2.8):

Lemma 6.1.4 *Let $\widehat{\mathbf{w}}$ be the weights of an optimal solution to the approximate primal LP in (2.8), then there exists an optimal solution $\widehat{\phi}_a$ to the approximate dual such that:*

$$\textit{Greedy}[\mathcal{V}^{\widehat{\mathbf{w}}}] \in \mathsf{PoliciesOf}[\widehat{\phi}_a] \,,$$

where $\mathcal{V}^{\widehat{\mathbf{w}}}(\mathbf{x}) = \sum_i \widehat{w}_i h_i(\mathbf{x})$ *is our approximate value function with weights* $\widehat{\mathbf{w}}$.
Proof: *See Appendix A.4.3.* ∎

This lemma proves that if $\widehat{\mathbf{w}}$ is an optimal solution to the LP-based approximation formulation in (2.8), then the greedy policy with respect to this value function is in the set of policies $\mathsf{PoliciesOf}[\widehat{\phi}_a]$ associated with some optimal dual solution $\widehat{\phi}_a$. We now prove a result bounding the quality of all policies in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$.

Note that if the optimal solution $\widehat{\phi}_a$ of the approximate dual LP is a feasible solution to the exact dual LP, then it is also guaranteed to be an exact optimal solution. Intuitively, if $\widehat{\phi}_a$ is almost feasible in the exact dual, then it should close to the optimal solution. Thus, we explicitly define a measure of violation, that indicates how close $\widehat{\phi}_a$ is from satisfying each flow constraint in the exact dual LP:

Definition 6.1.5 (dual violation) *Let* $\widetilde{\phi}_a$ *be any feasible solution to the approximate dual LP in (6.2). We define the* dual violation $\Delta[\widetilde{\phi}_a](\mathbf{x})$ *for state* $\mathbf{x}$ *by:*

$$\Delta[\widetilde{\phi}_a](\mathbf{x}) = \sum_a \widetilde{\phi}_a(\mathbf{x}) - \alpha(\mathbf{x}) - \gamma \sum_{\mathbf{x}',a'} \widetilde{\phi}_{a'}(\mathbf{x}') P(\mathbf{x} \mid \mathbf{x}', a') \ . \ \blacksquare$$

Our first result bounds the quality of the policies in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$ in terms of the dual violation $\Delta[\widehat{\phi}_a]$:

Theorem 6.1.6 *Let* $\widehat{\phi}_a$ *be an optimal solution to the approximate dual LP in (6.2), and let* $\widehat{\rho}$ *be any policy in* $\mathsf{PoliciesOf}[\widehat{\phi}_a]$*; then:*

$$\|\mathcal{V}^* - \mathcal{V}_{\widehat{\rho}}\|_{1,\alpha} \ \leq \ \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x})\, \mathcal{V}_{\widehat{\rho}}(\mathbf{x}), \tag{6.9}$$

where $\mathcal{V}_{\widehat{\rho}}$ *is the actual value function of the policy* $\widehat{\rho}$*; and the weighted* $\mathcal{L}_1$ *norm is defined by* $\|\mathcal{V}\|_{1,\alpha} = \sum_{\mathbf{x}} \alpha(\mathbf{x})\, |\mathcal{V}(\mathbf{x})|$.

Furthermore, if $\widehat{\mathbf{w}}$ *is an optimal solution to the primal LP associated with the dual solution* $\widehat{\phi}_a$*, then:*

$$\left\|\mathcal{V}^* - \mathcal{V}^{\widehat{\mathbf{w}}}\right\|_{1,\alpha} \ \leq \ \min_{\widehat{\rho} \in \mathsf{PoliciesOf}[\widehat{\phi}_a]} \left[\sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x})\, \mathcal{V}_{\widehat{\rho}}(\mathbf{x}) \right], \tag{6.10}$$

where $\mathcal{V}^{\widehat{\mathbf{w}}}$ is the approximate value function with weights $\widehat{\mathbf{w}}$.
Proof: *See Appendix A.4.4.* ▮

Recall that $\widehat{\phi}_a$ is not a feasible solution to the exact dual LP in (6.1). Our Theorem 6.1.6 bounds the quality of the approximations obtained by the LP-based algorithm in Section 2.3.2, and also the quality of all the policies in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$, by a term that measures the infeasibility of $\widehat{\phi}_a$. Our next result build on this theorem to bound the quality of our approximate value function and of our policies by the quality of the best achievable approximation in our basis function space. One of our results uses the notion of *Lyapunov function* defined by de Farias and Van Roy [2001a]. This function is used to weigh our approximation differently in different parts of the state space.

Theorem 6.1.7 *Let $\widehat{\phi}_a$ be an optimal solution to the approximate dual LP in (6.2). Let $\widehat{\rho}$ be any policy in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$, and $\mathcal{V}_{\widehat{\rho}}$ be the actual value of the policy $\widehat{\rho}$. Let the error $\varepsilon^{\infty}_{\widehat{\rho}}$ of the best max-norm approximation of $\mathcal{V}_{\widehat{\rho}}$ in the space of our basis functions be given by:*

$$\varepsilon^{\infty}_{\widehat{\rho}} = \min_{\mathbf{w}} \left\| \mathcal{V}_{\widehat{\rho}} - \mathbf{H}\mathbf{w} \right\|_{\infty}; \tag{6.11}$$

then:

$$\left\| \mathcal{V}^* - \mathcal{V}_{\widehat{\rho}} \right\|_{1,\alpha} \leq \frac{2\varepsilon^{\infty}_{\widehat{\rho}}}{1-\gamma}. \tag{6.12}$$

If $\widehat{\mathbf{w}}$ is an optimal solution to the primal LP associated with the dual solution $\widehat{\phi}_a$, then:

$$\left\| \mathcal{V}^* - \mathcal{V}^{\widehat{\mathbf{w}}} \right\|_{1,\alpha} \leq \min_{\widehat{\rho} \in \mathsf{PoliciesOf}[\widehat{\phi}_a]} \frac{2\varepsilon^{\infty}_{\widehat{\rho}}}{1-\gamma}, \tag{6.13}$$

where $\mathcal{V}^{\widehat{\mathbf{w}}}$ is our approximate value function with weights $\widehat{\mathbf{w}}$.

Furthermore, let $L(\mathbf{x}) = \sum_i w_i^L h_i(\mathbf{x})$ be any Lyapunov function *in the space of our basis functions, with contraction factor $\kappa \in (0,1)$ for the transition model $P_{\widehat{\rho}}$, that is, any strictly positive function such that:*

$$\kappa L(\mathbf{x}) \geq \gamma \sum_{\mathbf{x}'} P_{\widehat{\rho}}(\mathbf{x}' \mid \mathbf{x}) L(\mathbf{x}'). \tag{6.14}$$

Let the error $\varepsilon_{\widehat{\rho}}^{\infty,1/L}$ *of the best* $1/L$ *weighted max-norm approximation of* $\mathcal{V}_{\widehat{\rho}}$ *in the space of our basis functions be given by:*

$$\varepsilon_{\widehat{\rho}}^{\infty,1/L} = \min_{\mathbf{w}} \left\|\mathcal{V}_{\widehat{\rho}} - \mathbf{H}\mathbf{w}\right\|_{\infty,1/L}, \tag{6.15}$$

where $\|\mathcal{V}\|_{\infty,1/L} = \max_{\mathbf{x}} \frac{1}{L(\mathbf{x})} |\mathcal{V}(\mathbf{x})|$ *; then:*

$$\left\|\mathcal{V}^* - \mathcal{V}_{\widehat{\rho}}\right\|_{1,\alpha} \leq \frac{2\alpha^\intercal L}{1-\kappa}\varepsilon_{\widehat{\rho}}^{\infty,1/L}, \tag{6.16}$$

and

$$\left\|\mathcal{V}^* - \mathcal{V}^{\widehat{\mathbf{w}}}\right\|_{1,\alpha} \leq \min_{\widehat{\rho}\in\mathsf{PoliciesOf}[\widehat{\phi}_a]} \frac{2\alpha^\intercal L}{1-\kappa}\varepsilon_{\widehat{\rho}}^{\infty,1/L}. \tag{6.17}$$

Proof: *See Appendix A.4.5.* ▮

As the greedy policy with respect to the primal approximate solution is in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$, our bound of course also applies:

Corollary 6.1.8 *Let* $\widehat{\mathbf{w}}$ *be the weights of an optimal solution to the approximate primal LP in (2.8), and* $\mathcal{V}^{\widehat{\mathbf{w}}}(\mathbf{x}) = \sum_i \widehat{w}_i h_i(\mathbf{x})$ *be our approximate value function with weights* $\widehat{\mathbf{w}}$*. Let the greedy policy with respect to this value function be:*

$$\pi^{\widehat{\mathbf{w}}} = \mathit{Greedy}[\mathcal{V}^{\widehat{\mathbf{w}}}] \, ;$$

then:

$$\left\|\mathcal{V}^* - \mathcal{V}_{\pi^{\widehat{\mathbf{w}}}}\right\|_{1,\alpha} \leq \frac{2}{1-\gamma}\min_{\mathbf{w}} \left\|\mathcal{V}_{\pi^{\widehat{\mathbf{w}}}} - \mathbf{H}\mathbf{w}\right\|_{\infty}, \tag{6.18}$$

where $\mathcal{V}_{\pi^{\widehat{\mathbf{w}}}}$ *is the actual value of policy* $\pi^{\widehat{\mathbf{w}}}$*.*

Let L *be a Lyapunov function as defined in Theorem 6.1.7, then:*

$$\left\|\mathcal{V}^* - \mathcal{V}_{\pi^{\widehat{\mathbf{w}}}}\right\|_{1,\alpha} \leq \frac{2\alpha^\intercal L}{1-\kappa}\min_{\mathbf{w}} \left\|\mathcal{V}_{\pi^{\widehat{\mathbf{w}}}} - \mathbf{H}\mathbf{w}\right\|_{\infty,1/L}. \tag{6.19}$$

Proof: *This result is a corollary of Lemma 6.1.4 and Theorem 6.1.7.* ▮

In other words, the term $\|\mathcal{V}^* - \mathcal{V}_{\widehat{\rho}}\|_{1,\alpha}$ measures the quality of each policy in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$, in terms of how well the basis functions can approximate $\mathcal{V}_{\widehat{\rho}}$, the value function of *this* policy. In particular, we can bound the quality of the greedy policy associated with $\widehat{\mathbf{w}}$, the weights of an optimal solution to the approximate primal LP in (2.8), in terms of how well our basis functions can approximate $\mathcal{V}_{\pi^{\widehat{\mathbf{w}}}}$.

6.1.3 Relationship to existing theoretical analyzes

The results of de Farias and Van Roy [2001a] provide a foundation for the understanding of the quality of the solution obtained by the LP-based approximation algorithm. The theoretical bounds presented thus far provide further intuitions about this solution. To understand this relationship, we review some results by de Farias and Van Roy [2001a]. First, their Theorem 4.1 proves that:

$$\left\|\mathcal{V}^* - \mathcal{V}^{\widehat{\mathbf{w}}}\right\|_{1,\alpha} \;\leq\; \frac{2}{1-\gamma}\min_{\mathbf{w}}\|\mathcal{V}^* - \mathbf{H}\mathbf{w}\|_{\infty}\,. \tag{6.20}$$

Theorem 4.2 of de Farias and Van Roy [2001a] states that:

$$\left\|\mathcal{V}^* - \mathcal{V}^{\widehat{\mathbf{w}}}\right\|_{1,\alpha} \;\leq\; \frac{2\alpha^{\intercal}L}{1-\kappa}\min_{\mathbf{w}}\|\mathcal{V}^* - \mathbf{H}\mathbf{w}\|_{\infty,1/L}\,, \tag{6.21}$$

for a Lyapunov function L as defined in Theorem 6.1.7. Finally, Theorem 3.1 of de Farias and Van Roy [2001a] states that:

$$\|\mathcal{V}^* - \mathcal{V}_{\pi^{\widehat{\mathbf{w}}}}\|_{1,\alpha} \;\leq\; \frac{1}{1-\gamma}\left\|\mathcal{V}^* - \mathcal{V}^{\widehat{\mathbf{w}}}\right\|_{1,(1-\gamma)\phi^{\pi^{\widehat{\mathbf{w}}}}}\,, \tag{6.22}$$

where $\phi^{\pi^{\widehat{\mathbf{w}}}}$ are the visitation frequencies of the greedy policy $\pi^{\widehat{\mathbf{w}}}$. We now compare these results by de Farias and Van Roy [2001a], with our results in Theorem 6.1.7 and Corollary 6.1.8.

First note that the results of de Farias and Van Roy [2001a] in Equation (6.20) and Equation (6.21) bound the quality of the solution in terms of the best possible approximation of the optimal value function. Our results in Theorem 6.1.7, Equations (6.13) and (6.17), respectively, present related bounds in terms of how well the basis functions can approximate the "easiest" policy in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$. Our bounds are, in some sense, weaker, because we are bounding the quality of the solution as a function of the solution (the approximability of the policies obtained by the algorithm), while the results of de Farias and Van Roy [2001a] depend on how well the basis functions can approximate the optimal value function $\mathcal{V}^*$, which is independent of the algorithm. However, intuitively, it may be easier to approximate the "easiest" policy in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$. Thus, we view our bound as introducing the additional intuition that the LP-based approximation algorithm obtains good approximations both when the optimal value function can be well-approximated by the basis functions, and when the value function of the policies generated by the approach can be well approximated by the basis functions.

Additionally, de Farias and Van Roy [2001a] bound the quality of the greedy policy $\pi^{\widehat{\mathbf{w}}}$ by substituting the bound in Equation (6.20) or the one in Equation (6.21), into Equation (6.22). On the other hand, our Corollary 6.1.8 bounds the greedy policy directly, albeit as a function of the approximability of the solution.

Our results thus add some interesting properties to the results of de Farias and Van Roy [2001a]: first, as discussed above, when both approaches are combined, we obtain bounds that can depend on the approximability of the value function of $\pi^{\widehat{\mathbf{w}}}$ or on the approximability of $\mathcal{V}^*$; second, our bound on the quality of the greedy policy is a factor of $\frac{1}{1-\gamma}$ tighter than that of de Farias and Van Roy [2001a], though our result depends on the approximability of the value function of this greedy policy; finally, there is an incompatibility between the norms on the left and on the right hand side of the de Farias and Van Roy bound in Equation (6.22), this issue does not arise in our result.

6.2 Factored dual approximation algorithm

In the previous section, we presented an interpretation and theoretical analysis of the dual LP in (6.2). Unfortunately, the number of variables in this LP is exponential in the number state variables. Thus, as in the primal formulation, a direct solution for this linear program is infeasible. Fortunately, by considering the dual of the factored LP decomposition in Chapter 4 we obtain a very compact formulation for the approximate dual.

6.2.1 Factored objective function

First consider the objective function of the dual LP in (6.2), again given by: $\sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a)$. Recall that in factored MDPs, the reward function is decomposed as the sum of restricted-scope functions: $R^a(\mathbf{X}) = \sum_{j=1}^r R_j^a(\mathbf{W}_j^a)$. Using this representation, the objective function becomes:

$$\begin{aligned}\sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a) &= \sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) \sum_{j=1}^r R_j^a(\mathbf{x}[\mathbf{W}_j^a]) \; ; \\ &= \sum_{j=1}^r \sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R_j^a(\mathbf{x}[\mathbf{W}_j^a]) \; .\end{aligned}$$

Note that, as $\phi_a(\mathbf{x})$ forms a density, we can decompose this expectation in an analogous manner as the backprojection in Section 3.3. To understand this process, we define the *marginal visitation frequencies*:

Definition 6.2.1 (marginal visitation frequency, consistent flows) *For some subset (cluster) of variables* $\mathbf{B} \subseteq \mathbf{X}$, *let the* marginal visitation frequency $\mu_a(\mathbf{B})$ *be:*

$$\mu_a(\mathbf{b}) = \sum_{\mathbf{x} \sim [\mathbf{b}]} \phi_a(\mathbf{x}) \; , \quad \forall \mathbf{b} \in \mathrm{Dom}[\mathbf{B}] \; , \tag{6.23}$$

where $\mathbf{x} \sim [\mathbf{b}]$ *are the assignments of* $\mathbf{x}$ *that are consistent with* $\mathbf{b}$.

We can, furthermore, marginalize out the action variable, defining:

$$\mu(\mathbf{b}) = \sum_a \mu_a(\mathbf{b}) \; . \tag{6.24}$$

Finally, we say that a set of marginal visitation frequencies μ_a *and a set of global visitation frequencies* ϕ_a *are* consistent flows *iff* μ_a *and* ϕ_a *satisfy Equations (6.23) and (6.24).*

∎

Using this definition, we can rewrite the objective function of our dual LP as:

$$\sum_a \sum_{\mathbf{x}} \phi_a(\mathbf{x}) R(\mathbf{x}, a) = \sum_{j=1}^r \sum_a \sum_{\mathbf{w}_j^a \in \mathrm{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a) \; . \tag{6.25}$$

Thus, the representation of the objective function is now only exponential in the number of variables in each local reward function. In our SysAdmin example, the reward function has the form $R^a(\mathbf{X}) = \sum_j R_j(X_j)$. Thus, to represent the objective function for this problem, we only need marginal visitation frequencies over single variables $\mu(X_j)$. Therefore our objective becomes to maximize $\sum_j \mu(x_j) R_j(x_j)$.

6.2.2 Factored flow constraints

Consider the flow constraint in the dual LP in (6.2) induced by each basis function h_i, again given by:

$$\sum_{\mathbf{x},a} \phi_a(\mathbf{x}) h_i(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_i(\mathbf{x}) + \gamma \sum_{\mathbf{x}',a} \phi_a(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a) h_i(\mathbf{x}') \,. \tag{6.26}$$

Now recall that h_i has scope $\mathbf{C}_i$. Using the marginal visitation frequencies, as in the objective function, this constraint can be restated as an equivalent *factored flow constraint*:

$$\sum_{\mathbf{c} \in \mathrm{Dom}[\mathbf{C}_i]} \mu(\mathbf{c}) h_i(\mathbf{c}) = \sum_{\mathbf{c} \in \mathrm{Dom}[\mathbf{C}_i]} \alpha(\mathbf{c}) h_i(\mathbf{c}) + \gamma \sum_a \sum_{\mathbf{y} \in \mathrm{Dom}[\Gamma_a(\mathbf{C}'_i)]} \mu_a(\mathbf{y}) g_i^a(\mathbf{y}) \,, \tag{6.27}$$

where $\Gamma_a(\mathbf{C}'_i)$ is the backprojection (parents) of the variables $\mathbf{C}_i$ in the DBN as defined in Section 3.3; $g_i^a = \mathit{Backproj}_a(h_i)$ is the backprojection of h_i as defined in Figure 3.2; and $\alpha(\mathbf{c})$ are the marginal state relevance weights, as defined in Equation (5.2).

To understand the factored flow constraint, consider our SysAdmin example. Each basis function h_i is an indicator that takes value 1 if machine i is working. In this case, the marginal visitation frequency constraint becomes:

$$\mu(x_i) = \alpha(x_i) + \gamma \sum_a \sum_{x'_i, x'_{i-1} \in \mathrm{Dom}[X'_i, X'_{i-1}]} \mu_a(x'_i, x'_{i-1}) P(x_i \mid x'_i, x'_{i-1}, a) \,;$$

that is, the visitation frequency $\mu(x_i)$ for states where machine i is working is equal to the starting probability $\alpha(x_i)$ that machine i is working, plus the discounted visitation frequencies for the states of the parent machines in the network that lead to a state where machine i is working.

In order to express the objective function and the factored flow constraints for each basis function, our optimization problem must include variables to represent the marginal visitation frequencies $\mu(\mathbf{b})$ and $\mu_a(\mathbf{b}),\ \forall a$, for each assignment $\mathbf{b}$ of each cluster $\mathbf{B}$ used in the formulation. We call this set of clusters the *factored MDP cluster set*:

Definition 6.2.2 (factored MDP cluster set) *The* cluster set $\mathcal{B}_{\mathsf{FMDP}}$ *for a factored MDP is defined as:*

$$\mathcal{B}_{\mathsf{FMDP}} = \{\mathbf{W}_1^a, \dots, \mathbf{W}_r^a : \forall a\} \cup \{\mathbf{C}_1, \dots, \mathbf{C}_k\} \cup \{\Gamma_a(\mathbf{C}'_1), \dots, \Gamma_a(\mathbf{C}'_k) : \forall a\}\ ,$$

where $\{\mathbf{W}_1^a, \dots, \mathbf{W}_r^a\}$ *are the scopes of the local reward functions;* $\{\mathbf{C}_1, \dots, \mathbf{C}_k\}$ *are the scopes of our basis functions; and,* $\{\Gamma_a(\mathbf{C}'_1), \dots, \Gamma_a(\mathbf{C}'_k)\}$ *are the scopes of the backprojections of our basis functions, as defined in Section 3.3.* ▮

Note that the scope of the constant basis function h_0 is the empty set, thus the empty set $\{\emptyset\}$ is always included in $\mathcal{B}_{\mathsf{FMDP}}$.

6.2.3 Global consistency

If we maximize the factored objective function in Equation (6.25) with non-negative variables, under the factored flow constraints in Equation (6.27), and under the constraints in Equations (6.23) and (6.24) that enforce the definition of marginal visitation frequencies, we would clearly obtain the same solution as solving the dual LP in (6.2). Unfortunately, the constraints in the definition of marginal visitation frequencies in Equations (6.23) and (6.24) require us to keep the global visitation frequency variables $\phi_a(\mathbf{x})$ in the optimization problem, and, thus, the formulation remains exponentially large.

However, in some cases, we can remove the global visitation frequency variables $\phi_a(\mathbf{x})$ (and the constraints in Equations (6.23) and (6.24)) from the optimization problem, and still obtain the same solution as the one for the dual LP in (6.2).

This equivalency occurs when the set of marginal visitation frequencies are *global consistent*:

Definition 6.2.3 (global consistency) *As set of marginal visitation frequencies $\mu_a(\mathbf{b})$ over a set of clusters $\mathcal{B}$ is said to be* globally consistent *if there exists a set of non-negative global visitation frequencies $\phi_a(\mathbf{x})$, such that ϕ_a and μ_a are consistent flows.* ▮

Lemma 6.2.4 *Let an optimal solution to the factored objective function in Equation (6.25), under the factored flow constraints in Equation (6.27) for each basis function h_i, be given by the non-negative marginal visitation frequencies $\mu_a^*(\mathbf{b})$, for each assignment $\mathbf{b} \in \mathrm{Dom}[\mathbf{B}]$ of each cluster $\mathbf{B}$ in $\mathcal{B}_{\mathsf{FMDP}}$. If the $\mu_a^*(\mathbf{b})$ are globally consistent, let $\phi_a^*(\mathbf{x})$ be any set of global visitation frequencies such that ϕ_a^* and μ_a^* are consistent flows; then $\phi_a^*(\mathbf{x})$ is an optimal solution to the dual LP in (6.2).*

Proof: *See Appendix A.4.6.* ▮

6.2.4 Marginal consistency constraints

Even if we were given an optimal set of globally consistent marginal visitation frequencies μ_a^*, it is still infeasible to obtain the global visitation frequencies ϕ_a^*. Fortunately, we never need to compute ϕ_a^*. The coefficient of basis function h_i in our approximation is simply the Lagrange multiplier associated with the factored flow constraint induced by h_i. Therefore, we can restate our problem as one of finding a *globally consistent* set of non-negative marginal visitation frequencies $\mu_a^*(\mathbf{b})$ that maximizes the factored objective function in Equation (6.25), under the factored flow constraint in Equation (6.27) for each basis function h_i. Unfortunately, our factored flow constraints are not generally sufficient to ensure that the marginal visitation frequencies μ_a are globally consistent. We now show that we can guarantee global consistency by including additional constraints in our LP.

First, note that if a set of marginal visitation frequencies $\mu_a(\mathbf{b})$ is globally consistent, then, for any two clusters of variables $\mathbf{B}_i$ and $\mathbf{B}_j$, the respective marginals $\mu_a(\mathbf{B}_i)$ and $\mu_a(\mathbf{B}_j)$ assign the same frequency to the variables they share. We can add constraints to the LP to guarantee this consistency:

Definition 6.2.5 (marginal consistency constraints) *For two clusters of variables $\mathbf{B}_i$ and $\mathbf{B}_j$, the* marginal consistency constraints *for the marginal visitation frequencies $\mu_a(\mathbf{B}_i)$ and $\mu_a(\mathbf{B}_j)$ are given by:*

$$\sum_{\mathbf{b}_i \sim [\mathbf{y}]} \mu_a(\mathbf{b}_i) = \sum_{\mathbf{b}_j \sim [\mathbf{y}]} \mu_a(\mathbf{b}_j) \ , \ \ \forall \mathbf{y} \in \mathrm{Dom}[\mathbf{B}_i \cap \mathbf{B}_j] \ , \forall a \ . \tag{6.28}$$

For some set of variables $\mathbf{B}$*, the* action-marginalization consistency constraints *are given by:*

$$\mu(\mathbf{b}) = \sum_a \mu_a(\mathbf{b}) \ , \ \ \forall \mathbf{b} \in \mathrm{Dom}[\mathbf{B}] \ . \tag{6.29}$$

∎

For every pair of clusters, $\mathbf{B}_i$ and $\mathbf{B}_j$, in our factored MDP cluster set $\mathcal{B}_{\mathsf{FMDP}}$, we thus introduce a set of marginal consistency constraints. The number of such constraints is only exponential in the number of variables in $\mathbf{B}_i \cap \mathbf{B}_j$. This number of variables is no larger than the size of the larger cluster in $\mathcal{B}_{\mathsf{FMDP}}$. In some cases, these constraints are sufficient to guarantee global consistency, as the following example shows:

Example 6.2.6 (marginal consistency may imply global consistency) *Assume, for example, that our state space is defined via the state variables* A, B, C*, and that we have 2 clusters of variables:* $\mathbf{B}_1 = \{A, B\}$*,* $\mathbf{B}_2 = \{B, C\}$*. We are given two non-negative marginal visitation frequencies over these three clusters,*

$$\{\mu(A, B), \mu(B, C)\},$$

where $\sum_{a,b} \mu(a, b) = \frac{1}{1-\gamma}$ *and* $\sum_{b,c} \mu(b, c) = \frac{1}{1-\gamma}$*.*

Suppose that we add marginal consistency constraints for these marginal visitation frequencies:

$$\sum_a \mu(a, b) \ = \ \sum_c \mu(b, c) \ , \ \forall b \in \mathrm{Dom}[B] \ .$$

Now, let us define a set of global visitation frequencies $\phi(A, B, C)$ *by:*

$$\phi(a, b, c) = \frac{\mu(a, b)\mu(b, c)}{\sum_{c'} \mu(b, c')}, \ \forall a, b, c \in \mathrm{Dom}[A, B, C],$$

where we define $\frac{0}{0} = 0$*.*

We must prove that the global visitation frequencies ϕ and the set of marginals μ are consistent flows. Consider first $\mu(a,b)$:

$$\begin{aligned}\sum_c \phi(a,b,c) &= \sum_c \frac{\mu(a,b)\mu(b,c)}{\sum_{a'}\mu(a',b)};\\ &= \frac{\mu(a,b)\sum_c \mu(b,c)}{\sum_{c'}\mu(b,c')};\\ &= \mu(a,b).\end{aligned}$$

A similar derivation proves the consistency of ϕ and $\mu(b,c)$.

We must finally show that ϕ is a well-defined density function. Clearly $\phi(a,b,c)$ is non-negative, as it is composed of non-negative functions. We also need to show that $\phi(a,b,c)$ is normalized appropriately. We show above that $\sum_c \phi(a,b,c) = \mu(a,b)$. As $\sum_{a,b}\mu(a,b) = \frac{1}{1-\gamma}$, we have that ϕ is a well-defined density function, and the marginal consistency constraints are sufficient to guarantee global consistency of the set of marginals μ are consistent flows, in this example. ∎

Although the marginal consistency constraints ensure that the $\mu_a(\mathbf{B}_i)$ and $\mu_a(\mathbf{B}_j)$ agree on the visitation frequency of the variables they share, they do not always enforce global consistency, *i.e.*, the existence of a set of global visitation frequencies ϕ_a, such that ϕ_a and μ_a are consistent flows. The following example illustrates this problem:

Example 6.2.7 (marginal consistency does not always imply global consistency) *Assume, for example, that our state space is defined via the binary state variables A, B, C, and that we have 3 clusters of variables: $\mathbf{B}_1 = \{A,B\}$, $\mathbf{B}_2 = \{B,C\}$, and $\mathbf{B}_3 = \{C,A\}$. We are given three non-negative marginal visitation frequencies over these three clusters,*

$$\{\mu(A,B), \mu(B,C), \mu(C,A)\},$$

where each μ sums to $\frac{1}{1-\gamma}$, as in the previous example.

Now, suppose that we add marginal consistency constraints for these marginal visitation frequencies:

$$\begin{aligned}\sum_a \mu(a,b) &= \sum_c \mu(b,c)\ ,\ \forall b \in \mathrm{Dom}[B]\ ;\\ \sum_b \mu(b,c) &= \sum_a \mu(c,a)\ ,\ \forall c \in \mathrm{Dom}[C]\ ;\\ \sum_c \mu(c,a) &= \sum_b \mu(a,b)\ ,\ \forall a \in \mathrm{Dom}[A]\ .\end{aligned}$$

These constraints enforce local consistency between the marginal frequencies. However, local consistency does not, in general, imply global consistency, i.e., that there exists a set of global visitation frequencies $\phi(A,B,C)$ such that ϕ and the μ's are consistent flows.

Consider, for example, the following assignment to the marginal visitation frequencies:

$$\mu(A,B) = \begin{pmatrix} \frac{0.5}{1-\gamma} & 0 \\ 0 & \frac{0.5}{1-\gamma} \end{pmatrix},\quad \mu(B,C) = \begin{pmatrix} \frac{0.5}{1-\gamma} & 0 \\ 0 & \frac{0.5}{1-\gamma} \end{pmatrix},\quad \mu(C,A) = \begin{pmatrix} \frac{0.5-\varepsilon}{1-\gamma} & \frac{\varepsilon}{1-\gamma} \\ \frac{\varepsilon}{1-\gamma} & \frac{0.5-\varepsilon}{1-\gamma} \end{pmatrix},$$

for any $\varepsilon \in (0, 0.5]$; where, for a particular marginal $\mu(X,Y)$, the rows in the matrix correspond to values of X, and the columns correspond to values of Y. Clearly, these marginals satisfy the marginal consistency constraints.

Let us assume that there exists a set of global visitation frequencies $\phi(A,B,C)$, such that ϕ and the set of marginals μ are consistent flows, and:

$$\sum_{a,b,c} \phi(a,b,c) = \frac{1}{1-\gamma},\ \text{ and }\ \phi(a,b,c) \geq 0,\ \forall a,b,c\ .$$

If ϕ and μ are consistent flows, then:

$$\phi(a,b,\bar{c}) + \phi(\bar{a},b,\bar{c}) = \mu(b,\bar{c}) = 0.$$

Thus, by non-negativity of the global visitation frequencies, we have that:

$$\phi(a,b,\bar{c}) = \phi(\bar{a},b,\bar{c}) = 0. \tag{6.30}$$

Similarly, the consistent flows property implies that:

$$\phi(a,b,c) + \phi(a,b,\bar{c}) = \mu(a,b) = \frac{0.5}{1-\gamma}.$$

Substituting the term $\phi(a,b,\bar{c})$ from Equation (6.30), we obtain:

$$\phi(a,b,c) = \frac{0.5}{1-\gamma}. \tag{6.31}$$

Now consider $\mu(c,a)$:

$$\phi(a,b,c) + \phi(a,\bar{b},c) = \mu(c,a) = \frac{0.5-\varepsilon}{1-\gamma}.$$

Substituting the term $\phi(a,b,c)$ from Equation (6.31), we finally obtain:

$$\phi(a,\bar{b},c) = \frac{-\varepsilon}{1-\gamma},$$

violating the positivity requirement on the global visitation frequencies. Thus, even satisfying the marginal consistency constraints, the marginal visitation frequencies μ are not globally consistent. ▮

6.2.5 Global consistency constraints

In Example 6.2.6, the marginal consistency constraints were sufficient to guarantee global consistency. In Example 6.2.7, on the other hand, we show that the marginal consistency constraints do not always guarantee global consistency. Intuitively, this inconsistency is caused by the cyclic nature of the marginals in the second example. That is, $\mu(A,B)$ is consistent with $\mu(B,C)$, $\mu(B,C)$ with $\mu(C,A)$, and $\mu(C,A)$ with $\mu(A,B)$, but there is no constraint enforcing the joint consistency of the three terms. In general, if marginals can be arranged in a forest (collection of trees) graph, then global consistency is guaranteed, as we show in this section.

We can guarantee global consistency by using the notion of *decomposable models* [Lauritzen & Spiegelhalter, 1988], where local consistency does imply global consistency.

These models are characterized by cluster graphs that have a special, *cluster tree* (or forest), structure with a particular property:

Definition 6.2.8 (cluster tree, running intersection property) *A* cluster tree $\mathcal{F}(\mathcal{B})$ *is an undirected tree (or forest) with a node for each cluster* $\mathbf{B} \in \mathcal{B}$*. A cluster tree is said to satisfy the* running intersection property *if, whenever there is a variable* X *such that* $X \in \mathbf{B}_i \in \mathcal{B}$ *and* $X \in \mathbf{B}_j \in \mathcal{B}$*, there exists a (unique) path between* $\mathbf{B}_i$ *and* $\mathbf{B}_j$ *in the* $\mathcal{F}(\mathcal{B})$*, and* X *is also present in every cluster* $\mathbf{B}_k$ *in this path.* ∎

We can now define the requirements for global consistency of a set of clusters $\mathcal{B}$:

Definition 6.2.9 (junction tree) *A set of clusters* $\mathcal{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_n\}$ *is said to form a* junction tree *if there exists a cluster forest* $\mathcal{F}(\mathcal{B})$ *satisfying the running intersection property.* ∎

Clusters that form a junction tree are globally consistent:

Lemma 6.2.10 *Let* $\mathcal{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_n\}$ *be a cluster set forming a junction tree* $\mathcal{F}(\mathcal{B})$*. Let* $\mu(\mathbf{B})$ *and* $\mu_a(\mathbf{B})$ *be a set of non-negative marginal visitation frequencies over* $\mathcal{B}$*. If, for every pair of clusters* $\mathbf{B}_i$ *and* $\mathbf{B}_j$ *in* $\mathcal{B}$*, the marginal consistency constraints in Equations (6.28) and (6.29) hold, and for each* $\mathbf{B} \in \mathcal{B}$ *we have that* $\sum_{\mathbf{b} \in \mathrm{Dom}[\mathbf{B}]} \mu(\mathbf{b}) = K$*; then the marginals* $\mu(\mathbf{B})$ *and* $\mu_a(\mathbf{B})$ *are globally consistent. That is, there exists a density* $\phi_a(\mathbf{x})$ *such that for each* $\mathbf{B} \in \mathcal{B}$*:*

$$\mu_a(\mathbf{b}) = \sum_{\mathbf{x} \sim [\mathbf{b}]} \phi_a(\mathbf{x}) \ , \ \ \forall \mathbf{b} \in \mathrm{Dom}[\mathbf{B}], \ \forall a \ ,$$

with $\sum_{\mathbf{x},a} \phi_a(\mathbf{x}) = K$ *and* $\phi_a(\mathbf{x}) \geq 0$.
Proof: *see for example the book by Lauritzen and Spiegelhalter [1988].* ∎

Unfortunately, the factored MDP cluster set $\mathcal{B}_{\mathsf{FMDP}}$ as stated in Definition 6.2.2 does not necessarily form a junction tree as required by Theorem 6.2.10. However, using a simple *triangulation* procedure, we can obtain such a junction tree [Lauritzen & Spiegelhalter, 1988]. Let $\mathbf{Tr}(\mathcal{B})$ denote such a procedure that takes a cluster set $\mathcal{B}$ and returns a (larger) cluster set forming a junction tree. For simplicity of presentation, we assume that $\mathcal{B} \subseteq$

$\texttt{Tr}(\mathcal{B})$. Figure 6.1 illustrates a simple implementation of one such triangulation procedure, where a variable elimination order $\mathcal{O}$ is required.

We are now ready to present the complete *factored dual approximation* formulation:

Definition 6.2.11 (factored dual approximation) *The* factored dual approximation *LP formulation for any set of clusters $\mathcal{B} \supseteq \mathcal{B}_{\mathsf{FMDP}}$ is given by:*

Variables: $\forall \mathbf{B} \in \mathcal{B},\ \forall \mathbf{b} \in \mathrm{Dom}[\mathbf{B}],\ \forall a:$
$\mu(\mathbf{b})$ *and* $\mu_a(\mathbf{b})$;

Maximize: $\sum_{j=1}^{r} \sum_a \sum_{\mathbf{w}_j^a \in \mathrm{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a)$;

Subject to:

- $\forall i = 1, \ldots, k:$
  $$\sum_{\mathbf{c} \in \mathrm{Dom}[\mathbf{C}_i]} \mu(\mathbf{c}) h_i(\mathbf{c}) = \sum_{\mathbf{c} \in \mathrm{Dom}[\mathbf{C}_i]} \alpha(\mathbf{c}) h_i(\mathbf{c}) + \gamma \sum_a \sum_{\mathbf{y} \in \mathrm{Dom}[\Gamma_a(\mathbf{C}_i')]} \mu_a(\mathbf{y}) g_i^a(\mathbf{y}) ,$$
  where $\mathbf{C}_i = \mathit{Scope}[h_i]$;

- $\forall \mathbf{B}_i, \mathbf{B}_j \in \mathcal{B}, \quad \forall \mathbf{y} \in \mathrm{Dom}[\mathbf{B}_i \cap \mathbf{B}_j]\ , \forall a:$
  $$\sum_{\mathbf{b}_i \sim [\mathbf{y}]} \mu_a(\mathbf{b}_i) = \sum_{\mathbf{b}_j \sim [\mathbf{y}]} \mu_a(\mathbf{b}_j) \ ;$$

- $\forall \mathbf{B} \in \mathcal{B}, \quad \forall \mathbf{b} \in \mathrm{Dom}[\mathbf{B}],\ \forall a:$
  $\mu_a(\mathbf{b}) \geq 0$,
  $\mu(\mathbf{b}) = \sum_{a'} \mu_{a'}(\mathbf{b})$,
  $\sum_{\mathbf{b}' \in \mathrm{Dom}[\mathbf{B}]} \mu(\mathbf{b}') = \frac{1}{1-\gamma}$;

where the backprojection of basis function h_i, given by

$$g_i^a(\mathbf{y}) = \sum_{\mathbf{c}' \in \mathrm{Dom}[\mathbf{C}_i']} P(\mathbf{c}' \mid \mathbf{y}, a) h_i(\mathbf{c}'),$$

is defined in Section 3.3. ∎

The factored dual approximation formulation is guaranteed to be equivalent to the dual LP-based approximation formulation in (6.2):

```
TR($\mathcal{B}$,$\mathcal{O}$)
        // $\mathcal{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_m\}$ is a set of clusters.
        // $\mathcal{O}$ stores the elimination order.
        // Return a set of clusters $\mathcal{B}' \supseteq \mathcal{B}$ that forms a junction tree.
    // Initialize set of clusters.
    LET $\mathcal{B}' = \mathcal{B}$.
    FOR $i = 1$ TO NUMBER OF VARIABLES:
        // Select the next variable to be eliminated.
        LET $l = \mathcal{O}(i)$ ;
        // Select the clusters to be eliminated.
        LET $\mathbf{B}_1, \ldots, \mathbf{B}_L$ BE THE CLUSTERS IN $\mathcal{B}$ CONTAINING VARIABLES $X_l$.
        LET $\mathcal{B} = \mathcal{B} \setminus \{\mathbf{B}_1, \ldots, \mathbf{B}_L\}$.
        // Create a new union cluster.
        LET $\mathbf{B} = \bigcup_{i=1}^{L} \mathbf{B}_i$.
        // Add new cluster to the junction tree.
        LET $\mathcal{B}' = \mathcal{B}' \cup \{\mathbf{B}\}$.
        // Remove eliminated variable and store the new cluster.
        LET $\mathbf{B}' = \mathbf{B} \setminus X_l$.
        LET $\mathcal{B} = \mathcal{B} \cup \{\mathbf{B}'\}$.
    // We can now return a cluster set that forms a junction tree.
    RETURN $\mathcal{B}'$.
```

Figure 6.1: Triangulation procedure, returns a cluster set that forms a junction tree.

Theorem 6.2.12 *If the marginal visitation frequencies $\mu_a^*(\mathbf{b})$ are an optimal solution to the factored dual approximation formulation in Definition 6.2.11 using a set of clusters $\mathcal{B} \supseteq$ **Tr**$(\mathcal{B}_{FMDP})$; then there exists a set of global visitation frequencies $\phi_a^*(\mathbf{x})$ such that ϕ_a^* and the marginals μ_a^* are consistent flows, and $\phi_a^*(\mathbf{x})$ is an optimal solution to the dual LP in (6.2).*

Proof: *The existence of a set of global visitation frequencies $\phi_a^*(\mathbf{x})$ such that ϕ_a^* and the marginals μ_a^* are consistent flows is guaranteed by Lemma 6.2.10. The optimality of $\phi_a^*(\mathbf{x})$ is then guaranteed by Lemma 6.2.4.* ▮

To obtain a value function estimate from the formulation in Definition 6.2.11, we simply set the weight w_i of the ith basis function h_i to be the Lagrange multiplier associated with the ith factored flow constraint:

Corollary 6.2.13 *Let the marginal visitation frequencies $\mu_a^*(\mathbf{b})$, for each assignment $\mathbf{b} \in \mathrm{Dom}[\mathbf{B}]$ of each cluster $\mathbf{B}$ in $\mathcal{B} \supseteq$ **Tr**$(\mathcal{B}_{FMDP})$ be an optimal solution to the factored dual* *approximation formulation in Definition 6.2.11. Let w_i be the Lagrange multiplier associated with the factored flow constraint:*

$$\sum_{\mathbf{c}\in\mathrm{Dom}[\mathbf{C}_i]} \mu^*(\mathbf{c})h_i(\mathbf{c}) = \sum_{\mathbf{c}\in\mathrm{Dom}[\mathbf{C}_i]} \alpha(\mathbf{c})h_i(\mathbf{c}) + \gamma \sum_a \sum_{\mathbf{y}\in\mathrm{Dom}[\Gamma_a(\mathbf{C}'_i)]} \mu^*_a(\mathbf{y})g^a_i(\mathbf{y}) \; ;$$

then $\sum_i w_i h_i$ is an optimal solution to the primal formulation LP in (2.8).

Proof: *This result is a corollary of Theorem 6.2.12 and of standard complementarity results in duality theory (e.g., [Bertsimas & Tsitsiklis, 1997, Theorem 4.5]).* ▮

Therefore, by solving the compact dual LP in Definition 6.2.11, we obtain the same value function approximation as solving the exponentially-large dual LP in (6.2), in turn, yielding the same approximation as the linear programming-based approximation in (2.8).

6.2.6 Approximately factored dual approximation

As with the factored LP construction in Chapter 4, the largest cluster generated by the triangulation procedure is given by the induced width of an undirected graph defined over the variables $X_1, \ldots, X_n$, with an edge between X_l and X_m if they appear together in one of the original clusters $\mathcal{B}_{\mathsf{FMDP}}$. This induced width is exactly the size of the largest cluster in a junction tree that includes the clusters in $\mathcal{B}_{\mathsf{FMDP}}$. The number of marginal consistency constraints is exponential in this induced width. In some systems, the induced width may be too large to allow us to solve such a optimization problem. A more efficient alternative is to use an *approximate triangulation* procedure, relaxing the consistency constraints on the visitation frequencies:

Definition 6.2.14 (approximate triangulation) *An* approximate triangulation *procedure $\widehat{\mathbf{Tr}}(\mathcal{B})$ for cluster set $\mathcal{B}$ returns some cluster set $\mathcal{B}'$ such that $\mathcal{B} \subseteq \mathcal{B}'$.* ▮

Clearly, the approximate triangulation procedure $\widehat{\mathbf{Tr}}(\mathcal{B})$ need not return a cluster set that forms a junction tree, and it may even just return the original clusters $\mathcal{B}$. Using this procedure, we can solve an *approximately factored dual approximation* formulation by solving the LP in Definition 6.2.11 over the clusters in $\widehat{\mathbf{Tr}}(\mathcal{B}_{\mathsf{FMDP}})$. If $\widehat{\mathbf{Tr}}(\mathcal{B}_{\mathsf{FMDP}})$ does not increase the size of the clusters significantly, the size of this approximately factored formulation can be exponentially smaller than that of the globally consistent one obtained when using the exact triangulation procedure $\mathbf{Tr}(\mathcal{B}_{\mathsf{FMDP}})$.

By definition, for any approximate triangulation procedure $\widehat{\mathbf{Tr}}(\mathcal{B}_{\mathsf{FMDP}})$, our approximately factored dual LP contains a factored flow constraint for each basis function h_i, as in Equation (6.27). Thus, for any choice of $\widehat{\mathbf{Tr}}(\mathcal{B}_{\mathsf{FMDP}})$, we can obtain a factored value function, where the coefficient w_i for each h_i is simply the Lagrange multiplier of the factored flow constraint induced by h_i. This approximately factored formulation thus allows us to find a value function approximation very efficiently, even in many problems with large induced width.

Unfortunately, at this point, we cannot provide any theoretical guarantees for the quality of the value function obtained by this approximately triangulated formulation. However, this relaxed formulation does provide us with an "anytime" version of our factored LP decomposition technique: Note that, for two sets of clusters $\mathcal{B}$ and $\mathcal{B}'$ such that $\mathcal{B}_{\mathsf{FMDP}} \subseteq \mathcal{B}' \subset \mathcal{B}$, the set of constraints in the factored dual LP for $\mathcal{B}$ is exactly a super set of those in the dual LP for $\mathcal{B}'$, and both LPs have the same objective function. Our "anytime" algorithm thus starts by formulating and solving the factored dual LP over the clusters $\mathcal{B}_{\mathsf{FMDP}}$. We then choose a set of clusters $\mathcal{B}$, such that $\mathcal{B} \supset \mathcal{B}_{\mathsf{FMDP}}$. The dual LP formulation for $\mathcal{B}$ can be obtained simply by adding the extra constraints and variables corresponding to the clusters in $\mathcal{B} \setminus \mathcal{B}_{\mathsf{FMDP}}$. Interestingly, this procedure corresponds to using a delayed constraint generation procedure [Bertsimas & Tsitsiklis, 1997] to solve the dual LP formulation with the full triangulation $\mathbf{Tr}(\mathcal{B})$. This process can be repeated for increasing sets of constraints until either the full triangulation $\mathbf{Tr}(\mathcal{B}_{\mathsf{FMDP}})$ is obtained, or a preset running time limit is reached.

6.3 Discussion and related work

This chapter focused on the dual of the LP-based approximation algorithm. We first described an interpretation of this approach, showing that solutions to this approximate dual no longer have the one to one correspondence to policies that was present in the exact formulation. We then presented a new analysis of the quality of the policies obtained by the LP-based approximation algorithm. In this analysis, we defined a mapping between approximate dual solutions and policies. We then presented a new bound on the quality of all policies associated with the optimal solution of the approximate dual. These policies include the greedy policy with respect to the optimal solution to the primal LP-based approximation algorithm used thus far in this thesis.

Our theoretical results provide some complementary intuitions to those of de Farias and Van Roy [2001a]. We are able to obtain a potentially tighter bound on the quality of the greedy policy than de Farias and Van Roy [2001a], though our bound depends on the approximability of the value function of the greedy policy obtained by the algorithm, while de Farias and Van Roy [2001a] provide an *a priori* bound, in terms of the approximability of the optimal value function. We thus view our bound as providing the intuition that the LP-based approximation algorithm will yield good solutions when the value function of the resulting greedy policy can be well-approximated by the basis functions, in addition to when the optimal value function allows for such an approximation, which is the original result of de Farias and Van Roy [2001a].

Our interpretation of the approximate dual also leads to a new link between value function approximation and the representation of exponentially-large distributions in graphical models. This link is analogous to the one between value function approximation and maximization in a cost network presented in our factored primal LP decomposition technique. The complexity of the primal formulation is equivalent to that of the dual. Furthermore, the data structures used in the implementation of both formulations are very similar. Thus, there are no significant advantages to solving the dual LP with the exact triangulation $\mathbf{Tr}(\mathcal{B})$, over solving the primal formulation.

However, our dual formulation does yield approximate and "anytime" versions of our factored LP decomposition technique, as discussed in Section 6.2.6. Note that the simplest formulation of our approximately factored dual LP must contain at least the clusters in $\mathcal{B}_{\mathsf{FMDP}}$. Thus, this approximately factored dual formulation is particularly appropriate when each cluster in $\mathcal{B}_{\mathsf{FMDP}}$ only involves a small number of variables, but the cost network formed by these clusters has high induced width. For example, consider a set of variables $\{X_1, \ldots, X_n\}$. If $\mathcal{B}_{\mathsf{FMDP}}$ contains a cluster for every pair of variables $\{X_i, X_j\}$, then $\mathbf{Tr}(\mathcal{B}_{\mathsf{FMDP}})$ contains a cluster with all variables, and the representation of our factored LP would be exponential in the number of variables. Alternatively, we can formulate an approximately factored dual where $\widehat{\mathbf{Tr}}(\mathcal{B}_{\mathsf{FMDP}}) = \mathcal{B}_{\mathsf{FMDP}}$. This formulation would only be quadratic in the number of variables.

The use of such a locally consistent approximation is motivated by the success of approximate inference algorithms in graphical models. Exact inference in a graphical model requires the same triangulation procedure used to create a junction tree. Analogously, the complexity of such an inference procedure is exponential in the size of the largest cluster in this junction tree. Thus, inference in graphical models is generally infeasible for problems with large induced width. Recently, Yedidia *et al.* [2001] proposed a very successful approximate inference algorithm, which leads only to local consistency between clusters of variables, when the algorithm converges. The success of this procedure motivates the locally consistent relaxation of our factored dual approximation algorithm induced by $\widehat{\mathbf{Tr}}(\mathcal{B})$.

We believe that our approximately factored dual formulation will extend the applicability of our efficient solution techniques to highly-connected real-world systems.

Chapter 7

Exploiting context-specific structure

Thus far, we have presented a suite of algorithms which exploit additive structure in the reward and basis functions and sparse connectivity in the DBN representing the transition model. However, there exists another important type of structure that should also be exploited for efficient decision making: *context-specific independence* (CSI), a type of symmetry [Boutilier *et al.*, 1995]. For example, consider an agent responsible for building and maintaining a house, if the painting task can only be completed after the plumbing and the electrical wiring have been installed, then the probability that the painting is done is 0 in all contexts where plumbing or electricity are not done, *independently* of the agents action. The representation we have used so far in this thesis would use a table to represent this type of function. This table is exponentially large in the number of variables in the scope of the function, and ignores the context-specific structure inherent in the problem definition.

Boutilier *et al.* [Boutilier *et al.*, 1995; Dearden & Boutilier, 1997; Boutilier *et al.*, 1999; Boutilier *et al.*, 2000] have developed a set of algorithms which can exploit CSI in the transition and reward models to perform efficient (approximate) planning. Although this approach is often successful in problems where the *value function* contains sufficient context-specific structure, the approach is not able to exploit the additive structure which is also often present in real-world problems. (We discuss these algorithms further at the end of this chapter.)

In this chapter, we first review the extension of the factored MDP model to include context-specific structure. We then present an extension of our algorithms that can exploit both CSI and additive structure to obtain efficient approximations for factored MDPs.

7.1 Factored MDPs with context-specific and additive structure

There are several representations for context-specific functions. The most common are decision trees [Boutilier *et al.*, 1995], algebraic decision diagrams (ADDs) [Hoey *et al.*, 1999], and rules [Zhang & Poole, 1999]. We choose to use rules as our basic representation, for two main reasons. First, the rule-based representation allows a fairly simple algorithm for variable elimination, which is a key operation in our framework. Second, rules are not required to be mutually exclusive and exhaustive, a requirement that can be restrictive if we want to exploit additive independence, where functions can be represented as a linear combination of a set of non-mutually exclusive functions.

We begin by describing the rule-based representation (along the lines of Zhang and Poole's presentation [1999]) for the probabilistic transition model, in particular, the CPDs of our DBN model. Roughly speaking, each rule corresponds to some set of CPD entries that are all associated with a particular probability value. These entries with the same value are referred to a *consistent* contexts:

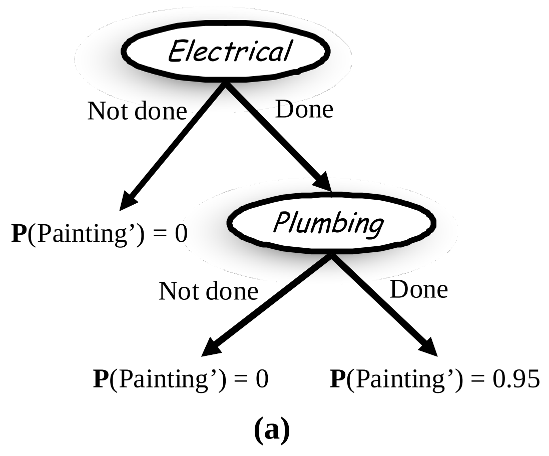
Definition 7.1.1 (consistent) *Let* $\mathbf{C} \subseteq \{\mathbf{X}, \mathbf{X}'\}$ *and* $\mathbf{c} \in \mathrm{Dom}(\mathbf{C})$*. We say that* $\mathbf{c}$ *is* consistent *with* $\mathbf{b} \in \mathrm{Dom}(\mathbf{B})$*, for* $\mathbf{B} \subseteq \{\mathbf{X}, \mathbf{X}'\}$*, if* $\mathbf{c}$ *and* $\mathbf{b}$ *have the same assignment for the variables in* $\mathbf{C} \cap \mathbf{B}$*.* ▮

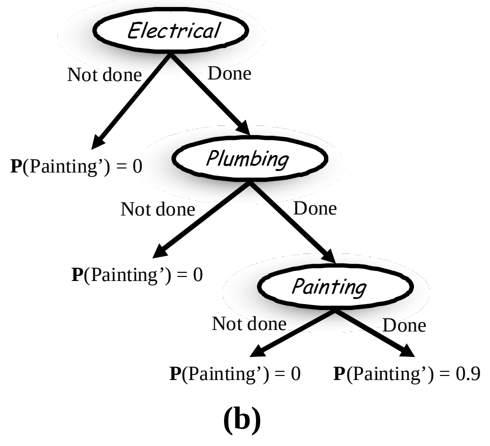
The probability of these consistent contexts will be represented by *probability rules*:

Definition 7.1.2 (probability rule, context) *A* probability rule $\eta = \langle \mathbf{c} : p \rangle$ *is a function* $\eta : \{\mathbf{X}, \mathbf{X}'\} \mapsto [0, 1]$*, where the* context $\mathbf{c} \in \mathrm{Dom}(\mathbf{C})$ *for* $\mathbf{C} \subseteq \{\mathbf{X}, \mathbf{X}'\}$ *and* $p \in [0, 1]$*, such that* $\eta(\mathbf{x}, \mathbf{x}') = p$ *if* $(\mathbf{x}, \mathbf{x}')$ *is consistent with* $\mathbf{c}$ *and is equal to* 1 *otherwise.* ▮

Definition 7.1.3 (rule CPD) *A (*rule CPD*)* P_a *is a function* $P_a : (\{X_i'\} \cup \mathbf{X}) \mapsto [0, 1]$*, composed of a set of probability rules*

$$\{\eta_1, \eta_2, \ldots, \eta_m\},$$





$$\eta_1 = \langle \neg\text{Electrical} \wedge \text{Painting'} : 0\rangle$$
$$\eta_2 = \langle \text{Electrical} \wedge \neg \text{ Plumbing} \wedge \text{Painting'} : 0\rangle$$
$$\eta_3 = \langle \text{Electrical} \wedge \text{Plumbing} \wedge \text{Painting'} : 0.95\rangle$$

(c)

$$\eta_4 = \langle \neg\text{Electrical} \wedge \text{Painting'} : 0\rangle$$
$$\eta_5 = \langle \text{Electrical} \wedge \neg\text{Plumbing} \wedge \text{Painting'} : 0\rangle$$
$$\eta_6 = \langle \text{Electrical} \wedge \text{Plumbing} \wedge \neg\text{Painting} \wedge \text{Painting'} : 0\rangle$$
$$\eta_7 = \langle \text{Electrical} \wedge \text{Plumbing} \wedge \text{Painting} \wedge \text{Painting'} : 0.9\rangle$$

(d)

Figure 7.1: Example CPDs for the true assignment of variable *Painting'* represented as decision trees: (a) when the action is paint; (b) when the action is not paint. The same CPDs can be represented by probability rules as shown in (c) and (d), respectively.

whose contexts are mutually exclusive and exhaustive. We define:

$$P_a(x'_i \mid \mathbf{x}) = \eta_j(\mathbf{x}, \mathbf{x}'),$$

where η_j *is the unique rule in* P_a *for which* $\mathbf{c}_j$ *is consistent with* $(x'_i, \mathbf{x})$. *We require that, for all* $\mathbf{x}$,

$$\sum_{x'_i} P_a(x'_i \mid \mathbf{x}) = 1. \quad \blacksquare$$

In this case, it is convenient to require that the rules be mutually exclusive and exhaustive, so that each CPD entry is uniquely defined by its association with a single rule. We can define $\mathsf{Parents}_a(X'_i)$ to be the union of the contexts of the rules in $P_a(X'_i \mid \mathbf{X})$. An example of a CPD represented by a set of probability rules is shown in Figure 7.1.

Rules can also be used to represent additive functions, such as reward or basis functions. We represent such context-specific value dependencies using *value rules*:

Definition 7.1.4 (value rule) *A* value rule $\rho = \langle \mathbf{c} : v \rangle$ *is a function* $\rho : \mathbf{X} \mapsto \mathbb{R}$ *such that* $\rho(\mathbf{x}) = v$ *when* $\mathbf{x}$ *is consistent with* $\mathbf{c}$ *and* 0 *otherwise.* ▮

Note that a value rule $\langle \mathbf{c} : v \rangle$ has a scope $\mathbf{C}$.

In general, our reward function R^a is represented as a *rule-based function*:

Definition 7.1.5 (rule-based function) *A* rule-based function $f : \mathbf{X} \mapsto \mathbb{R}$ *is composed of a set of rules* $\{\rho_1, \ldots, \rho_n\}$ *such that* $f(\mathbf{x}) = \sum_{i=1}^{n} \rho_i(\mathbf{x})$. ▮

In the same manner, each one of our basis functions h_j is now represented as a rule-based function.

Example 7.1.6 *In our construction example, we might have a set of rules:*

$$\begin{aligned} \rho_1 &= \langle \textit{Plumbing} = \text{done} : 100 \rangle; \\ \rho_2 &= \langle \textit{Electricity} = \text{done} : 100 \rangle; \\ \rho_3 &= \langle \textit{Painting} = \text{done} : 100 \rangle; \\ \rho_4 &= \langle \textit{Action} = \text{plumb} : -10 \rangle; \\ &\vdots \end{aligned}$$

which, when summed together, define the reward function $R = \rho_1 + \rho_2 + \rho_3 + \rho_4 + \cdots$. *At a sate where only the plumbing and electricity are done, and the action is to paint, the reward will be* 190. ▮

It is important to note that value rules are not required to be mutually exclusive and exhaustive. Each value rule represents a (weighted) indicator function, which takes on a value v in states consistent with some context **c**, and 0 in all other states. In any given state, the values of the zero or more rules consistent with that state are simply added together.

This notion of a rule-based function is related to the tree-structure functions used by Boutilier *et al.* [2000], but is substantially more general. In the tree-structure value functions, the rules corresponding to the different leaves are mutually exclusive and exhaustive. Thus, the total number of different values represented in the tree is equal to the number of leaves (or rules). In the rule-based function representation, the rules are not mutually exclusive, and their values are added to form the overall function value for different settings of the variables. Different rules are added in different settings, and, in fact, with k rules, one can easily generate 2^k different possible values, as is demonstrated in Section 7.7.2. Thus, the rule-based functions can provide a compact representation for a much richer class of value functions. Using this rule-based representation, we can exploit both CSI and additive independence in the representation of our factored MDP and basis functions.

7.2 Adding, multiplying and maximizing consistent rules

In our table-based algorithms, we relied on standard sum and product operators applied to tables. In order to exploit CSI using a rule-based representation, we must redefine these standard operations. In particular, the algorithms will need to add or multiply rules that ascribe values to overlapping sets of states.

We will start by defining these operations for rules with the same context:

Definition 7.2.1 (rule product, rule sum) *Let $\rho_1 = \langle \mathbf{c} : v_1 \rangle$ and $\rho_2 = \langle \mathbf{c} : v_2 \rangle$ be two rules with the same context* **c**. *Define the* rule product *as $\rho_1 \times \rho_2 = \langle \mathbf{c} : v_1 \cdot v_2 \rangle$; and the* rule sum *as $\rho_1 + \rho_2 = \langle \mathbf{c} : v_1 + v_2 \rangle$.* ▮

Note that this definition is restricted to rules with the same context. We will address this issue in a moment.

We also introduce an additional operation which maximizes a variable from a set of rules, which otherwise share a common context:

Definition 7.2.2 (rule maximization) *Let Y be a variable with $\mathrm{Dom}[Y] = \{y_1, \ldots, y_k\}$, and let ρ_i, for each $i = 1, \ldots, k$, be a rule of the form $\rho_i = \langle \mathbf{c} \wedge Y = y_i : v_i \rangle$. Then for the rule-based function $f = \rho_1 + \cdots + \rho_k$, define the* rule maximization *over Y as* $\max_Y f = \langle \mathbf{c} : \max_i v_i \rangle$. ▮

After this operation, Y has been maximized out from the scope of the function f.

These three operations we have just described can only be applied in to sets of rules that satisfy very stringent conditions. In order to make our set of rules amenable to the application of these operations, we might need to refine some of these rules. We therefore define the following operation:

Definition 7.2.3 (rule split) *Let $\rho = \langle \mathbf{c} : v \rangle$ be a rule, and Y be a variable. Define the* rule split *$\mathit{Split}(\rho \angle Y)$ of ρ on a variable Y as follows: If $Y \in \mathit{Scope}[\mathbf{C}]$, then $\mathit{Split}(\rho \angle Y) = \{\rho\}$; otherwise,*

$$\mathit{Split}(\rho \angle Y) = \{\langle \mathbf{c} \wedge Y = y_i : v \rangle \mid y_i \in \mathrm{Dom}[Y]\} \ .$$

▮

Thus, if we split a rule ρ on a variable Y that is not in the scope of the context of ρ, then we generate a new set of rules, with one for each assignment in the domain of Y.

In general, the purpose of rule splitting is to extend the context $\mathbf{c}$ of one rule ρ to coincide with the context $\mathbf{c}'$ of another consistent rule ρ'. Naively, we might take all variables in $\mathit{Scope}[\mathbf{C}'] - \mathit{Scope}[\mathbf{C}]$ and split ρ recursively on each one of them. However, this process creates unnecessarily many rules: If Y is a variable in $\mathit{Scope}[\mathbf{C}'] - \mathit{Scope}[\mathbf{C}]$ and we split ρ on Y, then only one of the $|\mathrm{Dom}[Y]|$ new rules generated will remain consistent with ρ': the one which has the same assignment for Y as the one in $\mathbf{c}'$. Thus, only this consistent rule needs to be split further. We can now define the recursive splitting procedure that achieves this more parsimonious representation:

Definition 7.2.4 (recursive rule split) *Let $\rho = \langle \mathbf{c} : v \rangle$ be a rule, and $\mathbf{b}$ be a context such that $\mathbf{b} \in \mathrm{Dom}[\mathbf{B}]$. Define the* recursive rule splitrule split!recursive *$\mathit{Split}(\rho \angle \mathbf{b})$ of ρ on a context $\mathbf{b}$ as follows:*

1. *$\{\rho\}$, if $\mathbf{c}$ is not consistent with $\mathbf{b}$; else,*
2. *$\{\rho\}$, if $\mathit{Scope}[\mathbf{B}] \subseteq \mathit{Scope}[\mathbf{C}]$; else,*
3. *$\{\mathit{Split}(\rho_i \angle \mathbf{b}) \mid \rho_i \in \mathit{Split}(\rho \angle Y)\}$, for some variable $Y \in \mathit{Scope}[\mathbf{B}] - \mathit{Scope}[\mathbf{C}]$.*

▮

In this definition, each variable $Y \in \mathit{Scope}[\mathbf{B}] - \mathit{Scope}[\mathbf{C}]$ leads to the generation of $k = |\mathrm{Dom}(Y)|$ rules at the step in which it is split. However, only one of these k rules is used in the next recursive step because only one is consistent with $\mathbf{b}$. Therefore, the size of the split set is simply $1 + \sum_{Y \in \mathit{Scope}[\mathbf{B}] - \mathit{Scope}[\mathbf{C}]} (|Dom(Y)| - 1)$. This size is independent of the order in which the variables are split within the operation.

Note that only one of the rules in $\mathsf{Split}(\rho\angle\mathbf{b})$ is consistent with $\mathbf{b}$: the one with context $\mathbf{c}\wedge\mathbf{b}$. Thus, if we want to add two consistent rules $\rho_1 = \langle \mathbf{c}_1 : v_1\rangle$ and $\rho_2 = \langle \mathbf{c}_2 : v_2\rangle$, then all we need to do is replace these rules by the set:

$$\mathsf{Split}(\rho_1\angle\mathbf{c}_2) \cup \mathsf{Split}(\rho_2\angle\mathbf{c}_1),$$

and then simply replace the resulting rules $\langle \mathbf{c}_1\wedge\mathbf{c}_2 : v_1\rangle$ and $\langle \mathbf{c}_2\wedge\mathbf{c}_1 : v_2\rangle$ by their sum $\langle \mathbf{c}_1\wedge\mathbf{c}_2 : v_1+v_2\rangle$. Multiplication is performed in an analogous manner.

Example 7.2.5 *Consider adding the following set of consistent rules:*

$$\begin{aligned}\rho_1 &= \langle a\wedge b : 5\rangle,\\ \rho_2 &= \langle a\wedge\neg c\wedge d : 3\rangle.\end{aligned}$$

In these rules, the context $\mathbf{c}_1$ of ρ_1 is $a\wedge b$, and the context $\mathbf{c}_2$ of ρ_2 is $a\wedge\neg c\wedge d$.

Rules ρ_1 and ρ_2 are consistent, therefore, we must split them to perform the addition operation:

$$\mathsf{Split}(\rho_1\angle\mathbf{c}_2) = \begin{cases}\langle a\wedge b\wedge c : 5\rangle,\\ \langle a\wedge b\wedge\neg c\wedge\neg d : 5\rangle,\\ \langle a\wedge b\wedge\neg c\wedge d : 5\rangle.\end{cases}$$

Likewise,

$$\mathsf{Split}(\rho_2\angle\mathbf{c}_1) = \begin{cases}\langle a\wedge\neg b\wedge\neg c\wedge d : 3\rangle,\\ \langle a\wedge b\wedge\neg c\wedge d : 3\rangle.\end{cases}$$

The result of adding rules ρ_1 and ρ_2 is

$$\begin{aligned}&\langle a\wedge b\wedge c : 5\rangle,\\ &\langle a\wedge b\wedge\neg c\wedge\neg d : 5\rangle,\\ &\langle a\wedge b\wedge\neg c\wedge d : 8\rangle,\\ &\langle a\wedge\neg b\wedge\neg c\wedge d : 3\rangle. \quad \blacksquare\end{aligned}$$

7.3 Rule-based one-step lookahead

As in Section 3.3 for the table-based case, the rule-based Q_a function can be represented as the sum of the reward function and the discounted expected value of the next state. Due to our linear approximation of the value function, the expectation term is, in turn, represented as the linear combination of the backprojections of our basis functions. To exploit CSI, we are representing the rewards and basis functions as rule-based functions. In order to represent Q_a as a rule-based function, it is sufficient for us to show how to represent the backprojection g_j of the basis function h_j as a rule-based function.

Each h_j is a rule-based function, which can be written as $h_j(\mathbf{x}) = \sum_i \rho_i^{(h_j)}(\mathbf{x})$, where $\rho_i^{(h_j)}$ has the form $\left\langle \mathbf{c}_i^{(h_j)} : v_i^{(h_j)} \right\rangle$. Each rule is a restricted-scope function; thus, we can simplify the backprojection as:

$$\begin{aligned}
g_j^a(\mathbf{x}) &= \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) h_j(\mathbf{x}')\;; \\
&= \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) \sum_i \rho_i^{(h_j)}(\mathbf{x}'); \\
&= \sum_i \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) \rho_i^{(h_j)}(\mathbf{x}'); \\
&= \sum_i \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) v_i^{(h_j)} \mathbb{1}(\mathbf{x}' = \mathbf{c}_i^{(h_j)}); \\
&= \sum_i v_i^{(h_j)} P_a(\mathbf{c}_i^{(h_j)} \mid \mathbf{x}).
\end{aligned}$$

The term $P_a(\mathbf{c}_i^{(h_j)} \mid \mathbf{x})$ is equivalent to:

$$P_a(\mathbf{c}_i^{(h_j)} \mid \mathbf{x}) = \prod_{l:\, X_l' \in\, \mathbf{C}_i'} P_a(\mathbf{c}_i^{(h_j)}[X_l'] \mid \mathbf{x}).$$

Each $P_a(\mathbf{c}_i^{(h_j)}[X_l'] \mid \mathbf{x})$ corresponds to a particular instantiation to the CPD of X_l, and is thus a rule-based function. $P_a(\mathbf{c}_i^{(h_j)} \mid \mathbf{x})$ is then the product of rule-based functions, and can thus be also written as a rule-based function. We denote this backprojection operation for a particular rule by $\textsc{RuleBackproj}_a(\rho_i^{(h_j)})$.

The backprojection procedure, described in Figure 7.2, follows three steps. First, the

```
RULEBACKPROJ$_a(\rho)$ ,  WHERE $\rho$ IS GIVEN BY $\langle \mathbf{c} : v \rangle$, WITH $\mathbf{c} \in \mathrm{Dom}[\mathbf{C}]$.
    LET $g = \{\}$.
    SELECT THE SET $\mathcal{P}$ OF RELEVANT PROBABILITY RULES:
     $\mathcal{P} = \{\eta_j \in P(X'_i \mid \mathrm{PARENTS}(X'_i)) \mid X'_i \in \mathbf{C}$ AND $\mathbf{c}$ IS CONSISTENT WITH $\mathbf{c}_j\}$.
    REMOVE THE $\mathbf{X}'$ ASSIGNMENMENTS FROM THE CONTEXT OF ALL RULES IN $\mathcal{P}$.
   // Multiply consistent rules:
    WHILE THERE ARE TWO CONSISTENT RULES $\eta_1 = \langle \mathbf{c}_1 : p_1 \rangle$ AND $\eta_2 = \langle \mathbf{c}_2 : p_2 \rangle$:
        IF $\mathbf{c}_1 = \mathbf{c}_2$, REPLACE THESE TWO RULES BY $\langle \mathbf{c}_1 : p_1 p_2 \rangle$;
        ELSE REPLACE THESE TWO RULES BY THE SET: $\mathrm{SPLIT}(\eta_1 \angle \mathbf{c}_2) \cup \mathrm{SPLIT}(\eta_2 \angle \mathbf{c}_1)$.
   // Generate value rules:
    FOR EACH RULE $\eta_i$ IN $\mathcal{P}$:
        UPDATE THE BACKPROJECTION $g = g \cup \{\langle \mathbf{c}_i : p_i v \rangle\}$.
    RETURN $g$.
```

Figure 7.2: Rule-based backprojection.

relevant rules are selected: In the CPDs for the variables that appear in the context of ρ, we select the rules consistent with this context, as these are the only rules that play a role in the backprojection computation. Second, we multiply all consistent probability rules to form a local set of mutually-exclusive rules. This procedure is analogous to the addition procedure described in Section 7.2. Now that we have represented the probabilities that can affect ρ by a mutually-exclusive set, we can simply represent the backprojection of ρ by the product of these probabilities with the value of ρ. That is, the backprojection of ρ is a rule-based function with one rule for each one of the mutually-exclusive probability rules η_i. The context of this new value rule is the same as that of η_i, and the value is the product of the probability of η_i and the value of ρ.

Example 7.3.1 *For example, consider the backprojection of a simple rule,*

$$\rho = \langle \textit{Painting} = \text{done} : 100 \rangle,$$

through the CPD in Figure 7.1(c) for the paint action:

$$\begin{aligned}
\textsc{RuleBackproj}_{\text{paint}}(\rho) &= \sum_{\mathbf{x}'} P_{\text{paint}}(\mathbf{x}' \mid \mathbf{x})\rho(\mathbf{x}'); \\
&= \sum_{\textit{Painting}'} P_{\text{paint}}(\textit{Painting}' \mid \mathbf{x})\rho(\textit{Painting}'); \\
&= 100 \prod_{i=1}^{3} \eta_i(\textit{Painting'} = done, \mathbf{x}) \; .
\end{aligned}$$

Note that the product of these simple rules is equivalent to the decision tree CPD shown in Figure 7.1(a). Hence, this product is equal to 0 *in most contexts, for example, when electricity is not done at time* t*. The product is non-zero only in one context: in the context associated with rule* η_3*. Thus, we can express the result of the backprojection operation by a rule-based function with a single rule:*

$$\textsc{RuleBackproj}_{\text{paint}}(\rho) = \langle \textit{Plumbing} \wedge \textit{Electrical} : 95 \rangle.$$

Similarly, the backprojection of ρ *when the action is not paint can also be represented by a single rule:*

$$\textsc{RuleBackproj}_{\neg\text{paint}}(\rho) = \langle \textit{Plumbing} \wedge \textit{Electrical} \wedge \textit{Painting} : 90 \rangle. \quad \blacksquare$$

Using this algorithm, we can now write the *backprojection* of the rule-based basis function h_j as:

$$g_j^a(\mathbf{x}) = \sum_i \textsc{RuleBackproj}_a(\rho_i^{(h_j)}), \tag{7.1}$$

where g_j^a is a sum of rule-based functions, and therefore also a rule-based function. For simplicity of notation, we use $g_j^a = \textsc{RuleBackproj}_a(h_j)$ to refer to this definition of backprojection. Using this notation, we can write $Q_a(\mathbf{x}) = R^a(\mathbf{x}) + \gamma \sum_j w_j g_j^a(\mathbf{x})$, which is again a rule-based function.

7.4 Rule-based maximization over the state space

The second key operation required to extend our planning algorithms to exploit CSI is to modify the variable elimination algorithm in Section 4.2 to handle the rule-based representation. In Section 4.2, we showed that the maximization of a linear combination of table-based functions with restricted scope can be performed efficiently using non-serial dynamic programming [Bertele & Brioschi, 1972], or variable elimination. To exploit structure in rules, we use an algorithm similar to variable elimination in a Bayesian network with context-specific independence [Zhang & Poole, 1999]. Specifically, we will substitute the generic operation ELIMOPERATOR in Figure 4.1 with a new rule-based maximization operation, which we denote by RULEMAXOUT.

Intuitively, the algorithm operates by selecting the value rules relevant to the variable being maximized in the current iteration. Then, a local maximization is performed over this subset of the rules, generating a new set of rules without the current variable. The procedure is then repeated recursively until all variables have been eliminated.

More precisely, our algorithm "eliminates" variables one by one, where the elimination process performs a maximization step over the variable's domain. Suppose that we are eliminating X_i, whose collected value rules lead to a rule function f, and f involves additional variables in some set $\mathbf{B}$, so that f's scope is $\mathbf{B} \cup \{X_i\}$. We need to compute the maximum value for X_i for each choice of $\mathbf{b} \in \mathrm{Dom}[\mathbf{B}]$. We use RULEMAXOUT$(f, X_i)$ to denote a procedure that takes a rule function $f(\mathbf{B}, X_i)$ and returns a rule function $g(\mathbf{B})$ such that: $g(\mathbf{b}) = \max_{x_i} f(\mathbf{b}, x_i)$. This is exactly a rule-based version of the table-based MAXOUT procedure in Figure 4.2. Such a rule-based maximization procedure is an adaptation of the variable elimination algorithm of [Zhang & Poole, 1999].

The rule-based variable elimination algorithm maintains a set $\mathcal{F}$ of value rules, initially containing the set of rules to be maximized. The algorithm then repeats the following steps for each variable X_i until all variables have been eliminated:

1. Collect all rules that depend on X_i into f_i — $f_i = \{\langle \mathbf{c} : v \rangle \in \mathcal{F} \mid X_i \in \mathbf{C}\}$ — and remove these rules from $\mathcal{F}$.
2. Perform the local maximization step over X_i: $g_i =$ RULEMAXOUT(f_i, X_i).
3. Add the rules in g_i to $\mathcal{F}$; now, X_i has been "eliminated".

```
RULEMAXOUT($f, B$)
    LET $g = \{\}$.
    ADD COMPLETING RULES TO $f$: $\langle B = b_i : 0\rangle, i = 1, \ldots, k$.
   // Summing consistent rules:
    WHILE THERE ARE TWO CONSISTENT RULES $\rho_1 = \langle \mathbf{c}_1 : v_1\rangle$ AND $\rho_2 = \langle \mathbf{c}_2 : v_2\rangle$:
        IF $\mathbf{c}_1 = \mathbf{c}_2$, THEN REPLACE THESE TWO RULES BY $\langle \mathbf{c}_1 : v_1 + v_2\rangle$;
        ELSE REPLACE THESE TWO RULES BY THE SET: SPLIT$(\rho_1 \angle \mathbf{c}_2) \cup$ SPLIT$(\rho_2 \angle \mathbf{c}_1)$.
   // Maximizing out variable $B$:
    REPEAT UNTIL $f$ IS EMPTY:
        IF THERE ARE RULES $\langle \mathbf{c} \wedge B = b_i : v_i\rangle, \forall b_i \in \mathrm{Dom}(B)$ :
            THEN REMOVE THESE RULES FROM $f$ AND ADD RULE $\langle \mathbf{c} : \max_i v_i\rangle$ TO $g$;
        ELSE SELECT TWO RULES: $\rho_i = \langle \mathbf{c}_i \wedge B = b_i : v_i\rangle$ AND $\rho_j = \langle \mathbf{c}_j \wedge B = b_j : v_j\rangle$ SUCH
         THAT $\mathbf{c}_i$ IS CONSISTENT WITH $\mathbf{c}_j$, BUT NOT IDENTICAL, AND REPLACE THEM WITH
         SPLIT$(\rho_i \angle \mathbf{c}_j) \cup$ SPLIT$(\rho_j \angle \mathbf{c}_i)$ .
    RETURN $g$.
```

Figure 7.3: Maximizing out variable B from rule function f.

In the remainder of this section, we present the algorithm for computing the local maximization RULEMAXOUT(f_i, X_i). The procedure, presented in Figure 7.3, is divided into two parts: first, all consistent rules are added together as described in Section 7.2; then, variable B is maximized. This maximization is performed by generating a set of rules, one for each assignment of B, whose contexts have the same assignment for all variables except for B, as in Definition 7.2.2. This set is then substituted by a single rule without a B assignment in its context and with value equal to the maximum of the values of the rules in the original set. Note that, to simplify the algorithm, we initially need to add a set of value rules with 0 value, which guarantee that our rule function f is complete (*i.e.*, there is at least one rule consistent with every context).

The correctness of this procedure follows directly from the correctness of the rule-based variable elimination procedure described by Zhang and Poole, merely by replacing summations with $\max$, and products with sums.

The cost of this algorithm is polynomial in the number of new rules generated in the maximization operation RULEMAXOUT(f_i, X_i). The number of rules is never larger and in many cases exponentially smaller than the complexity bounds on the table-based maximization in Section 4.2, which, in turn, was exponential only in the *induced width* of the cost network graph [Dechter, 1999]. However, the computational costs involved in managing sets of rules usually imply that the computational advantage of the rule-based approach over the table-based one will only be significant in problems that possess a fair amount of context-specific structure.

We conclude this section with a small example to illustrate the algorithm:

Example 7.4.1 *Suppose we are maximizing the variable A for the following set of rules:*

$$\begin{aligned}\rho_1 &= \langle \neg a : 1 \rangle,\\ \rho_2 &= \langle a \wedge \neg b : 2 \rangle,\\ \rho_3 &= \langle a \wedge b \wedge \neg c : 3 \rangle,\\ \rho_4 &= \langle \neg a \wedge b : 1 \rangle.\end{aligned}$$

When we add completing rules, we get:

$$\begin{aligned}\rho_5 &= \langle \neg a : 0 \rangle,\\ \rho_6 &= \langle a : 0 \rangle.\end{aligned}$$

In the first part of the algorithm, we need to add consistent rules: We add ρ_5 to ρ_1 (which remains unchanged), combine ρ_1 with ρ_4, ρ_6 with ρ_2, and then split ρ_6 on the context of ρ_3, to get the following inconsistent set of rules:

$$\begin{aligned}\rho_2 &= \langle a \wedge \neg b : 2 \rangle,\\ \rho_3 &= \langle a \wedge b \wedge \neg c : 3 \rangle,\\ \rho_7 &= \langle \neg a \wedge b : 2 \rangle, && \textit{(from adding } \rho_4 \textit{ to the consistent rule from } \mathsf{Split}(\rho_1 \angle \mathbf{b}))\\ \rho_8 &= \langle \neg a \wedge \neg b : 1 \rangle, && \textit{(from } \mathsf{Split}(\rho_1 \angle \mathbf{b}))\\ \rho_9 &= \langle a \wedge b \wedge c : 0 \rangle, && \textit{(from } \mathsf{Split}(\rho_6 \angle a \wedge b \wedge \neg c)).\end{aligned}$$

Note that several rules with value 0 are also generated, but not shown here because they are added to other rules with consistent contexts. We can move to the second stage (repeat loop) of RULEMAXOUT. *We remove ρ_2, and ρ_8, and maximize A out of them, to give:*

$$\rho_{10} = \langle \neg b : 2 \rangle.$$

We then select rules ρ_3 and ρ_7 and split ρ_7 on C (ρ_3 is split on the empty set and is not *changed),*

$$\rho_{11} = \langle \neg a \wedge b \wedge c : 2 \rangle,$$
$$\rho_{12} = \langle \neg a \wedge b \wedge \neg c : 2 \rangle.$$

Maximizing out A from rules ρ_{12} and ρ_3, we get:

$$\rho_{13} = \langle b \wedge \neg c : 3 \rangle.$$

We are left with ρ_{11}, which maximized with its counterpart ρ_9 gives the final result that does not depend on A:

$$\rho_{12} = \langle b \wedge \neg c : 2 \rangle.$$

Notice that, throughout this maximization, we have not split on the variable C when $\neg b \in \mathbf{c}_i$, giving us only 6 distinct rules in the final result. This is not possible in a table-based representation, since our functions would then be over the 3 variables A,B,C, and therefore must have 8 entries. ▮

7.5 Rule-based factored LP

In Section 4.3, we showed that the LPs used in our algorithms have exponentially many constraints of the form: $\phi \geq \sum_i w_i\, c_i(\mathbf{x}) - b(\mathbf{x}), \forall \mathbf{x}$, which can be substituted by a single, equivalent, non-linear constraint: $\phi \geq \max_{\mathbf{x}} \sum_i w_i\, c_i(\mathbf{x}) - b(\mathbf{x})$. We then showed that, using variable elimination, we can represent this non-linear constraint by an equivalent set of linear constraints in a construction we called the factored LP. The number of constraints in the factored LP is linear in the size of the largest table generated in the variable elimination procedure. This table-based algorithm can only exploit additive independence. We now extend the algorithm in Section 4.3 to exploit *both* additive and context-specific structure, by using the rule-based variable elimination described in the previous section.

Suppose we wish to enforce the more general constraint $0 \geq \max_{\mathbf{y}} F^{\mathbf{w}}(\mathbf{y})$, where $F^{\mathbf{w}}(\mathbf{y}) = \sum_j f_j^{\mathbf{w}}(\mathbf{y})$ such that each f_j is a rule. As in the table-based version, the superscript $\mathbf{w}$ means that f_j might depend on $\mathbf{w}$. Specifically, if f_j comes from basis function h_i, it is multiplied by the weight w_i; if f_j is a rule from the reward function, it is not.

In our rule-based factored linear program, we generate LP variables associated with contexts; we call these *LP rules*. An LP rule has the form $\langle \mathbf{c} : u \rangle$; it is associated with a context $\mathbf{c}$ and a variable u in the linear program. We begin by transforming all our original rules $f_j^{\mathbf{w}}$ into LP rules as follows: If rule f_j has the form $\langle \mathbf{c}_j : v_j \rangle$ and comes from basis function h_i, we introduce an LP rule $e_j = \langle \mathbf{c}_j : u_j \rangle$ and the equality constraint $u_j = w_i v_j$. If f_j has the same form but comes from a reward function, we introduce an LP rule of the same form, but the equality constraint becomes $u_j = v_j$.

Now, we have only LP rules and need to represent the constraint: $0 \geq \max_{\mathbf{y}} \sum_j e_j(\mathbf{y})$. To represent such a constraint, we follow an algorithm very similar to the variable elimination procedure in Section 7.4. The main difference occurs in the $\textsc{RuleMaxOut}(f, B)$ operation in Figure 7.3. Instead of generating new value rules, we generate new LP rules, with associated new variables and new constraints. The simplest case occurs when computing a split or adding two LP rules. For example, when we add two value rules in the original algorithm, we instead perform the following operation on their associated LP rules: If the LP rules are $\langle \mathbf{c} : u_i \rangle$ and $\langle \mathbf{c} : u_j \rangle$, we replace these by a new rule $\langle \mathbf{c} : u_k \rangle$, associated with a new LP variable u_k with context $\mathbf{c}$, whose value should be $u_i + u_j$. To enforce this value constraint, we simply add an additional constraint to the LP: $u_k = u_i + u_j$.

A similar procedure can be followed when a rule split is computed. If we are splitting $\langle \mathbf{c} : u_i \rangle$ on variable Y, we introduce new rules $\langle \mathbf{c} \wedge y_k : u_i \rangle$ for each assignment y_k to Y. All of these new rules refer to the same LP variable u_i, thus no new LP variables or constraints need to be introduced.

More interesting constraints are generated when we perform a maximization. In the rule-based variable elimination algorithm in Figure 7.3, this maximization occurs when we replace a set of rules:

$$\langle \mathbf{c} \wedge B = b_i : v_i \rangle, \forall b_i \in \mathrm{Dom}(B),$$

by a new rule

$$\left\langle \mathbf{c} : \max_i v_i \right\rangle.$$

Following the same process as in the LP rule summation above, if we are maximizing

$$e_i = \langle \mathbf{c} \wedge B = b_i : u_i \rangle, \forall b_i \in \mathrm{Dom}(B),$$

we generate a new LP variable u_k associated with the rule $e_k = \langle \mathbf{c} : u_k \rangle$. We cannot add the nonlinear constraint $u_k = \max_i u_i$, but we can add a set of equivalent linear constraints

$$u_k \geq u_i, \ \forall i.$$

Therefore, using these simple operations, we can exploit structure in the rule functions to represent the nonlinear constraint $e_n \geq \max_{\mathbf{y}} \sum_j e_j(\mathbf{y})$, where e_n is the very last LP rule we generate. A final constraint $u_n = \phi$ implies that we are representing exactly the constraints in Equation (4.2), without having to enumerate every state.

The correctness of our rule-based factored LP construction is a corollary of Theorem 4.3.2 and of the correctness of the rule-based variable elimination algorithm [Zhang & Poole, 1999].

Corollary 7.5.1 *The constraints generated by the* rule-based *factored LP construction are equivalent to the non-linear constraint in Equation (4.2). That is, an assignment to* $(\phi, \mathbf{w})$ *satisfies the rule-based factored LP constraints if and only if it satisfies the constraint in Equation (4.2).* ▮

The number of variables and constraints in the rule-based factored LP is linear in the number of rules generated by the variable elimination process. In turn, the number of rules is no larger, and often exponentially smaller, than the number of entries in the table-based approach.

To illustrate the generation of LP constraints as just described, we now present a small example:

Example 7.5.2 *Let* e_1*,* e_2*,* e_3*, and* e_4 *be the set of LP rules which depend on the variable* B *being maximized. Here, rule* e_i *is associated with the LP variable* u_i*:*

$$\begin{aligned} e_1 &= \langle a \wedge b : u_1 \rangle, \\ e_2 &= \langle a \wedge b \wedge c : u_2 \rangle, \\ e_3 &= \langle a \wedge \neg b : u_3 \rangle, \\ e_4 &= \langle a \wedge b \wedge \neg c : u_4 \rangle. \end{aligned}$$

In this set, note that rules e_1 *and* e_2 *are consistent. We combine them to generate the* *following rules:*

$$\begin{aligned} e_5 &= \langle a \wedge b \wedge c : u_5 \rangle, \\ e_6 &= \langle a \wedge b \wedge \neg c : u_1 \rangle. \end{aligned}$$

and the constraint $u_1 + u_2 = u_5$. *Similarly,* e_6 *and* e_4 *may be combined, resulting in:*

$$e_7 = \langle a \wedge b \wedge \neg c : u_6 \rangle.$$

with the constraint $u_6 = u_1 + u_4$. *Now, we have the following three inconsistent rules for the maximization:*

$$\begin{aligned} e_3 &= \langle a \wedge \neg b : u_3 \rangle, \\ e_5 &= \langle a \wedge b \wedge c : u_5 \rangle, \\ e_7 &= \langle a \wedge b \wedge \neg c : u_6 \rangle. \end{aligned}$$

Following the maximization procedure, since no pair of rules can be eliminated right away, we split e_3 *and* e_5 *to generate the following rules:*

$$\begin{aligned} e_8 &= \langle a \wedge \neg b \wedge c : u_3 \rangle, \\ e_9 &= \langle a \wedge \neg b \wedge \neg c : u_3 \rangle, \\ e_5 &= \langle a \wedge b \wedge c : u_5 \rangle. \end{aligned}$$

We can now maximize B *out from* e_8 *and* e_5*, resulting in the following rule and constraints respectively:*

$$\begin{aligned} & e_{10} = \langle a \wedge c : u_7 \rangle, \\ & u_7 \geq u_5, \\ & u_7 \geq u_3. \end{aligned}$$

Likewise, maximizing B *out from* e_9 *and* e_7*, we get:*

$$\begin{aligned} & e_{11} = \langle a \wedge \neg c : u_8 \rangle, \\ & u_8 \geq u_3, \\ & u_8 \geq u_6; \end{aligned}$$

which completes the elimination of variable B *in our rule-based factored LP.* ▮

7.6 Rule-based factored planning algorithms

We have presented an algorithm for exploiting both additive and context-specific structure in the LP construction steps of our planning algorithms. This rule-based factored LP approach can now be applied directly in our linear programming-based approximation and approximate policy iteration algorithms, which were presented in Sections 5.1 and 5.2, respectively.

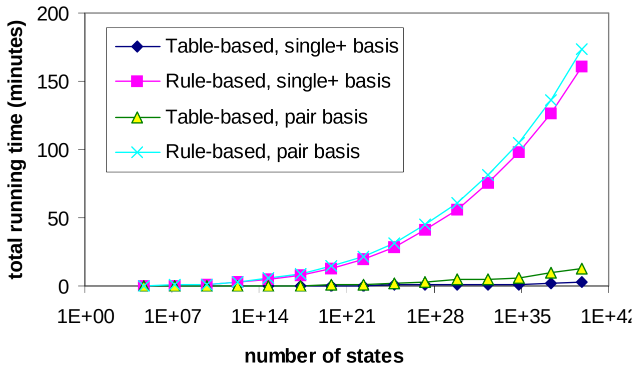
Our linear programming-based approximation in Section 5.1 requires no further modification. Approximate policy iteration does require an additional modification concerning the manipulation of the decision list policies presented in Section 5.2.2. Specifically, consider the conditional branches $\langle \mathbf{t}_i, a_i, \delta_i \rangle$ in the decision list policy. This condition is exactly a context-specific rule, where $\mathbf{t}_i$ is the context, and a_i and δ_i the "values" associated with this context. Thus, the policy representation algorithm in Section 5.2.2 can be applied directly with our new rule-based representation. The actual approximate policy iteration planning algorithm continues unchanged when we note that the indicators introduced into the constraints, as in Equation (5.8), are simply rules assigning a $-\infty$ value to the context $\mathbf{t}_j$. Therefore, we now have a complete framework for exploiting both additive and context-specific structure for efficient planning in factored MDPs.

7.7 Empirical evaluation

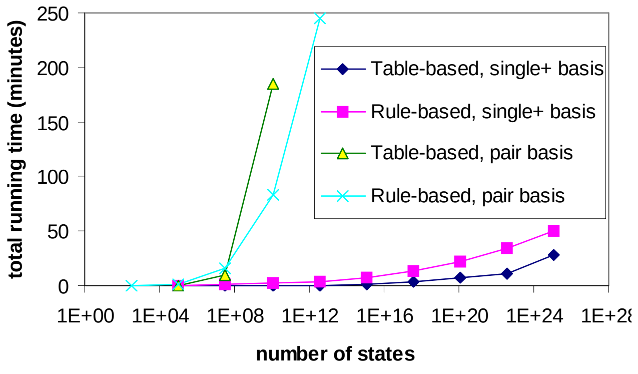
This section presents empirical evaluations of our rule-based planning algorithm. We report comparisons between the table-based and the rule-based implementations, and between our approach and the Apricodd algorithm of Hoey *et al.* [1999].

7.7.1 Comparing table-based and rule-based implementations

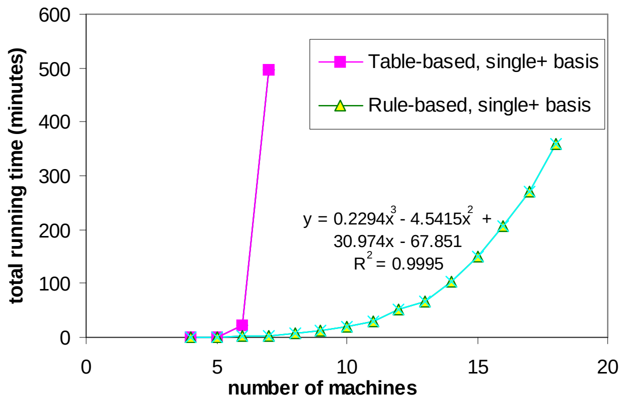
Our first evaluation compares a table-based representation, which exploits only additive independence, to the rule-based representation described in this chapter, which can exploit both additive and context-specific independence. For these experiments, we implemented our factored linear programming-based approximation algorithm with table-based and rule-based representations in C++, using CPLEX as the LP solver. Experiments were performed



(a)



(b)



(c)

Figure 7.4: Running time of rule-based and table-based algorithms in the Process-SysAdmin problem for various topologies: (a) "Star"; (b) "Ring"; (c) "Reverse star" (with fit function).

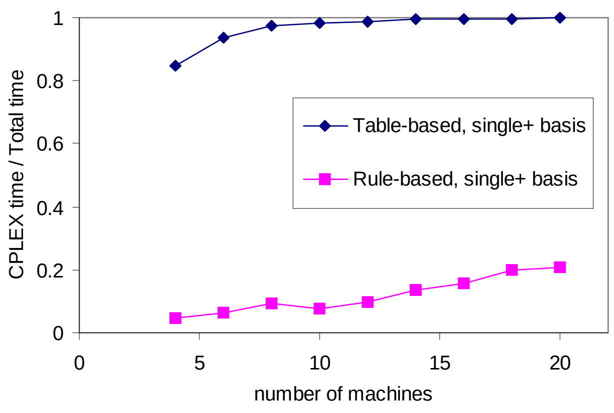


Figure 7.5: Fraction of total running time spent in CPLEX for table-based and rule-based algorithms in the Process-SysAdmin problem with a "Ring" topology.

on a Sun UltraSPARC-II, 400 MHz with 1GB of RAM.

To evaluate and compare the algorithms, we utilized a more complex extension of the SysAdmin problem. This problem, dubbed the Process-SysAdmin problem, contains three state variables for each machine i in the network: $Load_i$, $Status_i$ and $Selector_i$. Each computer runs processes and receives rewards when the processes terminate. These processes are represented by the $Load_i$ variable, which takes values in $\{Idle, Loaded, Success\}$, and the computer receives a reward when the assignment of $Load_i$ is *Success*. The $Status_i$ variable, representing the status of machine i, takes values in $\{$*Good, Faulty, Dead*$\}$; if its value is *Faulty*, then processes have a smaller probability of terminating and if its value is *Dead*, then any running process is lost and $Load_i$ becomes *Idle*. The status of machine i can become *Faulty* and eventually *Dead* at random; however, if machine i receives a packet from a dead machine, then the probability that $Status_i$ becomes *Faulty* and then *Dead* increases. The $Selector_i$ variable represents this communication by selecting one of the neighbors of i uniformly at random at every time step. The status of machine i in the next time step is then influenced by the status of this selected neighbor.

The SysAdmin can select at most one computer to reboot at every time step. If computer i is rebooted, then its status becomes *Good* with probability 1, but any running process is lost, *i.e.*, the $Load_i$ variable becomes *Idle*. Thus, in this problem, the SysAdmin must balance several conflicting goals: rebooting a machine kills processes, but not rebooting a machine may cause cascading faults in network. Furthermore, the SysAdmin can only choose one machine to reboot, which imposes the additional tradeoff of selecting only one of the (potentially many) faulty or dead machines in the network to reboot.

We experimented with two types of basis functions: "single+" includes indicators over all of the joint assignments of *Load*$_i$, *Status*$_i$ and *Selector*$_i$, and "pair" which, in addition, includes a set of indicators over *Status*$_i$, *Status*$_j$, and *Selector*$_i = j$, for each neighbor j of machine i in the network. The discount factor was $\gamma = 0.95$. The variable elimination order eliminated all of the *Load*$_i$ variables first, and then followed the same patterns as in the simple SysAdmin problem, eliminating first *Status*$_i$ and then *Selector*$_i$ when machine i is eliminated.

Figure 7.4 compares the running times for the table-based implementation to the ones for the rule-based representation for three topologies: "Star", "Ring" and "Reverse star". The "Reverse star" topology reverses the direction of the influences in the "Star": rather than the central machine influencing all machines in the topology, all machines influence the central one. These three topologies demonstrate three different levels of CSI: In the "Star" topology, the factors generated by variable elimination are small. Thus, although the running times are polynomial in the number of state variables for both methods, the table-based representation is significantly faster than the rule-based one, due to the overhead of managing the rules. The "Ring" topology illustrates an intermediate behavior: "single+" basis functions induce relatively small variable elimination factors, so the table-based approach is faster. However, with "pair" basis the factors are larger and the rule-based approach starts to demonstrate faster running times in larger problems. Finally, the "Reverse star" topology represents the worst-case scenario for the table-based approach. Here, the scope of the backprojection of a basis function for the central machine will involve all computers in the network, as all machines can potentially influence the central one in the next time step. Thus, the size of the factors in the table-based variable elimination approach is exponential in the number of machines in the network, which is illustrated by the exponential growth in Figure 7.4(c). The rule-based approach can exploit the CSI in this problem; for example, the status of the central machine *Status*$_0$ only depends on machine j, if *Selector*$_0 = j$. By exploiting CSI, we can solve the same problem in polynomial time

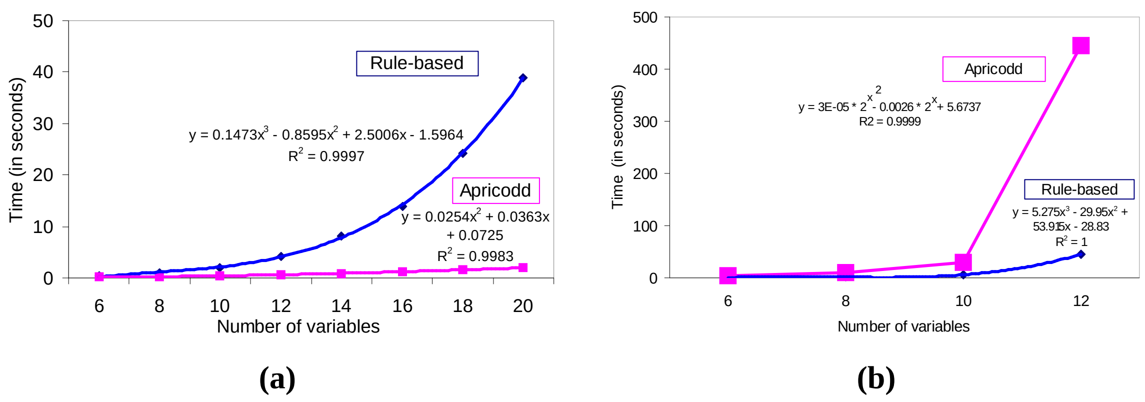


Figure 7.6: Comparing Apricodd [Hoey *et al.*, 2002] with our rule-based LP-based approximation algorithm on the (a) *Linear* and (b) *Expon* problems.

in the number of state variables, as seen in the second curve in Figure 7.4(c).

It is also instructive to compare the portion of the total running time spent in CPLEX for the table-based as compared to the rule-based approach. Figure 7.5 illustrates this comparison. Note that amount of time spent in CPLEX is significantly higher for the table-based approach. There are two reasons for this difference: first, due to CSI, the LPs generated by the rule-based approach are smaller than the table-based ones; second, rule-based variable elimination is more complex than the table-based one, due to the overhead introduced by rule management. Interestingly, the proportion of CPLEX time increases as the problem size increases, indicating that the asymptotic complexity of the LP solution is higher than that of variable elimination, thus suggesting that, for larger problems, additional large-scale LP optimization procedures, such as constraint generation, may be helpful.

7.7.2 Comparison to Apricodd

The most closely related work to ours is a line of research, which began with the work of Boutilier *et al.* [1995]. In particular, the approximate Apricodd algorithm of Hoey *et al.* [1999], which uses analytic decision diagrams (ADDs) to represent the value function is a strong alternative approach for solving factored MDPs. As we will discuss in detail in Section 7.8.1, the Apricodd algorithm can successfully exploit context-specific structure in the *value function*, by representing it with the set of mutually-exclusive and exhaustive branches of the ADD. On the other hand, our approach can exploit both additive and context-specific structure in the problem, by using a linear combination of non-mutually-exclusive rules. To better understand this difference, we evaluated both our rule-based linear programming-based approximation algorithm and Apricodd in two problems, *Linear* and *Expon*, designed by Boutilier *et al.* [2000] to illustrate respectively the best-case and the worst-case behavior of their algorithm. In these experiments, we used the web-distributed version of Apricodd [Hoey *et al.*, 2002], running it locally on a Linux Pentium III 700MHz with 1GB of RAM.

These two problems involve n binary variables $X_1, \ldots, X_n$ and n deterministic actions $a_1, \ldots, a_n$. The reward is 1 when all variables X_k are *true*, and is 0 otherwise. The problem is discounted by a factor $\gamma = 0.99$. The difference between the *Linear* and the *Expon* problems is in the transition probabilities. In the *Linear* problem, the action a_k sets the variable X_k to *true* and makes all *succeeding* variables, X_i for $i > k$, *false*. If the state space of the *Linear* problem is seen as a binary number, the optimal policy is to repeatedly set to true the largest bit (X_k variable) which has all preceding bits set to *true*. Using an ADD, the optimal value function for this problem can be represented in linear space, with $n+1$ leaves [Boutilier *et al.*, 2000]. This is the “best-case” for Apricodd, and the algorithm can compute this value function quite efficiently. Figure 7.6(a) compares the running time of Apricodd to the one of our algorithm with indicator basis functions between pairs of consecutive variables. Note that both algorithms obtain the same policy in polynomial time in the number of variables. However, in such a structured problem, the efficient implementation of the ADD package used in Apricodd makes it faster in this problem.

On the other hand, the *Expon* problem illustrates the worst case for Apricodd. In this problem, the action a_k sets the variable X_k to *true*, if all *preceding* variables, X_i for $i < k$, are *true*, and it makes all preceding variables *false*. If the state space is seen as a binary number, the optimal policy goes through all binary numbers in sequence, by repeatedly setting the largest bit (X_k variable) which has all preceding bits set to *true*. Due to discounting, the optimal value function assigns a value of γ^{2^n-j-1} to the jth binary number, so that the value function contains exponentially many different values. Using an ADD, the optimal value function for this problem requires an exponential number of leaves [Boutilier *et al.*, 2000], which is illustrated by the exponential running time in Figure 7.6(b). However, the same value function can be approximated very compactly as a factored linear value function using $n + 1$ basis functions: an indicator over each variable X_k and the constant base. As shown in Figure 7.6(b), using this representation, our factored linear programming-based approximation algorithm computes the value function in polynomial time. Furthermore, the policy obtained by our approach was optimal for this problem. Thus, in this problem, the ability to exploit additive independence allows an efficient polynomial time solution.

We also compared Apricodd to our rule-based linear programming-based approximation algorithm on the Process-SysAdmin problem. This problem has significant additive structure in the reward function and factorization in the transition model. Although this type of structure is not exploited directly by Apricodd, the ADD approximation steps performed by the algorithm can, in principle, allow Apricodd to find approximate solutions to the problem. We spent a significant amount of time attempting to find the best set of parameters for Apricodd for these problems.[1] We settled on the "sift" method of variable reordering and the "round" approximation method with the "size" (maximum ADD size) criterion. To allow the value function representation to scale with the problem size, we set the maximum ADD size to $4000 + 400n$ for a network with n machines. (We experimented with a variety of different growth rates for the maximum ADD size; here, as for the other parameters, we selected the choice that gave the best results for Apricodd.) We compared Apricodd with these parameters to our rule-based linear programming-based approximation algorithm with "single+" basis functions on a Pentium III 700MHz with 1GB of RAM. These results are summarized in Figure 7.7.

On very small problems (up to 4–5 machines), the performance of the two algorithms is fairly similar in terms of both the running time and the quality of the policies generated. However, as the problem size grows, the running time of Apricodd increases rapidly, and becomes significantly higher than that of our algorithm. Furthermore, as the problem size increases, the quality of the policies generated by Apricodd also deteriorates. This difference in policy quality is caused by the different value function representation used by the two algorithms. The ADDs used in Apricodd represent k different values with k leaves; thus, they are forced to agglomerate many different states and represent them using a single

[1]We are very grateful to Jesse Hoey and Robert St-Aubin for their assistance in selecting the parameters.

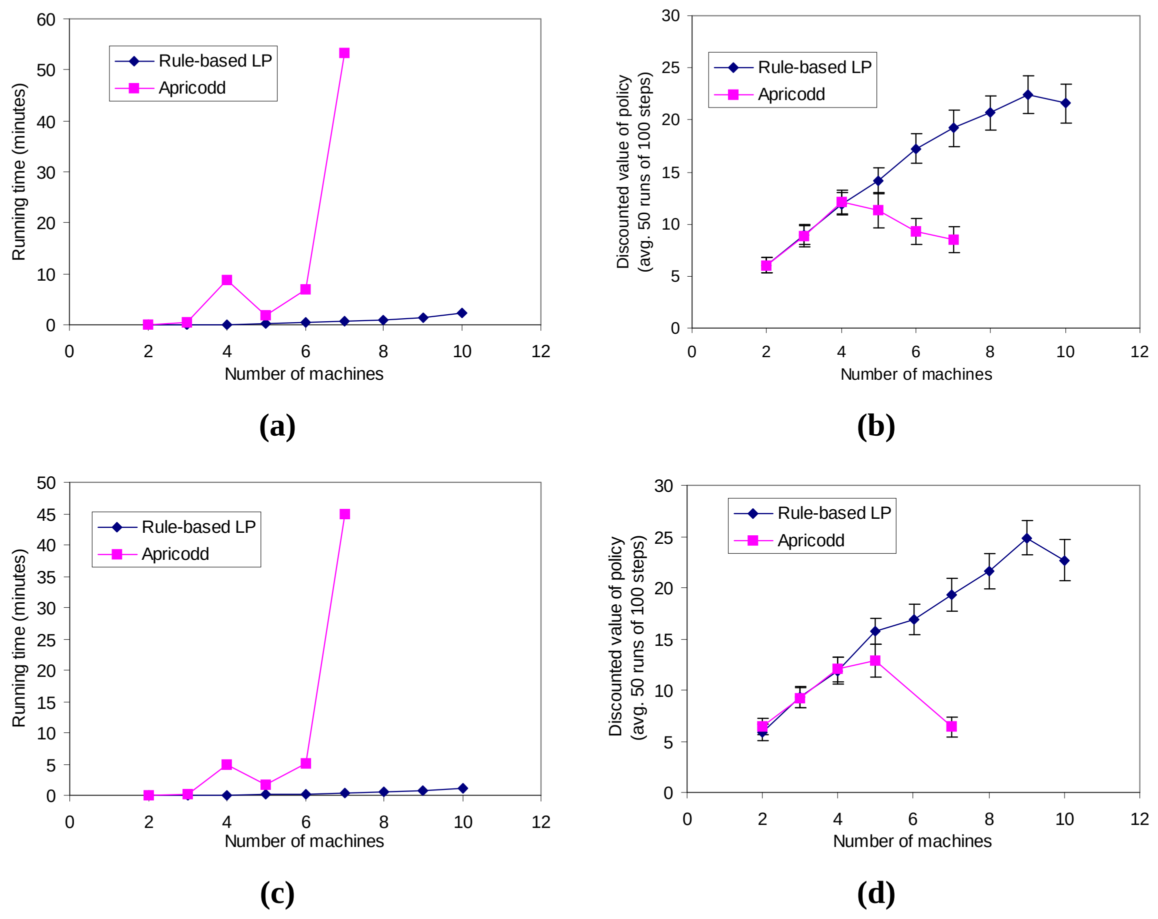


Figure 7.7: Comparing Apricodd [Hoey *et al.*, 2002] and rule-based LP-based approximation on the Process-SysAdmin problem with “Ring” topology, using “single+” basis functions: (a) running time and (b) value of the resulting policy; and with “Star” topology (c) running time and (d) value of the resulting policy.

value. For smaller problems, such agglomeration can still represent good policies. However, as the problem size increases and the state space grows exponentially, Apricodd's policy representation becomes inadequate, and the quality of the policies decreases. On the other hand, our linear value functions can represent exponentially many values with only k basis functions, which allows our approach to scale up to significantly larger problems.

7.8 Discussion and related work

Our factored LP decomposition technique, as discussed in Chapter 4, is able to exploit the additive structure in the factored value function. When combined with the planning algorithms in Chapter 5, we obtain efficient planning algorithms for factored MDPs. However, typical real-world systems possess both additive and context-specific structure. In order to increase the applicability of factored MDPs to more practical problems, in this chapter, we extended our factored LP decomposition technique to exploit both additive and context-specific structure in the factored model. Our table-based factored LP builds on the variable elimination algorithm of Bertele and Brioschi [1972]. In order to exploit CSI, our rule-based factored LP now builds on the rule-based variable elimination algorithm of Zhang and Poole [1999].

We demonstrate that exploiting CSI using a rule-based representation instead of the standard table-based one, can yield exponential improvements in computational time, when the problem has significant amounts of CSI. However, the overhead of managing sets of rules make it less well-suited for simpler problems.

7.8.1 Comparison to existing solution algorithms for factored MDPs

At this point, it is useful to compare our new factored planning algorithms, presented thus far in this thesis, with other solution methods for factored MDPs.

Tatman and Shachter [1990] considered the additive decomposition of value nodes in influence diagrams. This exact algorithm provides the first solution method for (finite horizon) factored MDPs. A number of approaches for factoring of general MDPs have been explored in the literature. Techniques for exploiting reward functions that decompose additively were studied by Meuleau *et al.* [1998], and by Singh and Cohn [1998].

The use of factored representations such as dynamic Bayesian networks was pioneered by Boutilier *et al.* [1995] and has developed steadily in recent years. These methods rely on the use of context-specific structures such as decision trees or analytic decision diagrams (ADDs) [Hoey *et al.*, 1999] to represent both the transition dynamics of the DBN and the value function. These algorithms use a form of dynamic programming on ADDs to partition the state space, representing the partition using a tree-like structure that branches on state variables and assigns values at the leaves. The tree is grown dynamically as part of the dynamic programming process and the algorithm creates new leaves as needed: A leaf is split by the application of a DP operator when two states associated with that leaf turn out to have different values in the backprojected value function. This process can also be interpreted as a form of model minimization [Dean & Givan, 1997].

The number of leaves in a tree used to represent a value function determines the computational complexity of the algorithm. It also limits the number of distinct values that can be assigned to states: since the leaves represent a partitioning of the state space, every state maps to exactly one leaf. However, as was recognized early on, there are trivial MDPs which require exponentially-large value functions. This observation led to a line of approximation algorithms aimed at limiting the tree size [Boutilier & Dearden, 1996] and, later, limiting the ADD size [St-Aubin *et al.*, 2001]. Kim and Dean [2001] also explored techniques for discovering tree-structured value functions for factored MDPs. While these methods permit good approximate solutions to some large MDPs, their complexity is still determined by the number of leaves in the representation and the number of distinct values than can be assigned to states is still limited as well.

Tadepalli and Ok [1996] were the first to apply linear value function approximation to Factored MDPs. Linear value function approximation is a potentially more expressive approximation method than trees or ADDs, because it can assign unique values to every state in an MDP without requiring storage space that is exponential in the number of state variables. The expressive power of a tree with k leaves can be captured by a linear function approximator with k basis functions such that basis function h_i is an indicator function that tests if a state belongs in the partition of leaf i. Thus, the set of value functions that can be represented by a tree with k leaves is a subset of the set of value functions that can be represented by a value function with k basis functions. Our experimental results in Section 7.7.2 highlight this difference by showing an example problem that requires exponentially many leaves in the value function, but that can be approximated well using a linear value function.

The main advantage of tree-based value functions is that their structure is determined dynamically during the solution of the MDP. In principle, as the value function representation is derived automatically from the model description, this approach requires less insight from the user. In problems for which the value function can be well approximated by a relatively small number of values, this approach provides an excellent solution to the problem. Our method of linear value function approximation aims to address what we believe to be the more common case, where a large range of distinct values is required to achieve a good approximation.

In this chapter, we empirically compare our approach to the work of Boutilier *et al.*. For problems with significant context-specific structure *in the value function*, their approach can be faster due to their efficient handling of the ADD representation. However, as discussed above, there are problems with significant context-specific structure in the problem representation, rather than in the value function, which require exponentially-large ADDs. In some such problem classes, we demonstrate that by using a linear value function our algorithm can obtain a polynomial-time near-optimal approximation of the true value function.

We note that Schuurmans and Patrascu [2001], based on our earlier work on max-norm projection using cost networks and our factored LP decomposition technique, independently developed an alternative approach for the (table-based) linear programming-based approximation presented in Section 5.1. Our method embeds a cost network inside a single linear program. By contrast, their method is based on a delayed constraint generation approach, as discussed in Section 4.5, using a cost network to detect constraint violations. When constraint violations are found, a new constraint is added, repeatedly generating and attempting to solve LPs until a feasible solution is found. As the approach of Schuurmans and Patrascu uses multiple calls to variable elimination in order to speed up the LP solution step, it will be most successful when the time spent solving the LP is significantly larger than the time required for variable elimination. As our experimental results in Section 7.7.1 suggest, the LP solution time is larger for the table-based approach. Thus, Schuurmans and Patrascu's constraint generation method will probably be more successful in table-based problems than in rule-based ones.

Finally, de Farias and Van Roy [2001b] propose a planning algorithm where the LP-based approximation formulation is solved by considering only a sampled subset of the exponential number of constraints. They prove that, by sampling a polynomial number of such constraints (under a particular distribution), they obtain a value function approximation close to that of considering all possible constraints. As de Farias and Van Roy discuss, their approach can be quite sensitive to the choice of sampling distribution. Our factored LP can efficiently decompose the exponentially-large constraint set in factored MDPs, in closed form. Thus, in structured models, our approach will probably provide better policies, as shown empirically in Section 9.3 for a particular domain, conversely the sampling method of de Farias and Van Roy [2001b] will apply to more general problems that cannot be represented compactly by factored MDPs. In Chapter 14, we suggest methods for combining the two approaches.

7.8.2 Limitations of the factored approach

In the previous section, we discuss some limitations of our algorithms, when compared to some other methods mostly for solving factored MDPs. Of course, there are other settings that are difficult to model with factored MDPs or to solve with our factored algorithms. For example, BlocksWorld is a domain where an agent must arrange blocks in a particular order on a table. This problem can be represented very compactly in deterministic settings using the STRIPS language [Fikes & Nilsson, 1971], or in stochastic settings using the probabilistic extension of this language [Kushmerick *et al.*, 1995]. Many algorithms have been proposed that can plan effectively in this setting, such as the methods of Kushmerick *et al.* [1995] and of Blum and Langford [1999]. Unfortunately, it is quite difficult to model this setting effectively using the propositional representation of in a factored MDP. In this case, the underlying model would have a very high connectivity, as each block can potentially affect every other block. Furthermore, the CSI representation described in this chapter would not decrease the representational complexity, as, again, we require a propositional representation that is not compact in BlocksWorld.

In Chapter 12, we present a relational representation for MDPs. such relational MDP can potentially represent the BlocksWorld domain compactly. However, we are faced with a second limitation of our approach: the reward function in BlocksWorld is specific to a particular arrangement of the blocks. Unfortunately, our factored approaches will often not be effective in such cases, where the reward function depends on a few very specific states. Although the resulting value function may have a compact description using rules that could probably be optimized effective by our method, it will often be difficult to select basis functions that cover this set of rules.

On the other hand, typical algorithms that successfully address problems such as Blocks-World tend to be goal-directed. That is, these algorithms can optimize for a single goal state, but not for a more complex reward function, such as a factored reward function. Furthermore, these methods are often not able to generate approximate solutions, as often required in large-scale environments. In general, our approach will be most successful in tasks were multiple goals must be balanced simultaneously, such as a maintenance task, rather than a single, very detailed goal. Thus, we can view approaches such as those of Kushmerick *et al.* [1995] and of Blum and Langford [1999] as complementary to our methods.

7.8.3 Summary

In the first part of this thesis, we presented the factored MDP representation, along with the basic tools required by our factored planning algorithms, including our novel LP decomposition technique. In this part of the thesis, we focused on developing a set of planning algorithms that build on these basic tools to find efficient approximate solutions to factored MDPs. This chapter, in particular, focused on extending our basic tools and algorithms to exploit context-specific structure, in addition to the additive structure addressed previously in this thesis. This novel algorithm can solve problems with very high induced width that could not be solved using the table-based representation. We believe that these efficient methods provide a strong framework for planning in large-scale real-world systems.

Part III

Multiagent coordination, planning and learning

Chapter 8

Collaborative multiagent factored MDPs

Consider a system where multiple agents, each with its own set of possible actions and its own observations, must coordinate in order to achieve a common goal. One obvious approach to this problem is to represent the system as an MDP, where the "action" now is a vector defining the joint action for all of the agents and the reward is the total reward received by all of these agents.

Thus far in this thesis, we have presented an efficient representation and algorithms for tackling very large, structured planning problems with exponentially-large states spaces. Our solution algorithms have assumed, though, that we are faced with single agent planning problems, where the action space A is relatively small. The factored linear programming-based approximation algorithm in Section 5.1, for example, requires us to apply our factored LP decomposition technique separately for each action $a \in A$. Unfortunately, as discussed in Chapter 1, the action space in multiagent planning problems is exponential in the number of agents, thus rendering impractical any approach that enumerates possible action choices explicitly.

In this part of the thesis, we present a representation and algorithms that will allow us to tackle the exponentially-large action spaces that arise in multiagent systems.

8.1 Representation

In our collaborative multiagent setting, we have a collection of agents $\mathbf{A} = \{A_1, \ldots, A_g\}$, where each agent A_j must choose an action a_j from a finite set of possible actions $\mathrm{Dom}[A_j]$. These agents are again acting in a space described by a set of discrete state variables, $\mathbf{X} = \{X_1 \ldots X_n\}$, as in the single agent case.

Consider a multiagent version of our system administrator problem:

Example 8.1.1 *Consider the problem of optimizing the behavior of many system administrators (multiagent SysAdmin) who must coordinate to maintain a network of computers. In this problem, we have m administrators (agents), where agent A_i is responsible for maintaining the ith computer in the network. As in Example 2.1.1, each machine in this network is connected to some subset of the other machines.*

We base this more elaborate multiagent example on the Process-SysAdmin problem in Section 7.7.1, without introducing the selector variables. Each machine is now associated with only two ternary random variables: Status $S_i \in \{$good, faulty, dead$\}$, and Load $L_i \in \{$idle, loaded, process successful$\}$. In this multiagent formulation, each agent A_i must decide whether machine i should be rebooted, in which case the status of this machine becomes good and any running process is lost. On the other hand, if the agent does not reboot a faulty machine, it may die and cause cascading faults in the network. Our goal here is to coordinate the actions of the administrators in order to maximize the total number of processes that terminate successfully in this network. ∎

This example illustrates some of the issues that arise in a collaborative multiagent problem: although each agent receives a local reward (when its process terminates), its actions can affect the long-term rewards of the entire system. As we are interested in maximizing these global rewards, rather than optimizing locally and greedily for each agent, we must design a model that will represent these long-term global interactions, and yield a global coordination strategy which maximizes the total reward.

In our *collaborative multiagent MDP* formulation, a state $\mathbf{x}$ is a state for the whole system and an action $\mathbf{a}$ is a joint action for all agents, as defined above. The transition model $P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a})$ now represents the probability that the entire system will transition from a joint state $\mathbf{x}$ to a joint state $\mathbf{x}'$ after the agents jointly take the action $\mathbf{a}$. Similarly, our reward function $R(\mathbf{x}, \mathbf{a})$ will now depend both on the joint state of the system and on the joint action of all agents. A factored MDP allows us to represent transition models with the exponentially many states represented by our state variables $\mathbf{X}$. Unfortunately, as defined in Chapter 3, our representation requires us to define a DBN for each joint action $\mathbf{a}$. The number of such DBNs would thus be exponential in the number of agents. In this chapter, we extend our factored MDP representation and basic framework to allow us to model multiagent problems.

8.1.1 Multiagent factored transition model

In the multiagent case, we describe the dynamics of the system using a *dynamic decision network (DDN)* [Dean & Kanazawa, 1989]. A DDN is a simple extension of a DBN, whose nodes are both the state variables $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$ and the agents' (action) variables $\{A_1, \ldots, A_g\}$. For simplicity of exposition, we again assume that $\text{Parents}(X'_i) \subseteq \{\mathbf{X}, \mathbf{A}\}$; this assumption is relaxed in Section 8.2. Each node X'_i is again associated with a CPD $P(X'_i \mid \text{Parents}(X'_i))$. In the single agent case, we had a set of CPDs for each action a, now we have one graph for the entire system, and the parents of X'_i are a subset of both state and agent variables. The global transition probability distribution is then defined to be:

$$P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) = \prod_i P(x'_i \mid \mathbf{x}[\text{Parents}(X'_i)], \mathbf{a}).$$

Figure 8.1(a), illustrates the part of the DDN corresponding to the ith machine in a multiagent SysAdmin network, where state variables are represented by circles, agent variables by squares and reward variables by diamonds in the usual influence diagram notation [Howard & Matheson, 1984]. The parents of the load variable L'_i for the ith machine are $\text{Parents}(L'_i) = \{L_i, S_i, A_i\}$, the load in the previous time step, the status of the ith machine and the action of the ith agent. Similarly, the parents of the status variable S'_i are $\text{Parents}(S'_i) = \{S_i, A_i\} \cup \{S_j \mid j \text{ is connected to } i \text{ in the computer network }\}$, the status of the ith machine in the previous time step, the action of the ith agent, and the status S_j of all machines j connected to i in the computer network. For the ring network topology in Figure 8.1(b), we obtain the complete DDN in Figure 8.1(c).

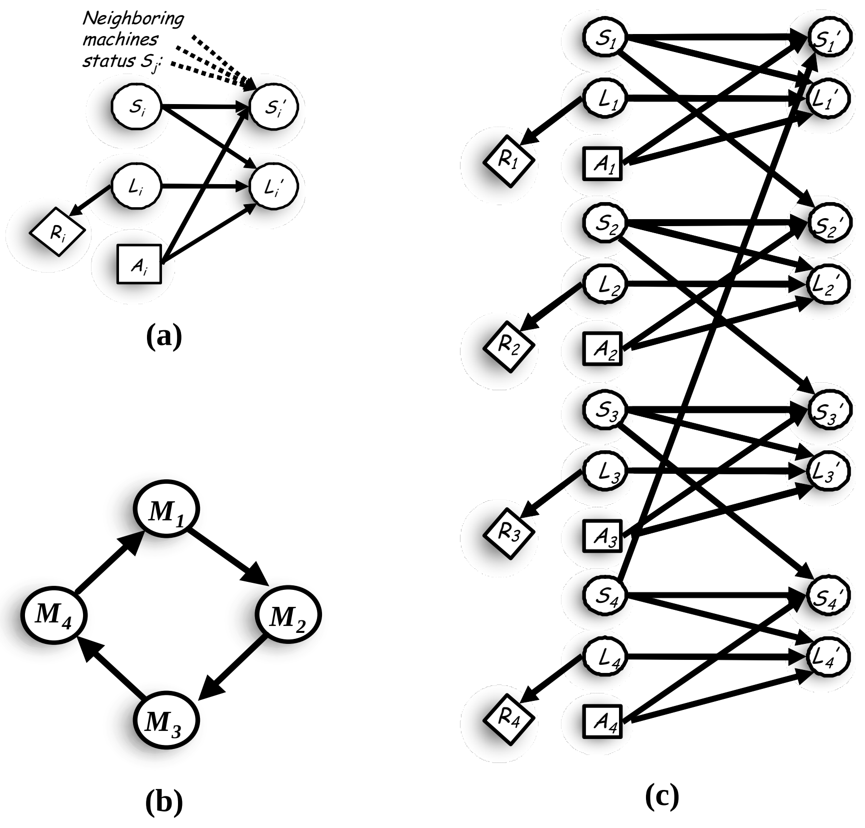

Neighboring machines status S_j:
S_i
S_i'
L_i
L_i'
R_i
A_i
(a)
M_1
M_2
M_3
M_4
(b)
S_1 S_1' L_1 L_1' R_1 A_1
S_2 S_2' L_2 L_2' R_2 A_2
S_3 S_3' L_3 L_3' R_3 A_3
S_4 S_4' L_4 L_4' R_4 A_4
(c)

Figure 8.1: Multiagent factored MDP example: (a) local DDN component for each computer in a network; (b) ring of 4 computers; (c) global DDN ring of 4 computers.

8.1.2 Multiagent factored rewards

As discussed in Chapter 1, in a collaborative multiagent setting, every agent has the same reward function, and agents are trying to maximize the long-term joint reward achieved by all agents. To model this process, we assume that each agent observes a small part of the global reward function, *e.g.*, each administrator observes the reward for the processes that terminate on its machine. Each agent i is associated with a local reward function $R_i(\mathbf{x}, \mathbf{a})$ whose scope $\mathsf{Scope}[R_i(\mathbf{x}, \mathbf{a})]$ is restricted to depend on a small subset of the state variables, and on the actions of only a few agents. The global reward function $R(\mathbf{x}, \mathbf{a})$ will be the sum of the rewards accrued by each agent $R(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{g} R_i(\mathbf{x}, \mathbf{a})$. In our multiagent SysAdmin example, the local reward function for agent i has scope restricted to its load

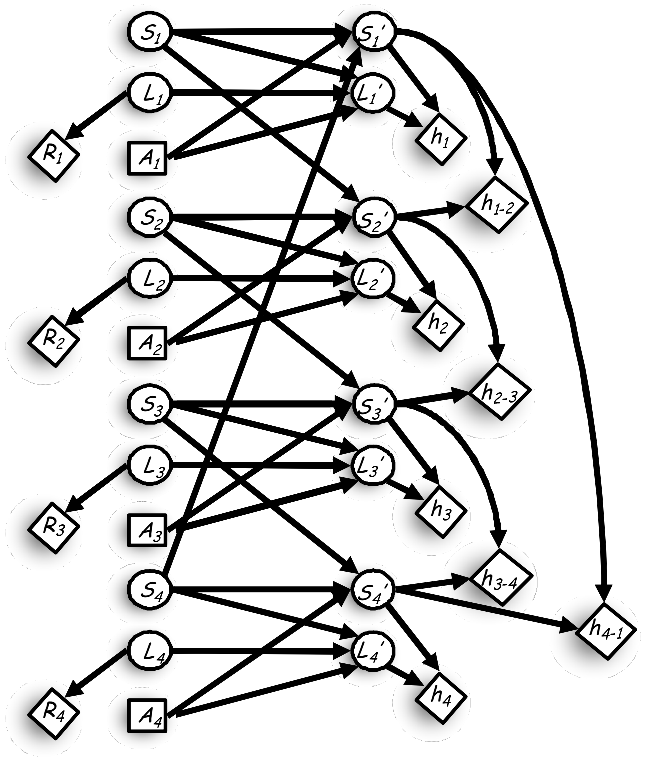


Figure 8.2: Multiagent factored MDP example, DDN for ring of 4 computers, including nodes for basis functions on the right.

variable L_i, as shown by the diamonds in Figure 8.1(a). The total reward for a network is the sum of the rewards accrued by each machine $\sum_i R_i(L_i)$. In the ring topology example in Figure 8.1(c), the reward function becomes $R_1(L_1) + R_2(L_2) + R_3(L_3) + R_4(L_4)$.

8.2 Factored Q-functions

Although multiagent factored MDPs allow us to model large collaborative multiagent problems very compactly, an exact solution to these problems remains infeasible. To address this issue, we resort to the same approximate factored value function framework we used for single agent problems in Section 3.2. We approximate the global value function $\mathcal{V}(\mathbf{x})$ as the weighted sum of local basis functions $\mathcal{V}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x})$, where each basis function h_i has scope restricted to a subset of variables $\mathsf{Scope}[h_i] = \mathbf{C}_i$.

In this multiagent setting, we maintain a distributed representation of the value function, where we associate a subset of the basis functions with each agent. These agent basis functions are summed together to define this agent's value. The particular assignment of basis functions to agents can be made arbitrarily, though, as we will show, the assignment affects the observation and communication requirements of each agent.

Definition 8.2.1 (agent's value) *Let the* agent's value *for agent i be given by:*

$$\mathcal{V}_i(\mathbf{x}) = \sum_{h_j \in \mathsf{Basis}[i]} w_j h_j(\mathbf{x}), \tag{8.1}$$

where $\mathsf{Basis}[i] \subseteq \{h_1, \ldots, h_k\}$ is a subset of the basis functions which is associated with agent i, such that $\mathsf{Basis}[i] \cap \mathsf{Basis}[j] = \emptyset,\ \forall i \neq j$, and $\bigcup_{i=1}^{g} \mathsf{Basis}[i] = \{h_1, \ldots, h_k\}$. ▮

In Figure 8.2, we have added two types of basis functions represented by diamonds in the next time step in our DDN: h_i, whose scope includes status S_i and load L_i of the ith computer, and $h_{i-(i+1)}$, with scope including status S_i of the ith computer and status S_{i+1} of the $i + 1$th computer. In this example, the basis functions associated with agent 1 are $\mathsf{Basis}[1] = \{h_1, h_{1-2}\}$.

Given a particular set of weights $\mathbf{w}$, we again choose the agents actions by computing the greedy policy with respect to this value function. Recall that greedy policy with respect to a value function $\mathcal{V}$ is given by $\mathsf{Greedy}[\mathcal{V}](\mathbf{x}) = \max_{\mathbf{a}} Q(\mathbf{x}, \mathbf{a})$, where the Q-function is again defined by:

$$Q(\mathbf{x}, \mathbf{a}) = R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \mathcal{V}(\mathbf{x}). \tag{8.2}$$

In Section 3.3, we show that by backprojecting our factored value function through the transition graph for each action a we can compute the Q-function efficiently.

In multiagent case, we no longer have a transition model for each action, but we can use a very similar procedure to compute the Q function efficiently by decomposing the global Q-function as a sum of local Q-functions for each agent:

Definition 8.2.2 (local Q-function) *The* local Q-function *for agent i is given by:*

$$Q_i(\mathbf{x}, \mathbf{a}) = R_i(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_{h_j \in \mathsf{Basis}[i]} w_j h_j(\mathbf{x}'). \quad ▮ \tag{8.3}$$

```
$Backproj(h)$ — WHERE BASIS FUNCTION $h$ HAS SCOPE $\mathbf{C}$.
    DEFINE THE SCOPE OF THE BACKPROJECTION: $\Gamma(\mathbf{C}') = \cup_{X_i' \in \mathbf{C}'} \textsf{PARENTS}(X_i')$.
    FOR EACH ASSIGNMENT $\mathbf{y} \in \mathrm{Dom}[\Gamma(\mathbf{C}')]$:
        $g(\mathbf{y}) = \sum_{\mathbf{c}' \in \mathbf{C}'} \prod_{i | X_i' \in \mathbf{C}'} P(\mathbf{c}'[X_i'] \mid \mathbf{y}) h(\mathbf{c}')$.

    RETURN $g$.
```

Figure 8.3: Backprojection of basis function h through a DDN.

Using this notation, our global Q-function is defined as:

$$Q(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{g} Q_i(\mathbf{x}, \mathbf{a}). \tag{8.4}$$

Each local Q-function can be computed efficiently using the backprojection procedure. Consider a basis function h; its backprojection is defined by $g(\mathbf{x}, \mathbf{a}) = \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h(\mathbf{x}')$. If the scope of h is restricted to $\textsf{Scope}[h] = \mathbf{Y}$, the scope of g will be defined by the backprojected scope of $\mathbf{Y}$ through our DDN, *i.e.*, the set of parents of $\mathbf{Y}'$ in the DDN:

$$\Gamma(\mathbf{Y}') = \cup_{Y_i' \in \mathbf{Y}'} \textsf{Parents}(Y_i').$$

In a multiagent problem, $\Gamma(\mathbf{Y}')$ will now include both state variables $\mathbf{X}$ and agent variables $\mathbf{A}$, that is, $\textsf{Scope}[g] \subseteq \{\mathbf{X}, \mathbf{A}\}$. In the example in Figure 8.2, the backprojected scope of $\{S_i', S_{i+1}'\}$ is given by $\Gamma(S_i', S_{i+1}') = \{S_{i-1}, S_i, S_{i+1}, A_i, A_{i+1}\}$.

If intra-time-slice arcs are included, so that

$$\textsf{Parents}(X_i') \in \{X_1, \ldots, X_n, A_1, \ldots, A_g, X_1', \ldots, X_n'\},$$

then the only change in our algorithm is in the definition of backprojected scope of $\mathbf{Y}$. The definition now includes not only direct parents of Y', but also all variables in $\{\mathbf{X}, \mathbf{A}\}$ that are ancestors of Y':

$$\Gamma(\mathbf{Y}') = \{B \in \{\mathbf{X}, \mathbf{A}\} \mid \text{there exists a directed path from } B \text{ to any } X_i' \in \mathbf{Y}'\}.$$

Figure 8.3 shows the backprojection procedure for a DDN. We denote the backprojection of basis function h by $g = Backproj(h)$, where $\textsf{Scope}[g]$ may include both state and agent variables. Using this notation, we can represent our local Q-function for agent i by:

$$Q_i(\mathbf{x}, \mathbf{a}) = R_i(\mathbf{x}, \mathbf{a}) + \gamma \sum_{h_j \in \mathsf{Basis}[i]} w_j g_j(\mathbf{x}, \mathbf{a}).$$

Thus, each local Q-function Q_i for agent i is the sum of restricted-scope functions. The global *factored Q-function* is the sum of the local Q-functions:

$$Q(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{g} Q_i(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{g} R_i(\mathbf{x}, \mathbf{a}) + \gamma \sum_{j=1}^{k} w_j g_j(\mathbf{x}, \mathbf{a}).$$

Although the total scope of Q_i, *i.e.*, $\mathsf{Scope}[Q_i] = \mathsf{Scope}[R_i] \cup \{\bigcup_{h_j \in \mathsf{Basis}[i]} \mathsf{Scope}[g_j]\}$, may be significantly larger than the scope of each g_j or of R_i. For simplicity of presentation, we assume that $\mathsf{Scope}[Q_i]$ is restricted to a small subset of the state and agents variables. Note that all of our methods can exploit further decomposition of Q_i; the purpose of this assumption is to simplify our notation and exposition. In our multiagent SysAdmin example, $\mathsf{Scope}[g_i] = \{S_{i-1}, S_i, L_i, A_i\}$, and $\mathsf{Scope}[g_{i-(i+1)}] = \{S_{i-1}, S_i, S_{i+1}, A_i, A_{i+1}\}$. Although the scope of Q_i is

$$\mathsf{Scope}[Q_i] = \{S_{i-1}, S_i, S_{i+1}, L_i, A_i, A_{i+1}\},$$

our algorithms exploit the locality of g_i and $g_{i-(i+1)}$.

8.3 Discussion and related work

Influence diagrams [Howard & Matheson, 1984] provide a graphical representation for decision processes involving multiple action variables. Multiagent factored MDPs, described in this chapter, combine influence diagrams with the DBN representation of Dean and Kanazawa [1989], to define a dynamic decision diagrams, a compact representation for large-scale collaborative multiagent planning problems.

We showed that, by combining the factored value function representation used thus far in this thesis with multiagent factored MDPs, we obtain a factored representation of the Q-function. This factored Q-function, given by the sum of local Q-functions, can then be stored in a distributed fashion, where agent i maintains the representation of the local term Q_i. Due to the factorizations of the value function and of the multiagent MDP, the scope of each term Q_i now depends on a subset of the state variables, as in the first part of this thesis, and on the actions of a subset of the agents. This last property is the key element in our efficient coordination algorithms described in the next chapter.

Chapter 9

Multiagent coordination and planning

In the previous chapter, we described multiagent factored MDPs, a compact representation for large-scale collaborative multiagent problems. Unfortunately, as in the single agent case, exact solutions for multiagent factored MDPs are intractable. Here, in addition to an exponentially-large state space, the size of the action space grows exponentially in the number of agents. As discussed in Chapter 1, multiagent settings have additional requirements. Exact solutions force each agent, online, to observe the full state of the system, and a centralized procedure that computes the maximal joint action at each time step. Both of these requirements will hinder the applicability of automated methods in many practical problems. To address this problem, we suggest, in Chapter 1, that agents should coordinate, while only observing a small subset of the state variables, and communicating with only a few other agents.

In this chapter, we exploit structure in multiagent factored MDPs to obtain exact solutions to the coordination problem and approximate solutions to multiagent planning problems: First, we present an efficient distributed action selection mechanism for tackling the exponentially-large maximization in $\arg\max_{\mathbf{a}} \sum_{i=1}^{g} Q_i(\mathbf{x}, \mathbf{a})$ required for agents to coordinate their actions. Then, we describe a simple extension to the linear programming-based approximation algorithm, which allows us to obtain approximate solutions to multiagent planning problems very efficiently.

9.1 Cooperative action selection

In this section, we assume that our basis function weights $\mathbf{w}$ are given, and consider the problem of computing the optimal greedy action that maximizes the approximate Q-function. In the next section, we address the problem of finding $\mathbf{w}$ which yields a good approximate value function.

The optimal greedy action for state $\mathbf{x}$ using our factored Q-function approximation is given by:

$$\arg\max_{\mathbf{a}} Q(\mathbf{x}, \mathbf{a}) = \arg\max_{\mathbf{a}} \sum_i Q_i(\mathbf{x}, \mathbf{a}). \tag{9.1}$$

As the Q-function depends on the action choices of all agents, they must coordinate in order to select the jointly optimal action that maximizes Equation (9.1).

Our first task is to instantiate the current state $\mathbf{x}$ in our Q-function. A naïve approach would require each agent to observe the all state variables, an unreasonable requirement in many practical situations. Our distributed representation of the Q-function, described in the previous chapter, will allow us to address this problem: We divide the scope of the local Q-function Q_i associated with agent i into two parts, the state variables

$$\textsf{Obs}[Q_i] = \{X_j \in \mathbf{X} \mid X_j \in \textsf{Scope}[Q_i]\}$$

and the agent variables

$$\textsf{Agents}[Q_i] = \{A_j \in \mathbf{A} \mid A_j \in \textsf{Scope}[Q_i]\}.$$

Note that, at each time step, agent i only needs to observe the variables in $\textsf{Obs}[Q_i]$, and use these variables only to instantiate its own local Q-function Q_i. Thus, each agent will only need to observe a small subset of the state variables, significantly reducing the observability requirements for each agent. To differentiate our requirements from partially observable Markov decision processes [Sondik, 1971], we call this property *limited observability*, as each agent observes the small part of the system determined by the function approximation architecture, but the agents are jointly solving a fully observable problem.

At this point, each agent i has observed the variables in $\textsf{Obs}[Q_i]$ and will instantiate

Q_i accordingly. We denote the instantiated local Q-function by $Q_i^{\mathbf{x}}$. The scope of each instantiated local Q-function includes only agent variables, *i.e.*, $\mathsf{Scope}[Q_i^{\mathbf{x}}] = \mathsf{Agents}[Q_i]$.

Next, the agents must coordinate to determine the optimal greedy action, that is, the joint action $\mathbf{a}$ that maximizes $\sum_i Q_i^{\mathbf{x}}(\mathbf{a})$. Unfortunately, the number of joint actions is exponential in the number of agents, which makes a simple action enumeration procedure infeasible. Furthermore such a procedure would require a centralized optimization step, which is not desirable in many multiagent applications. We now present a distributed procedure that efficiently computes the optimal greedy action.

Our procedure leverages on a very natural construct we call a *coordination graph*. Intuitively, a coordination graph connects agents whose local Q-functions interact with each other and represents the coordination requirements of the agents:

Definition 9.1.1 *A* coordination graph *for a set of agents with local Q-functions* $\{Q_1, \ldots, Q_g\}$ *is a directed graph whose nodes are* $\{A_1, \ldots, A_g\}$*, and which contains an edge* $A_i \rightarrow A_j$ *if and only if* $A_i \in \mathsf{Agents}[Q_j]$. ▮

Computing the action that maximizes $\sum_i Q_i^{\mathbf{x}}$ requires a maximization of local functions in a graph structure, suggesting the use of *non-serial dynamic programming* [Bertele & Brioschi, 1972], the same variable elimination algorithm which we used in Chapter 4 for our LP decomposition technique. We first illustrate this algorithm with a simple example:

Example 9.1.2 *Consider a simple coordination problem with* 4 *agents, where the global Q-function is approximated by:*

$$Q = Q_1(a_1, a_2) + Q_2(a_2, a_4) + Q_3(a_1, a_3) + Q_4(a_3, a_4),$$

and we wish to compute $\arg\max_{a_1,a_2,a_3,a_4} Q_1(a_1, a_2) + Q_2(a_2, a_4) + Q_3(a_1, a_3) + Q_4(a_3, a_4)$. *The initial coordination graph associated with this problem is shown in Figure 9.1(a).*

Let us begin our optimization with agent 4. To optimize A_4*, functions* Q_1 *and* Q_3 *are irrelevant. Hence, we obtain:*

$$\max_{a_1,a_2,a_3} Q_1(a_1, a_2) + Q_3(a_1, a_3) + \max_{a_4}[Q_2(a_2, a_4) + Q_4(a_3, a_4)].$$

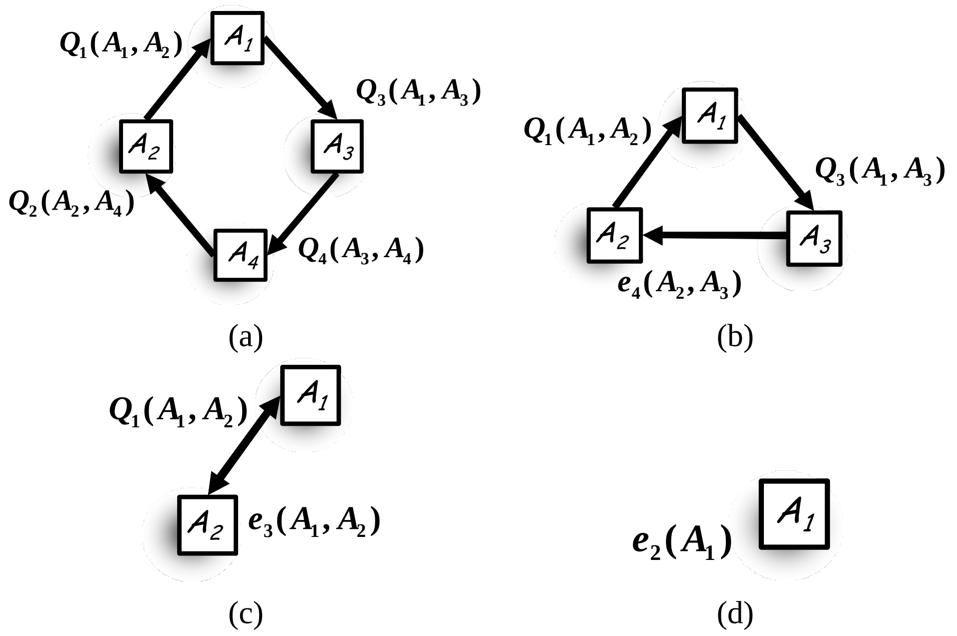


Figure 9.1: Example of distributed variable elimination in a coordination graph: (a) Initial coordination graph for a 4-agent problem; (b) after agent 4 performed its local maximization; (c) after agent 3 performed its local maximization; and (d) after agent 2 performed its local maximization.

We see that to make the optimal choice over A_4, the agent must know the values of A_2 and A_3. Additionally, agent A_2 must transmit Q_2 to A_4. In effect, agent A_4 is computing a conditional strategy, with a (possibly) different action choice for each action choice of agents 2 and 3. Agent 4 can summarize the value that it brings to the system in the different circumstances using a new function $e_4(A_2, A_3)$ whose value at the point a_2, a_3 is the value of the internal $\max$ *expression:*

$$e_4(a_2, a_3) = \max_{a_4}[Q_2(a_2, a_4) + Q_4(a_3, a_4)].$$

Agent 4 has now been "eliminated". The new function $e_4(a_2, a_3)$ is stored by agent 2 and the coordination graph is updated as shown in Figure 9.1(b).

Our problem now reduces to computing

$$\max_{a_1,a_2,a_3} Q_1(a_1, a_2) + Q_3(a_1, a_3) + e_4(a_2, a_3),$$

having one fewer agent involved in the maximization. Next, agent 3 makes its decision, giving:

$$\max_{a_1,a_2} Q_1(a_1,a_2) + e_3(a_1,a_2),$$

where $e_3(a_1,a_2) = \max_{a_3}[Q_3(a_1,a_3) + e_1(a_2,a_3)]$. *Once agent 3 is eliminated and the new function* $e_3(a_1,a_2)$ *is stored by agent* 2, *the coordination graph is updated as shown in Figure 9.1(c).*

Agent 2 now makes its decision, giving

$$e_2(a_1) = \max_{a_2}[Q_1(a_1,a_2) + e_3(a_1,a_2)],$$

The new function $e_2(a_1)$ *is stored by agent 1, and the coordination graph becomes simply a single node as shown in Figure 9.1(d).*

Agent 1 can now simply choose the action a_1 *that maximizes*

$$e_1 = \max_{a_1} e_2(a_1).$$

The result at this point is a scalar, e_1, *which is exactly the desired maximum over* $a_1,\ldots,a_4$.

We can recover the maximizing set of actions by performing the process in reverse: The maximizing choice for e_1 *defines the action* a_1^* *for agent 1:*

$$a_1^* = \arg\max_{a_1} e_2(a_1).$$

To fulfill its commitment to agent 1, agent 2 must choose the value a_2^* *which yielded* $e_2(a_1^*)$*:*

$$a_2^* = \arg\max_{a_2}[Q_1(a_1^*,a_2) + e_3(a_1^*,a_2)],$$

This, in turn, forces agent 3 and then agent 4 to select their actions appropriately:

$$a_3^* = \arg\max_{a_3}[Q_3(a_1^*,a_3) + e_4(a_2^*,a_3)],$$

and

$$a_4^* = \arg\max_{a_4}[Q_2(a_2^*,a_4) + Q_4(a_3^*,a_4)]. \quad \blacksquare$$

```
ARGVARIABLEELIMINATION ($\mathcal{F}$, $\mathcal{O}$, ELIMOPERATOR, ARGOPERATOR)
        // $\mathcal{F} = \{f_1, \ldots, f_m\}$ is the set of local functions.
        // $\mathcal{O}$ stores the elimination order.
        // ELIMOPERATOR is the operation used when eliminating variables.
        // ARGOPERATOR is the operation used to obtain the value of an eliminated variable.
    FOR $i = 1$ TO NUMBER OF VARIABLES:
        // Select the next variable to be eliminated.
        LET $l = \mathcal{O}(i)$ .
        // Select the relevant functions.
        CACHE THE SET $\mathcal{E}_l = \{e_1, \ldots, e_L\}$ OF FUNCTIONS IN $\mathcal{F}$ WHOSE SCOPE CONTAINS $A_l$.
        // Eliminate current variable $A_l$.
        LET $e =$ ELIMOPERATOR $(\mathcal{E}_l, A_l)$.
        // Update set of functions.
        UPDATE THE SET OF FUNCTIONS $\mathcal{F} = \mathcal{F} \cup \{e\} \setminus \{e_1, \ldots, e_L\}$.
    // Now, all functions have empty scopes, and the last step eliminates the empty set.
    LET $Z =$ ELIMOPERATOR $(\mathcal{F}, \emptyset)$.

    // We can obtain the assignment by eliminating the variables in the reverse order.
    LET $\mathbf{a}^* = \emptyset$.
    FOR $i =$ NUMBER OF VARIABLES DOWN TO $1$:
        // Select the next variable to be eliminated.
        LET $l = \mathcal{O}(i)$ .
        // Instantiate the functions corresponding to $A_l$.
        FOR EACH $e_i \in \mathcal{E}_l$:
            LET $e_i^*(a_l) = e_i(a_l, \mathbf{a}^*[\text{SCOPE}[e_i] - \{A_l\}]),\ \forall a_l \in A_l$.
            REPLACE $e_i$ WITH $e_i^*$ IN $\mathcal{E}_l$.
        // Compute assignment for $A_l$.
        LET $a_l^*$, THE ASSIGNMENT TO $A_l$ IN $\mathbf{a}^*$, BE $a_l^* =$ ARGOPERATOR $(\mathcal{E}_l, A_l)$.
    // Now, $\mathbf{a}^*$ has the assignment for all variables.
    RETURN THE ASSIGNMENT $\mathbf{a}^*$ AND VALUE OF THIS ASSIGNMENT $Z$.
```

Figure 9.2: Variable elimination procedure, where ELIMOPERATOR is used when a variable is eliminated and ARGOPERATOR is used to compute the argument of the eliminated variable. To compute the maximum assignment of $f_1 + \cdots + f_m$, and its value, where each f_i is a restricted-scope function, we must substitute ELIMOPERATOR with MAXOUT from Figure 4.2, and ARGOPERATOR with ARGMAXOUT from Figure 9.3.

```
ARGMAXOUT $(\mathcal{E}, A_l)$
        // $\mathcal{E} = \{e_1, \ldots, e_m\}$ is the set of functions that depend only on $A_l$.
        // $A_l$ variable to be maximized.
    RETURN $\arg\max_{a_l} \sum_{j=1}^{L} e_j$.
```

Figure 9.3: ARGMAXOUT operator for variable elimination, procedure that returns the assignment of variable A_l that maximizes $e_1 + \cdots + e_m$.

Figure 9.2 shows a simple extension of the variable elimination algorithm presented in Section 4.2. In this extension, we generalize the procedure used in the simple example above to an arbitrary set of functions $f_1, \ldots, f_m$. We divide this algorithm in two parts: The first part is exactly the maximization presented in Section 4.2. In the second part, we follow the variable elimination order in reverse to obtain the maximizing assignment. When computing the maximizing assignment for A_l, the ith variable to be eliminated, we have already computed the maximizing assignments to all variables later than i in the ordering. The scope of the cached local function f_l only depends on A_l and on the assignment to variables which appear later in the ordering, *i.e.*, whose optimal assignment has already been determined. We can thus compute A_l's optimal assignment a_l^* using a simple maximization over a_l.

The correctness of this approach is guaranteed by the correctness of variable elimination:

Theorem 9.1.3 *For any ordering $\mathcal{O}$ on the variables, the* ArgMaxVariableElimination *procedure computes the optimal greedy action for each state* $\mathbf{x}$, *that is:*

$$\textsc{ArgMaxVariableElimination}(\{Q_1^{\mathbf{x}}, \ldots, Q_g^{\mathbf{x}}\}, \mathcal{O}, \ \textsc{MaxOut}, \ \textsc{ArgMaxOut}) \in \arg\max_{\mathbf{a}} \sum_{i=1}^{g} Q_i^{\mathbf{x}}(\mathbf{a}).$$

Proof: *See for example the book by Bertele and Brioschi [1972].* ∎

As with the basic variable elimination procedure in Section 4.2, the cost of this algorithm is linear in the number of new "function values" introduced, or in our multiagent coordination case, only exponential in the *induced width* of the coordination graph.

The variable elimination algorithm can thus be used for computing the optimal greedy action very efficiently, in a centralized fashion. However, in practical multiagent coordination problems, we often need to use a distributed algorithm to avoid the need for any centralized computation. We have two coordination options in such a distributed procedure: In a *synchronous* implementation, each agent computes its local maximization (conditional strategy) by following a pre-specified ordering over agents. In an (more robust) *asynchronous* implementation, the elimination order is determined at runtime. We present only the simpler synchronous implementation, as the asynchronous extension is straightforward.

```
DISTRIBUTEDACTIONSELECTION($i$)
// Distributed action selection algorithm for agent $i$.
    REPEAT EVERY TIME STEP $t$:
        // INSTANTIATION.
        // Instantiate the current state.
        OBSERVE THE VARIABLES $\mathsf{OBS}[Q_i]$ IN THE CURRENT STATE $\mathbf{x}^{(t)}$.
        INSTANTIATE THE LOCAL Q-FUNCTION WITH THE CURRENT STATE:
            $Q_i^{\mathbf{x}^{(t)}}(\mathbf{a}) = Q_i(\mathbf{x}^{(t)}, \mathbf{a}).$

        // INITIALIZATION.
        // Initialize the coordination graph.
        LET THE PARENTS OF $A_i$ BE THE AGENTS IN $\mathsf{SCOPE}[Q_i^{\mathbf{x}^{(t)}}] = \mathsf{AGENTS}[Q_i]$.
        STORE $Q_i^{\mathbf{x}^{(t)}}$.

        // Maximization.
        // Wait for signal from parent of $i$ in the variable elimination order.
        WAIT FOR SIGNAL FROM AGENT $\mathcal{O}_i^-$, IF $\mathcal{O}_i^- = \emptyset$ CONTINUE.
        // We can now compute the maximization for agent $i$.
        // First we collect the functions that depend on $A_i$, i.e., the ones stored by $i$ and by the
          children of $i$ in the coordination graph.
        COLLECT THE LOCAL FUNCTIONS $e_1, \ldots, e_L$ FROM THE CHILDREN OF $i$ IN THE CO-
          ORDINATION GRAPH, AND THE ONES STORED BY AGENT $i$.
        CACHE THIS SET $\mathcal{E}_i = \{e_1, \ldots, e_L\}$ OF FUNCTIONS IN WHOSE SCOPE CONTAINS $A_i$.
        // Eliminate current variable $A_l$.
        LET $e =$ MAXOUT $(\mathcal{E}_l, A_l)$.
        // Update the coordination graph.
        STORE THE NEW FUNCTION $e$ WITH SOME AGENT $A_j \in \mathsf{SCOPE}[e]$.
        DELETE $A_i$ FROM THE COORDINATION GRAPH AND ADD EDGES FROM THE AGENTS
          IN $\mathsf{SCOPE}[e]$ TO $A_j$.
        SIGNAL AGENT $\mathcal{O}_i^+$.

        // Action selection.
        // Wait for signal from child of $i$ in the variable elimination order.
        WAIT FOR SIGNAL FROM AGENT $\mathcal{O}_i^+$; IF $\mathcal{O}_i^+ = \emptyset$ INITIALIZE $\mathbf{a}^{(t)} = \emptyset$ AND CONTINUE.
        RECEIVE THE CURRENT ASSIGNMENT TO THE MAXIMIZING ACTION $\mathbf{a}^{(t)}$ FROM
          AGENT $\mathcal{O}_i^+$.
        // We can now compute the maximizing action for agent $i$.
        // Instantiate the functions corresponding to $A_i$.
        FOR EACH $e_j \in \mathcal{E}_i$:
            LET $e_j^*(a_i) = e_j(a_i, \mathbf{a}^*[\mathsf{SCOPE}[e_j] - \{A_i\}]),\ \forall a_i \in A_i$.
            REPLACE $e_j$ WITH $e_j^*$ IN $\mathcal{E}_i$.
        // Compute assignment for $A_i$.
        LET $a_i^*$, THE ASSIGNMENT TO $A_i$ IN $\mathbf{a}^*$, BE $a_i^* =$ ARGMAXOUT $(\mathcal{E}_i, A_i)$.
        // Signal to next agent.
        SIGNAL AGENT $\mathcal{O}_i^-$ AND TRANSMIT $\mathbf{a}^{(t)}$.
```

Figure 9.4: Synchronous distributed variable elimination on a coordination graph.

As in the standard variable elimination algorithm, this synchronous implementation requires an elimination order $\mathcal{O}$ on the agents, where $\mathcal{O}(i)$ returns the ith agent to be maximized. Agents do not need knowledge of the full elimination order. Agent $j = \mathcal{O}(i)$ only needs to know the agents that come before and after it in the ordering, *i.e.*, $\mathcal{O}_j^- = \mathcal{O}(i-1)$ and $\mathcal{O}_j^+ = \mathcal{O}(i+1)$ respectively. To simplify our notation, $\mathcal{O}_j^- = \emptyset$ for the first agent in the ordering and $\mathcal{O}_j^+ = \emptyset$ for the last one.

Figure 9.4 presents the complete algorithm that will be executed by agent i. At every time step, the procedure follows 4 phases:

1. **Instantiation:** The agent makes local observations and instantiates the current state in its local Q-function, Q_i, resulting in $Q_i^{\mathbf{x}}$.
2. **Initialization:** The edges in the coordination graph are initialized, with agent i initially storing only the $Q_i^{\mathbf{x}}$ function.
3. **Maximization:** When it is agent i's turn to be eliminated, it collects the local functions $e_1, \ldots, e_L$ whose scope include A_i, *i.e.*, those functions stored by the children of A_i in the coordination graph and those stored by agent i. These functions are cached as $f_i = \sum_j e_j$. Agent i can now perform its local maximization by defining a new function $e = \max_{a_i} f_i$, the scope of e is $\cup_{j=1}^{L} \mathsf{Scope}[e_j] - \{A_i\}$. As the scope of this new function e does not contain A_i, it should now be stored by some different agent j such that $A_j \in \mathsf{Scope}[e]$. At this point, agent i has been eliminated, *i.e.*, there are no functions whose scope includes A_i, and the coordination graph is updated accordingly.
4. **Action selection:** The optimal action choice can be computed by following the reverse order over agents. When it is agent i's turn, all agents later than i in the ordering have already computed their optimal action and stored it in $\mathbf{a}^*$. The scope of the cached local function f_i only depends on A_i and on the actions of agents later in the ordering, whose optimal action has already been determined. Agent i can thus compute its optimal action choice a_i^* using a simple maximization over a_i.

The correctness of this distributed procedure is a corollary of Theorem 9.1.3:

Corollary 9.1.4 *For any ordering $\mathcal{O}$ over agents, if each agent executes the procedure in Figure 9.4, the agents will jointly compute the optimal greedy action $\mathbf{a}^{(t)}$ for each state $\mathbf{x}^{(t)}$, that is:*

$$\mathbf{a}^{(t)} \in \arg\max_{\mathbf{a}} \sum_{i=1}^{g} Q_i^{\mathbf{x}^{(t)}}(\mathbf{a}). \quad \blacksquare$$

It is important to note that in our distributed version of variable elimination, each agent does not need to communicate directly with every other agent in the system. Agent i only needs to communicate with agent j if the scope of one of the functions generated in our maximization procedure includes both A_i and A_j. We call this property *limited communication*, that is, rather than communicating with every agent in the environment, in our approach, agents only needs to communicate with a small set of other agents. The communication bandwidth required by our algorithm is directly determined by the induced width of the coordination graph. We note that the centralized version of our algorithm is essentially a special case of the algorithm used to solve influence diagrams with multiple parallel decisions [Jensen *et al.*, 1994]. However, to our knowledge, these ideas have not been applied to the problem of online coordination in the decision making process of multiple collaborating agents in a dynamic system.

Our distributed action selection scheme can be implemented as a negotiation procedure for selecting actions at run time. Alternatively, if all agents observe the complete state vector $\mathbf{x}$ at every time step, and these agents agree on a tie-breaking scheme upfront, each agent can efficiently determine the actions that will be taken by all of the collaborating agents without any communication at all. Thus, in such cases, each agent i would individually use the variable elimination algorithm in Figure 9.2, and take its optimal action a_i^* for the current state. Thus, there is a tradeoff between full observability by each agent with no communication required between the agents, and limited observability for each agent, but with some additional communication requirements.

9.2 Approximate planning for multiagent factored MDPs

In the previous section, we presented an efficient online distributed algorithm for selecting the optimal greedy action for multiagent problem whose value is approximated by a factored Q-function. In Section 8.2, we show that a factored approximation to the value function, *i.e.*, one where the value function is approximated as a linear combination of basis functions $\sum_i w_i h_i$, yields the necessary factored structure in the Q-function. We now present a small extension to the linear programming-based approximation algorithm in Section 5.1, which computes the weights $\mathbf{w}$ in our factored value function $\sum_i w_i h_i$.

As discussed in Section 2.3.2, the linear programming-based approximation formulation is based on the exact linear programming approach for solving MDPs presented in Section 2.2.1. However, in this approximate version, we restrict the space of value functions to the linear space defined by our basis functions. More precisely, in this approximate LP formulation, the variables are $w_1, \ldots, w_k$ — the weights for our basis functions. The LP is given by:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k \ ; \\ \text{Minimize:} & \sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \ h_i(\mathbf{x}) \ ; \\ \text{Subject to:} & \sum_i w_i \ h_i(\mathbf{x}) \geq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i \ h_i(\mathbf{x}') \ , \quad \forall \mathbf{x} \in \mathbf{X}, \ \mathbf{a} \in \mathbf{A}. \end{array} \tag{9.2}$$

This is exactly the same LP formulation as the one in (5.1), except that now our constraints span all possible joint assignments to the actions of the agents $\mathbf{a} \in \mathbf{A}$.

The decomposition of the LP in (9.2) follows the same procedure used in the single agent formulation in Section 5.1. First, the objective function is decomposed as:

$$\sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \ h_i(\mathbf{x}) = \sum_i w_i \sum_{\mathbf{c}_i \in \mathrm{Dom}[\mathbf{C}_i]} \alpha(\mathbf{c}_i) \ h_i(\mathbf{c}_i) = \sum_i \alpha_i w_i. \tag{9.3}$$

The we reformulate the constraints as:

$$0 \geq R(\mathbf{x}, a) + \sum_i w_i \left[\gamma g_i(\mathbf{x}, \mathbf{a}) - h_i(\mathbf{x}) \right], \ \ \forall \mathbf{x} \in \mathbf{X}, \ \mathbf{a} \in \mathbf{A}, \tag{9.4}$$

where the backprojection $g_i(\mathbf{x}, \mathbf{a}) = \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h_i(\mathbf{x}')$ is a restricted domain function computed efficiently as described in Figure 8.3. Using the same transformation we applied in the single agent case in Section 2.2.1, we can rewrite this exponentially-large set of constraints as a single, equivalent, non-linear constraint:

```
MULTIAGENTFACTOREDLPA ($P$, $R$, $\gamma$, $H$, $\mathcal{O}$, $\alpha$)
        // $P$ is the factored multiagent transition model.
        // $R$ is the set of factored reward functions.
        // $\gamma$ is the discount factor.
        // $H$ is the set of basis functions $H = \{h_1, \dots, h_k\}$.
        // $\mathcal{O}$ stores the elimination order for all state $\mathbf{X}$ and agent $\mathbf{A}$ variables.
        // $\alpha$ are the state relevance weights.
        // Return the basis function weights $\mathbf{w}$ computed by linear programming-based approximation.
    // Cache the backprojections of the basis functions.
    FOR EACH BASIS FUNCTION $h_i \in H$:
        LET $g_i = Backproj(h_i)$.
    // Compute factored state relevance weights.
    FOR EACH BASIS FUNCTION $h_i$, COMPUTE THE FACTORED STATE RELEVANCE WEIGHTS $\alpha_i$
      AS IN EQUATION (9.3).
    // Generate linear programming-based approximation constraints
    LET $\Omega = $ FACTOREDLP$(\{\gamma g_1 - h_1, \dots, \gamma g_k - h_k\}, R, \mathcal{O})$.
    // So far, our constraints guarantee that $\phi \geq R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_i w_i\ h_i(\mathbf{x}') -$
      $\sum_i w_i\ h_i(\mathbf{x})$; to satisfy the linear programming-approximation solution in (9.2) we must add
      a final constraint.
    LET $\Omega = \Omega \cup \{\phi = 0\}$.
    // We can now obtain the solution weights by solving an LP.
    LET $\mathbf{w}$ BE THE SOLUTION OF THE LINEAR PROGRAM: MINIMIZE $\sum_i \alpha_i w_i$, SUBJECT TO THE
      CONSTRAINTS $\Omega$.
    RETURN $\mathbf{w}$.
```

Figure 9.5: Multiagent factored linear programming-based approximation algorithm.

- **Offline:**
  1. Select a set of restricted-scope basis functions $\{h_1, \dots, h_k\}$.
  2. Apply efficient LP-based approximation algorithm as shown in Figure 9.5 to compute coefficients $\{w_1, \dots, w_k\}$ of the approximate value function $\mathcal{V} = \sum_j w_j h_j$.
  3. Use the one-step lookahead planning algorithm (Section 8.2) with $\mathcal{V}$ as a value function estimate to compute local Q_i functions for each agent.
- **Online:**
  - Each agent i executes the distributed procedure in Figure 9.4 to compute the greedy policy:
    1. Each agent i instantiates its local Q_i function with values of state variables in scope of Q_i.
    2. Agents apply distributed variable elimination on the coordination graph with local Q_i functions to compute the optimal greedy action.

Figure 9.6: Our approach for multiagent planning with factored MDPs.

Number of agents	Optimal policy	LP-based approximation	
		“single” basis	“pair” basis
1	4.27	4.36 $\pm$ 0.18	4.36 $\pm$ 0.18
2	4.16	4.27 $\pm$ 0.20	4.28 $\pm$ 0.21
3	4.16	3.96 $\pm$ 0.24	4.16 $\pm$ 0.16

Table 9.1: Comparing value per agent of policies on the multiagent SysAdmin problem with “ring” topology: optimal policy versus LP-based approximation with “single” and with “pair” basis functions. Value of approximate policies estimated by 20 runs of 100 steps.

$$0 \geq \max_{\mathbf{x},\mathbf{a}} R(\mathbf{x}, a) + \sum_i w_i \left[\gamma g_i(\mathbf{x}, \mathbf{a}) - h_i(\mathbf{x})\right]. \tag{9.5}$$

The difference between this constraint and the one in the single agent LP in (5.4) is that our maximization $\max_{\mathbf{x},\mathbf{a}}$ is now over both the state and agent variables.

We can use our factored LP decomposition technique in Chapter 4 to represent this non-linear constraint exactly, and in closed form, using a set of linear constraints that is exponentially smaller than the one in Equation (9.4). Note that our LP decomposition technique is now applied over both state and action variables. Thus, the variable elimination order $\mathcal{O}$ should now give us an ordering over both state and action variables. Figure 9.5 presents the complete multiagent factored LP-based approximation algorithm. Our overall algorithm for multiagent planning and coordination with factored MDPs in shown in Figure 9.6.

9.3 Empirical evaluation

We first evaluate our algorithms on the multiagent version of the SysAdmin problem presented in Example 8.1.1. Recall that, for a network of n machines, the number of states in the MDP is 9^n and the joint action space contains 2^n possible actions, *e.g.*, a problem with 30 agents has over 10^{28} states and a billion possible actions.

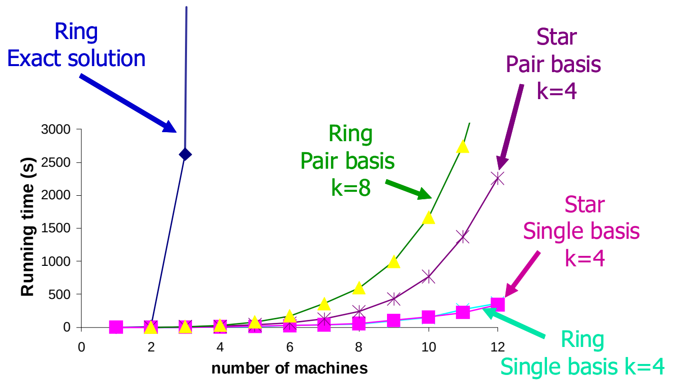
We implemented our factored multiagent LP-based approximation algorithm in C++, using CPLEX as our LP solver. The experiments were run on a Pentium III 700MHz with 1GB of RAM. We experimented with two types of basis functions: “single”, which contains an indicator basis function for each value of each S_i and L_i; and "pair" which, in addition, contains indicators over joint assignments of the Status variables of neighboring agents. We use a discount factor γ of 0.95.

For small problems, we can run an exact solution algorithm for computing the value of the optimal policy. These values can then be compared to the value of the approximate policies computed by our factored multiagent LP-based approximation algorithm. The results in Table 9.1 compare the value of the two policies for an initial state with all machines working. These results indicate that, for these small problems, the quality of our approximate solutions is very close to that of the optimal policy.

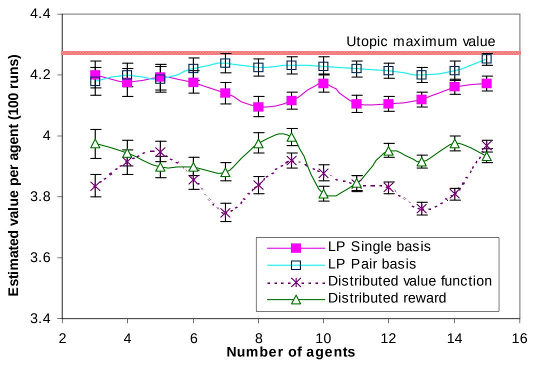
As shown in Figure 9.7(a), the running time of the exact solution algorithm grows exponentially in the number of agents, as expected. In contrast, the time required by our factored approximate algorithm grows only quadratically in the number of agents, for each fixed network and basis type. This is the expected asymptotic behavior, as each problem has a fixed induced tree width of our factored LP. The policies obtained tended to be intuitive: *e.g.*, for the "star" topology with pair basis, if the server becomes faulty, it is rebooted even if loaded. but for the clients, the agent waits until the process terminates or the machine dies before rebooting.

For comparison, we also implemented the distributed reward (DR) and distributed value function (DVF) algorithms of Schneider *et al.* [1999]. These algorithms define a local value function for each agent that may depend on the state of this agent and of a few other agents. These local value functions are then optimized simultaneously using a Q-learning-style update rule. This update rule is modified for each agent by including a term that depends on the neighboring agents' reward for DR, or value function for DVF.

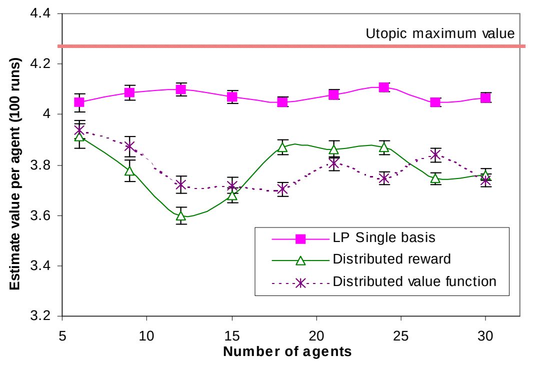
Our implementation of DR and DVF used 10000 learning iterations, with learning and exploration rates starting at 0.1 and 1.0 respectively and a decaying schedule after 5000 iterations; the observations for each agent were the status and load of its machine. The results of the comparison are shown in Figure 9.7(b) and (c). We also computed a utopic upper bound on the value of the optimal policy by removing the (negative) effect of the neighbors on the status of the machines. This is a loose upper bound, as a dead neighbor increases the probability of a machine dying by about 50%. For both network topologies tested, the estimated value of the approximate LP solution using single basis was significantly higher



(a)



(b)



(c)

Figure 9.7: Multiagent SysAdmin problem: (a) Running time for LP-based approximation versus the exact solution for increasing number of agents (induced width k of the underlying factored LP is shown). Policy performance of our LP-based approximation versus the DR and DVF algorithms [Schneider *et al.*, 1999] on: (b) “star” topology, and (c) “ring of rings” topology.

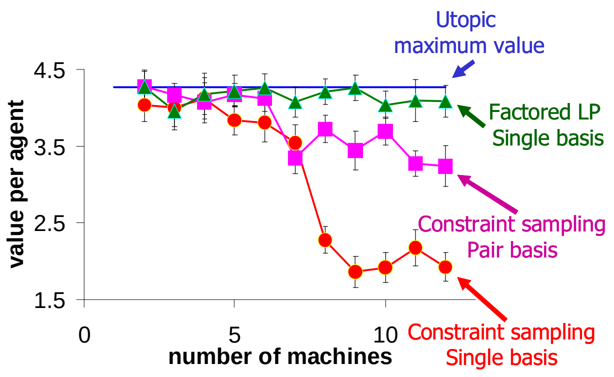


Figure 9.8: Comparing the quality of the policies obtained using our factored LP decomposition technique with constraint sampling.

than that of the DR and DVF algorithms. Note that the single basis solution requires no coordination when acting, so, in this sense, this is a "fair" comparison to DR and DVF which also do not communicate while acting. If we allow for pair bases, which implies agent communication, we achieve a further improvement in terms of estimated value.

Our factored LP decomposition technique represents the exponentially-large constraint set in the LP-based approximation formulation compactly and in closed form. An alternative to our decomposition technique is to solve the same optimization problem with a tractable subset of this exponentially-large constraint set. Recently, de Farias and Van Roy [2001b] analyze an algorithm that uses sampling to select such a subset. In Figure 9.8, we compare this sampling approach with our LP decomposition technique. Both algorithms were executed with the same set of basis functions. The number of sampled constraints was such that the running time was equal for both algorithms, for each set of basis functions. We used a simple uniform sampling distribution to generate constraints. As shown by de Farias and Van Roy [2001b], the choice of distribution may affect the quality of the solutions obtained by the sampling approach. They also suggest some heuristics for choosing a good sampling distribution in some queueing problems. It is possible that a non-uniform distribution could have improved the performance of the sampling approach, in the SysAdmin problem.

For smaller problems both sampling and our factored LP approach obtained policies with similar value. However, as the problem size increase, the quality of the policies obtained by sampling constraints deteriorated, while the ones generated with our factored LP maintained their value. If we apply the sampling algorithm with “pair” basis (and, thus, with the same running time as our factored LP approach with “pair” basis), the quality of the policies deteriorates more slowly as the problem size increases. However, the policies obtained by our factored LP approach with “single” basis are still better than the ones obtained by the sampling approach with “pair” basis (and a longer running time). We compare and contrast these two approaches further in the discussion below.

9.4 Discussion and related work

We provide a principled and efficient approach for planning in collaborative multiagent domains. Rather than placing *a priori* restrictions on the communication structure between agents, we first choose the form of the approximate factored value function and derive the optimal communication structure given the value function architecture. This approach provides a unified view of value function approximation and agent communication, as a better approximation will often require more communication between agents. We use a simple extension of our factored LP-based approximation algorithm to find an approximately optimal value function. The inter-agent communication and the LP avoid the exponential blowup in the state and action spaces, having computational complexity depend, instead, upon the induced tree width of the coordination graph used by the agents to negotiate their action selection.

Alternative approaches to this problem have used local optimization for the different agents, either via reward/value sharing [Schneider *et al.*, 1999; Wolpert *et al.*, 1999], including the algorithms we evaluate in Section 9.3, or direct policy search [Peshkin *et al.*, 2000]. In contrast, we provide a global optimization procedure, where agents can explicitly coordinate their actions. An important difference between the methods of Schneider *et al.* [1999] and our approach is that, although the agents communicate during learning with their approach, there is no communication between agents at runtime. The method of Peshkin *et al.* [2000] requires no communication between agents both during learning or at runtime.

The most closely related approach to our is that of Sallans and Hinton [2001], who use a product of experts to approximate the Q-function. Action selection is intractable in such models, and the authors address this problem by using Gibbs sampling [Geman & Geman, 1984]. The weights of the product of experts are optimized using a local search procedure. On the other hand, we restrict our value function to linear approximations. This restriction allows us to optimize the weights using a (convex) linear program, removing the reliance on local search methods, and lets us perform the action selection step optimally, in a distributed fashion, using the coordination graph.

We present empirical evaluations of the quality of the policies generated by our multiagent planning algorithm. For small multiagent problems, where we could obtain the optimal solution, we showed that our LP-based approximation algorithm obtains policies with near-optimal value. For larger problem, we could only compare the value of our policies with a loose theoretical upper bound on the value of the optimal policy. For these problems, our policies were again near-optimal, with significantly better values that those obtained with the algorithms of Schneider *et al.* [1999]. The running time of our algorithm, as expected, demonstrated polynomial scaling for problems with fixed induced width. Furthermore, the quality of our policies did not show decay in value as the problem size increased.

Boutilier [1996] partitions coordination methods for collaborative multiagent planning problems into ones where the agents negotiate their actions via communication, and ones where the coordination follows from social convention. As discussed at the end of Section 9.1, our coordination procedure can be implemented to fit both of these classes: As described, our distributed action selection scheme requires local communication between agents. Alternatively, if all agents observe the complete state vector $\mathbf{x}$ at every time step, and these agents agree on a tie-breaking scheme upfront (the social convention), each agent can then use variable elimination to compute its own action. This process is guaranteed to yield the globally optimal greedy action. Thus, our algorithm provides an intuitive tradeoff: at one end of the spectrum, we have full observability by each agent with no communication required between the agents, and, at the other end, limited observability for each agent, but with some additional communication requirements.

The analysis of constraint sampling of de Farias and Van Roy [2001b], discussed in more detail in Section 7.8.1, provides an alternative to our factored LP decomposition technique. The number of samples in the result of de Farias and Van Roy [2001b] depends on the number of actions in the MDP, which is exponential in multiagent problems. They also present an equivalent formulation where the state space is augmented with a state variable to indicate the choice of each action variable. At every time step, the agent then sets one of these state variables, in order. The number of actions in this modified formulation is now equal to the size of the domain of each action variable. The theoretical scaling of the number of samples thus depends on the log of the number of joint actions, but the size of the state space is multiplied by the number of joint actions. The increased number of states will probably increase the number of basis functions needed for a good approximation. Furthermore, as discussed by de Farias and Van Roy [2001b], their method can often be quite sensitive to the choice of sampling distribution. Our factored LP can efficiently decompose the exponentially-large constraint set in multiagent problems modelled as factored MDPs, in closed form. Thus, in structured multiagent systems, while the sampling method of de Farias and Van Roy [2001b] will apply to more general problems that cannot be represented compactly by factored MDPs. We present a preliminary empirical comparison of the two methods on a problem that can be represented by a factored MDP. We attempt to make the comparison "fair" by giving both algorithms the same amount of computer time, though we use a uniform sampling distribution for the method of de Farias and Van Roy [2001b]. A non-uniform distribution could potentially improve the quality of their approximation. The policies obtained by our methods outperformed those obtained by sampling constraints, even when sampling was given a more expressive basis function space, and increased running time.

Chapter 10

Variable coordination structure

In the previous chapter, we presented efficient coordination and planning algorithms for multiagent systems. However, this approach assumes that each agent only needs to interact with a small number of other agents. In many situations, an agent can *potentially* interact with many other agents, but not at the *same* time. For example, two agents that are both part of a construction crew might need to coordinate at times when they could both be working on the same task, but not at other times. If we use the approach presented in the previous chapter, we are forced to represent value functions over large numbers of agents, rendering the approach intractable.

In this chapter, we exploit *context specificity* — a common property of real-world decision making tasks [Boutilier *et al.*, 1999]. This is the same type of representation used in the single agent case in Chapter 7. Specifically, we assume that the agents' value function can be decomposed into a set of *value rules*, each describing a context — an assignment to state variables and actions — and a value increment which gets added to the agents' total value in situations when that context applies. For example, a value rule might assert that in states where two agents are at the same house and both try to install the plumbing, they get in each other's way and the total value is decremented by 100.

Based on this representation, we provide a significant extension to the notion of a coordination graph. We again describe a distributed decision-making algorithm that uses message passing over this graph to reach a jointly optimal action. However, the coordination used in the algorithm can vary significantly from one situation to another. For example, if two agents are not in the same house, they will not need to coordinate. The coordination structure can also vary based on the utilities in the model; e.g., if it is dominant for one agent to work on the plumbing (e.g., because he is an expert), the other agents will not need to coordinate with him.

As in Chapter 7, we use context specificity in the factored MDP model, assuming that the rewards and the transition dynamics are rule-structured. We extend the linear programming approach in Chapter 7 to construct an approximate rule-based value function for multiagent factored MDPs. The agents can then use the coordination graph to decide on a joint action at each time step. Interestingly, although the value function is computed once in an offline setting, the online choice of action using the coordination graph gives rise to a highly variable coordination structure.

10.1 Representation

In order to exploit both additive and context-specific independence in multiagent problems, we must define a rule-based representation for multiagent factored MDPs. This extension is analogous to the rule-based version of single agent factored MDPs presented in Chapter 7. Thus, our presentation will be very concise.

In Chapter 8, we represent the transition model in a multiagent problem using a dynamic decision network (DDN). In this model, each node X'_i is associated with a conditional probability distribution (CPD) $P(X'_i \mid \mathsf{Parents}(X'_i))$, where the parents of variable X'_i in the graph include both state and agent variables, $\mathsf{Parents}(X'_i) \subseteq \{\mathbf{X}, \mathbf{A}\}$. In order to exploit context-specific independence, we represent each $P(X'_i \mid \mathsf{Parents}(X'_i))$ using a rule CPD as in Definition 7.1.3.

Similarly, we must decompose the reward function into rule functions: In our collaborative multiagent setting, each agent i is associated with a local reward function $R_i(\mathbf{x}, \mathbf{a})$ whose scope $\mathsf{Scope}[R_i(\mathbf{x}, \mathbf{a})]$ is restricted to depend on a small subset of the state variables, and on the actions of only a few agents. The global reward function $R(\mathbf{x}, \mathbf{a})$ is the sum of the rewards accrued by each agent $R(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{g} R_i(\mathbf{x}, \mathbf{a})$. In order to exploit context-specific independence in the reward function, we represent each $R_i(\mathbf{x}_\mathbf{a})$ using a rule-based function as in Definition 7.1.5.

Our approximation architecture uses basis functions h_j defined as rule-based functions. Using this representation, h_j can be written as $h_j(\mathbf{x}) = \sum_i \rho_i^{(h_j)}(\mathbf{x})$, where $\rho_i^{(h_j)}$ has the form $\left\langle \mathbf{c}_i^{(h_j)} : v_i^{(h_j)} \right\rangle$, *i.e.*, a function that takes value v_i if the current state is consistent with $\mathbf{c}_i^{(h_j)}$, and 0 otherwise. Using this definition, we can compute the backprojection of basis function h_j as:

$$g_j(\mathbf{x}, \mathbf{a}) = \sum_i \textsc{RuleBackproj}(\rho_i^{(h_j)}), \tag{10.1}$$

where $\textsc{RuleBackproj}(\rho_i^{(h_j)})$ is computed by applying the algorithm in Figure 7.2 using our rule-based representation for the multiagent DDN. Note that g_j is a sum of rule-based functions, and therefore also a rule-based function. For simplicity of notation, we use $g_j = \textsc{RuleBackproj}(h_j)$ to refer to this definition of backprojection.

Using this rule-based backprojection, we can now define a rule-based version of the local Q-function associated with each agent:

Definition 10.1.1 (rule-based local Q-function) *The* rule-based local Q-function *for agent i is given by:*

$$Q_i(\mathbf{x}, \mathbf{a}) = R_i(\mathbf{x}, \mathbf{a}) + \gamma \sum_{h_j \in \textit{Basis}[i]} w_j g_j(\mathbf{x}, \mathbf{a}), \tag{10.2}$$

where both the reward function $R_i(\mathbf{x}, \mathbf{a})$ and the basis functions h_j are rule-based functions, and the rule-based backprojection g_j of basis function h_j is defined in Equation (10.1).
∎

Our global Q-function approximation is then defined as a rule-based function $Q(\mathbf{x}, \mathbf{a}) = \sum_i Q_i(\mathbf{x}, \mathbf{a})$.

10.2 Context-specific coordination

As in Chapter 9, we begin by assuming that the basis function weights $\mathbf{w}$ are given and we are interested in computing the optimal greedy action that maximizes:

$$\arg\max_{\mathbf{a}} Q(\mathbf{x}, \mathbf{a}) = \arg\max_{\mathbf{a}} \sum_i Q_i(\mathbf{x}, \mathbf{a}),$$

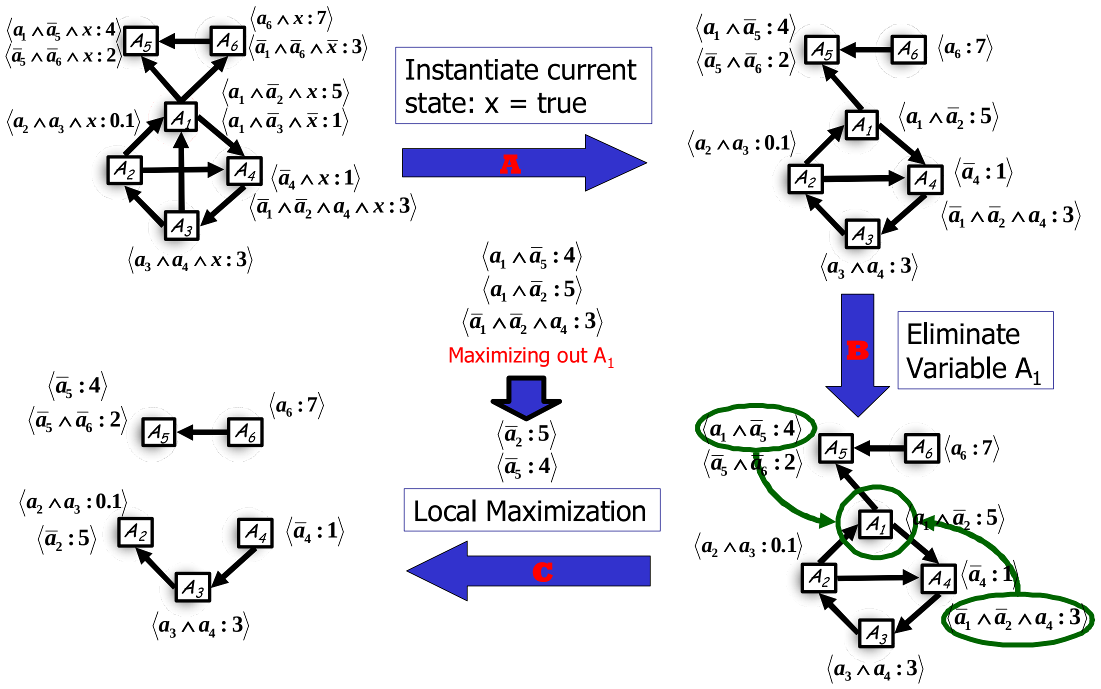

$\langle a_1 \wedge \bar{a}_5 \wedge x : 4\rangle$
$\langle \bar{a}_5 \wedge \bar{a}_6 \wedge x : 2\rangle$
$\langle a_6 \wedge x : 7\rangle$
$\langle \bar{a}_1 \wedge \bar{a}_6 \wedge \bar{x} : 3\rangle$
$\langle a_1 \wedge \bar{a}_2 \wedge x : 5\rangle$
$\langle a_1 \wedge \bar{a}_3 \wedge \bar{x} : 1\rangle$
$\langle a_2 \wedge a_3 \wedge x : 0.1\rangle$
$\langle \bar{a}_4 \wedge x : 1\rangle$
$\langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 \wedge x : 3\rangle$
$\langle a_3 \wedge a_4 \wedge x : 3\rangle$
A_1 A_2 A_3 A_4 A_5 A_6
Instantiate current state: x = true
A
$\langle a_1 \wedge \bar{a}_5 : 4\rangle$
$\langle \bar{a}_5 \wedge \bar{a}_6 : 2\rangle$
$\langle a_6 : 7\rangle$
$\langle a_1 \wedge \bar{a}_2 : 5\rangle$
$\langle a_2 \wedge a_3 : 0.1\rangle$
$\langle \bar{a}_4 : 1\rangle$
$\langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3\rangle$
$\langle a_3 \wedge a_4 : 3\rangle$
B
Eliminate Variable A_1
$\langle a_1 \wedge \bar{a}_5 : 4\rangle$
$\langle a_1 \wedge \bar{a}_2 : 5\rangle$
$\langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3\rangle$
Maximizing out A_1
$\langle \bar{a}_2 : 5\rangle$
$\langle \bar{a}_5 : 4\rangle$
Local Maximization
C
$\langle \bar{a}_5 : 4\rangle$
$\langle \bar{a}_5 \wedge \bar{a}_6 : 2\rangle$
$\langle a_6 : 7\rangle$
$\langle a_2 \wedge a_3 : 0.1\rangle$
$\langle \bar{a}_2 : 5\rangle$
$\langle \bar{a}_4 : 1\rangle$
$\langle a_3 \wedge a_4 : 3\rangle$

Figure 10.1: Example of variable coordination structure achieved by rule-based coordination graph, the rules in Q_j are indicated in the figure by the rules next to A_j. Clockwise from top-left: (a) initial coordination graph; (b) coordination graph for state $X = \mathit{true}$; (c) rules communicated to A_1; (d) coordination graph is simplified when A_1 is eliminated.

for the current state x.

In the previous chapter, the long-term utility, or Q-function is the sum of local Q-functions, associated with the "jurisdiction" of the different agents. For example, if multiple agents are constructing a house, we can decompose the value function as a sum of the values of the tasks accomplished by each agent. Thus, we specify the Q-function as a sum of agent-specific value functions Q_i, each with a restricted-scope. Each Q_i is typically represented as a table, listing agent i's local values for different combinations of variables in the scope. However, this representation is often highly redundant, forcing us to represent many irrelevant interactions. For example, an agent A_1's local Q-function might depend on the action of agent A_2 if both are trying to install the plumbing in the same house. However, there is no interaction if A_2 is currently working in another house, and there is no point in making A_1's entire local Q-function depend on A_2's action. Our rule-based representation of the local Q-function in Definition 10.1.1 allows us to represent exactly this type of context specific structure. A value rule in a local Q-function for our example could be:

$$\begin{aligned}\langle A_1\text{AtHouse} = A_2\text{AtHouse}\ \wedge& \\ & A_1 = \textit{plumbing} \wedge A_2 = \textit{plumbing} : -100\rangle.\end{aligned}$$

The rule-based local Q-function Q_i associated with agent i has the form:

$$Q_i = \sum_j \rho^i_j \,.$$

Note that if each rule ρ^i_j has scope $\mathbf{C}^i_j$, then Q_i will be a restricted-scope function of $\cup_j \mathbf{C}^i_j$. As in the previous chapter, the scope of Q_i can be further divided into two parts: The state variables

$$\textsf{Obs}[Q_i] = \{X_j \in \mathbf{X} \mid X_j \in \textsf{Scope}[Q_i]\}$$

are the observations agent i needs to make at each time step. The agent variables

$$\textsf{Agents}[Q_i] = \{A_j \in a \mid A_j \in \textsf{Scope}[Q_i]\}$$

are the agents with whom i interacts directly in the initialization of our coordination graph, as defined in Definition 9.1.1.

Example 10.2.1 *Consider a simple 6 agent example, where:*

$$\begin{array}{ll}
Q_1(\mathbf{x},\mathbf{a}) = \left\{\begin{array}{l}\langle a_1 \wedge \bar{a}_2 \wedge x : 5\rangle \\ \langle a_1 \wedge \bar{a}_3 \wedge \bar{x} : 1\rangle\end{array}\right. ; &
Q_4(\mathbf{x},\mathbf{a}) = \left\{\begin{array}{l}\langle \bar{a}_4 \wedge x : 1\rangle \\ \langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 \wedge x : 3\rangle\end{array}\right. ; \\
Q_2(\mathbf{x},\mathbf{a}) = \left\{\ \langle a_2 \wedge a_3 \wedge x : 0.1\rangle\right. ; &
Q_5(\mathbf{x},\mathbf{a}) = \left\{\begin{array}{l}\langle a_1 \wedge \bar{a}_5 \wedge x : 4\rangle \\ \langle \bar{a}_5 \wedge \bar{a}_6 \wedge x : 2\rangle\end{array}\right. ; \\
Q_3(\mathbf{x},\mathbf{a}) = \left\{\ \langle a_3 \wedge a_4 \wedge x : 3\rangle\right. ; &
Q_6(\mathbf{x},\mathbf{a}) = \left\{\begin{array}{l}\langle a_6 \wedge x : 7\rangle \\ \langle \bar{a}_1 \wedge \bar{a}_6 \wedge \bar{x} : 3\rangle\end{array}\right. .
\end{array}$$

The coordination graph for this example is shown in Figure 10.1(a). See, for example, that agent A_3 has the parent A_4, because A_4's action affects Q_3. ▮

Recall that, at every time step t, the agents' task is to coordinate in order to select the joint action $\mathbf{a}^{(t)}$ that maximizes $Q(\mathbf{x}^{(t)}, \mathbf{a}) = \sum_j Q_j(\mathbf{x}^{(t)}, \mathbf{a})$. If we apply the distributed action selection algorithm in Figure 9.4 in the previous chapter, the coordination structure would be always be the same. Surprisingly, as our example will illustrate, our simple rule-based representation of the Q-function will yield a coordination structure that will change with the state of the system, and even with the results of the local maximization performed by each agent.

Given a particular state $\mathbf{x}^{(t)} = \{x_1^{(t)}, \ldots, x_n^{(t)}\}$, agent i *instantiates* the current state on its local Q-function by discarding all rules in Q_i not consistent with the current state $\mathbf{x}^{(t)}$. Note that agent i only needs to observe the state variables in $\mathsf{Obs}[Q_i]$, and not the entire state of the system, substantially reducing the sensing requirements. Interestingly, after the agents observe the current state the coordination graph may become simpler:

Example 10.2.2 *Now consider the effect of observing the state $X = true$ on the rules in Example 10.2.1. Our instantiated Q-function $Q^x(\mathbf{a})$ now becomes:*

$$Q_1^x(\mathbf{a}) = \left\{ \langle a_1 \wedge \bar{a}_2 : 5 \rangle \right. ; \qquad Q_4^x(\mathbf{a}) = \left\{ \begin{array}{l} \langle \bar{a}_4 : 1 \rangle \\ \langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3 \rangle \end{array} \right. ;$$

$$Q_2^x(\mathbf{a}) = \left\{ \langle a_2 \wedge a_3 : 0.1 \rangle \right. ; \qquad Q_5^x(\mathbf{a}) = \left\{ \begin{array}{l} \langle a_1 \wedge \bar{a}_5 : 4 \rangle \\ \langle \bar{a}_5 \wedge \bar{a}_6 : 2 \rangle \end{array} \right. ;$$

$$Q_3^x(\mathbf{a}) = \left\{ \langle a_3 \wedge a_4 : 3 \rangle \right. ; \qquad Q_6^x(\mathbf{a}) = \left\{ \langle a_6 : 7 \rangle \right. .$$

Once we instantiate the current state, the coordination graph becomes simpler, as shown in Figure 10.1(b). See, for example, that agent A_6 is no longer a parent of agent A_1. Thus, agents A_1 and A_6 will only need to coordinate directly in the context of $X = \bar{x}$. ▮

After instantiating the current state $\mathbf{x}^{(t)}$, each $Q_i^{\mathbf{x}^{(t)}}$ will now only depend on the agents' action choices $\mathbf{a}$. Now, our task is to select a joint action $\mathbf{a}$ that maximizes $\sum_i Q_i^{\mathbf{x}^{(t)}}(\mathbf{a})$. Maximization in a graph with context-specific structure suggests the use of the rule-based version of variable elimination presented in Chapter 7. The only difference between this rule-based variable elimination algorithm and the table-based version presented in Figure 9.2 occurs in the maximization step. Here, we introduce a new function e, such that $e = \max_{a_l} f_l$. Instead of creating a table-based representation for e, we now generate a rule-based representation for this function by using the $\textsc{RuleMaxOut}(f, B)$ procedure presented in Figure 7.3. This procedure takes a rule-based function f and a variable B and returns a rule-based function g, such that $g = \max_b f$. Thus, we can compute the joint optimal greedy action for our multiagent system by substituting $e = \max_{a_l} f_l$, with $e = \textsc{RuleMaxOut}(f_l, A_l)$. The rest of the algorithm remains the same.

The cost of this algorithm is polynomial in the number of new rules generated in the maximization operation $\textsc{RuleMaxOut}(Q_l, A_l)$. The number of rules is never larger and in many cases exponentially smaller than the complexity bounds on the table-based coordination graph in the previous chapter, which, in turn, was exponential only in the *induced width* of this graph [Dechter, 1999]. However, the computational costs involved in managing sets of rules usually imply that the computational advantage of the rule-based approach will only manifest in problems that possess a fair amount of context-specific structure. When considering the distributed version of this algorithm, the rule-based representation has an additional advantage over the table-based one presented in the previous chapter: as we show in this section, the distributed rule-based approach may have significantly lower communication requirements.

Intuitively, the distributed algorithm, shown in Figure 10.2, follows very similar steps as the table-based one in the previous chapter. An individual agent "collect" value rules relevant to them from their children. The agent can then decide on its own conditional strategy, taking all of the implications into consideration. The choice of optimal action and the ensuing payoff will, of course, depend on the actions of agents whose strategies have not yet been decided. The agent then simply communicates the value ramifications of its strategy to other agents, so that they can make informed decisions on their own strategies.

Figure 10.2 presents the complete algorithm that will be executed by agent i. At every time step, the procedure follows 4 phases:

1. **Instantiation:** The agent makes local observations and instantiates the current state in its local Q-function by selecting the rules in Q_i consistent with the current state.

```
RULEBASEDDISTRIBUTEDACTIONSELECTION($i$)
// Distributed rule-based action selection algorithm for agent $i$.
    REPEAT EVERY TIME STEP $t$:
        // INSTANTIATION.
        // Instantiate the current state.
        OBSERVE THE VARIABLES $\text{OBS}[Q_i]$ IN THE CURRENT STATE $\mathbf{x}^{(t)}$.
        INSTANTIATE THE LOCAL Q-FUNCTION WITH THE CURRENT STATE BY SELECTING
          THE RULES IN $Q_i$ THAT ARE CONSISTENT WITH $\mathbf{x}^{(t)}$:
              $$Q_i^{\mathbf{x}^{(t)}}(\mathbf{a}) = Q_i(\mathbf{x}^{(t)}, \mathbf{a}).$$
        // Initialization.
        // Initialize the coordination graph.
        LET THE PARENTS OF $A_i$ BE THE AGENTS IN $\text{SCOPE}[Q_i^{\mathbf{x}^{(t)}}] = \text{AGENTS}[Q_i]$.
        STORE $Q_i^{\mathbf{x}^{(t)}}$.

        // Maximization.
        // Wait for signal from parent of $i$ in the variable elimination order.
        WAIT FOR SIGNAL FROM AGENT $\mathcal{O}_i^-$, IF $\mathcal{O}_i^- = \emptyset$ CONTINUE.
        // We can now compute the maximization for agent $i$.
        // First we collect the rules that depend on $A_i$, i.e., the ones stored by $i$, and the ones stored
          by the children of $i$ in the coordination graph whose context includes $A_i$.
        COLLECT THE LOCAL RULES $\rho_1, \ldots, \rho_L$ FROM THE CHILDREN OF $i$ IN THE COORDI-
          NATION GRAPH, WHOSE CONTEXT INCLUDES $A_i$, AND THE ONES STORED BY AGENT
          $i$.
        CACHE A NEW RULE-BASED FUNCTION $f_i = \sum_{j=1}^{L} \rho_j$; NOTE THAT $\text{SCOPE}[f_i] =$
          $\cup_{j=1}^{L} \text{SCOPE}[e_j]$.
        // Compute the local maximization for agent $i$.
        DEFINE A NEW FUNCTION $e = \text{RULEMAXOUT}(f_i, A_i)$, THE SCOPE OF $e$ IS
          $\text{SCOPE}[f_i] - \{A_i\}$.
        // Update the coordination graph.
        STORE EACH RULE $\rho_s$ IN THE NEW FUNCTION $e$ IN SOME AGENT $A_j \in \text{SCOPE}[\rho_s]$.
        DELETE $A_i$ FROM THE COORDINATION GRAPH, AND ADD EDGES FROM THE AGENTS
          IN $\text{SCOPE}[e]$ TO $A_j$.
        SIGNAL AGENT $\mathcal{O}_i^+$.

        // Action selection.
        // Wait for signal from child of $i$ in the variable elimination order.
        WAIT FOR SIGNAL FROM AGENT $\mathcal{O}_i^+$; IF $\mathcal{O}_i^+ = \emptyset$ INITIALIZE $\mathbf{a}^{(t)} = \emptyset$ AND CONTINUE.
        RECEIVE CURRENT ASSIGNMENT TO THE MAXIMIZING ACTION $\mathbf{a}^{(t)}$ FROM AGENT
          $\mathcal{O}_i^+$.
        // We can now compute the maximizing action for agent $i$.
        // Instantiate the maximization function corresponding to $A_i$ by selecting the rules in $f_i$
          whose context is consistent with the action choice thus far, i.e., $\mathbf{a}^{(t)}[\text{Scope}[f_i] - \{A_i\}]$.
        LET $f_i^*(a_i) = f_i(a_i, \mathbf{a}^{(t)}[\text{SCOPE}[f_i] - \{A_i\}])$, $\forall a_i \in A_i$.
        // Compute optimal assignment for $A_i$.
        LET $a_i^{(t)}$, THE ASSIGNMENT TO $A_i$ IN $\mathbf{a}^{(t)}$, BE $a_i^{(t)} = \arg\max_{a_i} f_i^*(a_i)$.
        // Signal to next agent.
        SIGNAL AGENT $\mathcal{O}_i^-$ AND TRANSMIT $\mathbf{a}^{(t)}$.
```

Figure 10.2: Synchronous distributed rule-based variable elimination algorithm on a coordination graph.

2. **Initialization:** The edges in the coordination graph are initialized, with agent i initially storing only the $Q_i^{\mathbf{x}}$ function.

3. **Maximization:** When it is agent i's turn to be eliminated, it collects the rules $\rho_1, \ldots, \rho_L$ whose scopes include A_i, *i.e.*, only the relevant ones out of those rules stored by the children of A_i in the coordination graph and those stored by agent i. These rules are combined into a new rule-based function $f_i = \sum_j \rho_j$, which is cached for the second pass of the algorithm. Agent i can now perform its local maximization by defining a new rule-based function $e = \text{RULEMAXOUT}(f_i, A_i)$, the scope of e is $\cup_{j=1}^{L} \mathsf{Scope}[\rho_j] - \{A_i\}$. As the scopes of all rules in this new function e do not contain A_i, each rule $\rho_s \in e$ should now be stored by some other agent j, such that, $A_j \in \mathsf{Scope}[\rho_s]$. At this point, agent i has been eliminated, *i.e.*, there are no functions whose scope includes A_i, and the coordination graph is updated accordingly.

4. **Action selection:** The optimal action choice can be computed by following the reverse order over agents. When it is agent i's turn, all agents later than i in the ordering have already computed their optimal action and stored it in $\mathbf{a}^*$. The scope of the cached rule-based function f_i only depends on A_i and on the actions of agents later in the ordering, whose optimal action has already been determined. Agent i can thus compute its optimal action choice a_i^* using a simple maximization over a_i.

The correctness of this distributed rule-based procedure is a corollary of Theorem 9.1.3 and of the correctness of rule-based variable elimination algorithm of Zhang and Poole [1999]:

Corollary 10.2.3 *For any ordering $\mathcal{O}$ over agents, if each agent executes the procedure in Figure 10.2, the agents will jointly compute the optimal greedy action $\mathbf{a}^{(t)}$ for each state $\mathbf{x}^{(t)}$, that is:*

$$\mathbf{a}^{(t)} \in \arg\max_{\mathbf{a}} \sum_{i=1}^{g} Q_i^{\mathbf{x}^{(t)}}(\mathbf{a}). \quad \blacksquare$$

Interestingly, the rule-based coordination structure exhibits several important properties. First, as we discussed, the structure often changes when instantiating the current state, as in Figure 10.1(b). Thus, in different states of the world, the agents may have to coordinate their actions differently. In our example, if the situation is such that the plumbing is ready to be installed, two qualified agents that are at the same house will need to coordinate. However, they may not need to coordinate in other situations.

The context-sensitivity of the rules also reduces communication between agents. In particular, agents only need to communicate relevant rules to each other, reducing unnecessary interaction. In the table-based version, when agent i performs its local maximization, it generates a new function f_i by summing up all the local functions that depend on A_i. In the rule-based version, we only need to collect the rules that depend on A_i. In this case, the scope, and thus the size, of f_i can be significantly smaller, as seen in our example:

Example 10.2.4 *When agent A_1 performs its local maximization, its children in the coordination graph transmit all rules whose scope includes A_1. Specifically, as shown in Figure 10.1(c), agent A_4 transmits $\langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3\rangle$ and agent A_5 transmits $\langle a_1 \wedge \bar{a}_5 : 4\rangle$. The local Q-function for agent A_1 becomes:*

$$Q_1^x(\mathbf{a}) = \begin{cases} \langle a_1 \wedge \bar{a}_2 : 5\rangle \\ \langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3\rangle \\ \langle a_1 \wedge \bar{a}_5 : 4\rangle \end{cases} .$$

Note that the scope of the rule-based Q_1^x is $\{A_1, A_2, A_4, A_5\}$. Had we used the table-based representation, the scope of Q_1^x would have been larger, i.e., $\{A_1, A_2, A_4, A_5, A_6\}$, as $Q_5^{\mathbf{x}}$ would include A_6 in its scope. ▮

More surprisingly, interactions that seem to hold between agents even after the state-based simplification and the limited communication of relevant rules can disappear as agents make strategy decisions. In the construction crew example, suppose electrical wiring and plumbing can be performed simultaneously. If there is an agent that can do both tasks and another that is only a plumber, then *a priori*agents need to coordinate so that they are not both working on plumbing. However, when the first agent is optimizing his strategy, he decides that electrical wiring is a dominant strategy, because either the other agent will do the plumbing and both tasks are done or the other agent will perform a different task, in which case the first agent can get to plumbing in the next time step, achieving the same total value. We can see this effect more precisely in our running example:

Example 10.2.5 *After collecting the relevant rules,the local Q-function for agent A_1 had become:*

$$Q_1^x(\mathbf{a}) = \begin{cases} \langle a_1 \wedge \bar{a}_2 : 5\rangle \\ \langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3\rangle \\ \langle a_1 \wedge \bar{a}_5 : 4\rangle \end{cases} .$$

As these are all the rules whose scope includes A_1, we can now perform the local maximization for this agent, which yields:

$$\textsc{RuleMaxOut}(Q_1^x, A_1) = \begin{cases} \langle \bar{a}_2 : 5\rangle \\ \langle \bar{a}_5 : 4\rangle \end{cases} .$$

The rule $\langle \bar{a}_1 \wedge \bar{a}_2 \wedge a_4 : 3\rangle$ disappeared, as $\langle a_1 \wedge \bar{a}_2 : 5\rangle$ dominates that rule for any assignment to A_4. Thus, A_1's optimal strategy is to do a_1 regardless.

In this example, there is an a priori *dependence between A_2, A_4 and A_5. However, after maximizing A_1, the dependence on A_4 disappears and agents A_4 and A_5 will no longer need to communicate, as shown in Figure 10.1(d).* ▮

Finally, we note that the rule structure provides substantial flexibility in constructing the system. In particular, the structure of the coordination graph can easily be adapted incrementally as new value rules are added or eliminated. For example, if it turns out that two agents intensely dislike each other, we can easily introduce an additional value rule that associates a negative value with pairs of action choices that puts them in the same house at the same time, thus forcing them to be in different houses. In the example in Figure 10.1(d), we may choose to remove the low-value rule $\langle a_2 \wedge a_3 : 0.1\rangle$, which will remove the communication requirement between A_2 and A_3, at the cost of some approximation in our action selection mechanism.

Therefore, by using the rule-based coordination graph, the coordination structure may change when:

- instantiating the current state;
- agents communicate relevant rules;
- an agent performs its local maximization;

- further approximating the value function by eliminating low-value rules.

10.3 Exploiting context-specific and additive structure in multiagent planning

Thus far, we have presented a representation for multiagent problems that can exploit both context-specific and additive independence. We have also described an algorithm for coordinating the agents actions given a rule-based approximation to the value function. It remains to show how such an approximation can be obtained. Fortunately, this approximation can be computed by a simple modification to the table-based multiagent factored LP-based approximation algorithm presented in Section 9.2. This algorithm, shown in Figure 9.5, relies on a call to our factored LP decomposition technique:

$$\textsc{FactoredLP}(\{\gamma g_1 - h_1, \ldots, \gamma g_k - h_k\}, R, \mathcal{O}).$$

This decomposition exploits additive structure in our model, but relies on a table-based representation. In order to exploit the context-specific structure in our rule-based representation, we should simply replace this procedure with the rule-based one described in Section 7.5.

10.4 Empirical evaluation

To verify the variable coordination property of our approach, we implemented our rule-based factored LP-based approximation algorithm, and the message passing coordination graph algorithm in C++, using again CPLEX as the LP solver. We experimented with a construction crew problem, where agents need to coordinate to build and maintain a set of houses. Each house has 5 features $\{$Foundation, Electric, Plumbing, Painting, Decoration$\}$. Each of these features is a state variable in our DDN. Each agent has a set of skills and some agents may move between houses. Each feature in the house requires two time steps to complete. Thus, in addition to the feature variables, the DDN for this problem contains

Prob.	♯houses	Agent skills	Agent location	♯states	♯actions	Time (min)
1	1	$A_1 \in \{\text{Found}, \text{Elec}, \text{Plumb}\}$; $A_2 \in \{\text{Plumb}, \text{Paint}, \text{Decor}\}$	House 1 House 1	2048	36	1.6
2	2	$A_1 \in \{\text{Paint}, \text{Decor}\}$ $A_2 \in \{\text{Found}, \text{Elec}, \text{Plumb}, \text{Paint}\}$ $A_3 \in \{\text{Found}, \text{Elec}\}$ $A_4 \in \{\text{Plumb}, \text{Decor}\}$	Moves House 1 Moves House 2	33,554,432	1024	33.7
3	3	$A_1 \in \{\text{Paint}, \text{Decor}\}$ $A_2 \in \{\text{Found}, \text{Elec}, \text{Plumb}\}$ $A_3 \in \{\text{Found}, \text{Elec}, \text{Plumb}, \text{Paint}\}$ $A_4 \in \{\text{Found}, \text{Elec}, \text{Plumb}, \text{Decor}\}$	Moves House 1 House 2 House 3	34,359,738,368	6144	63.9
4	2	$A_1 \in \{\text{Found}\}$ $A_2 \in \{\text{Decor}\}$ $A_3 \in \{\text{Found}, \text{Elec}, \text{Plumb}, \text{Paint}\}$ $A_4 \in \{\text{Elec}, \text{Plumb}, \text{Paint}\}$	Moves Moves House 1 House 2	8,388,608	768	5.7

Table 10.1: Summary of results of our rule-based multiagent factored planning algorithm on the building crew problem.

"action-in-progress" variables for each house feature, for each agent, *e.g.*, "A_1-Plumbing-in-progress-House1". Once an agent takes an action, the respective "action-in-progress" variable becomes true with high probability. If one of the "action-in-progress" variables for some house feature is true, that feature becomes true with high probability at the next time step. At every time step, with a small probability, a feature of the house may break, in which case there is a chain reaction and features that depend on the broken feature will break with probability 1. For example, if the plumbing breaks, the painting will peel in the next time step, and the decoration will be ruined in the following step. This effect makes the problem dynamic, incorporating both house construction and house maintenance in the same model. Agents receive 100 reward for each completed feature and -10 for each "action-in-progress". The discount factor is 0.95. We selected a simple bases: for each assignment to the variables corresponding to the parents of each house feature variable in the DDN, we introduced a rule-based basis function, whose context is exactly this assignment.

Table 10.1 summarizes the results for various settings. Note that, although the number of states may grow exponentially from one setting to the other, the running time grows polynomially. Furthermore, in Problem 2, the backprojections of the basis functions had scopes with up to 11 variables, too large for the table-based representation to be tractable. However, by using our rule-based representation, we can represent this backprojection very compactly.

Agent skills	*Actual* value of rule-based policy	Optimal value
$A_1 \in \{\text{Found}, \text{Elec}\}$; $A_2 \in \{\text{Plumb}, \text{Paint}, \text{Decor}\}$	6650	6653
$A_1 \in \{\text{Found}, \text{Elec}, \text{Plumb}\}$; $A_2 \in \{\text{Plumb}, \text{Paint}, \text{Decor}\}$	6653	6654

Table 10.2: Comparing the actual expected value of acting according to the rule-based policy obtained by our algorithm with the optimal policy, on the one house problem starting from the state with no features built in the house.

The policies generated in these problems are very intuitive. For example:

- In Problem 2, if we start with no features built, A_1 will go to House 2 and wait as its painting skills are going to be needed there before the decoration skills are needed in House 1.
- In Problem 1, we get very interesting coordination strategies: If the foundation is completed, A_1 will do the electrical fitting and A_2 will do the plumbing. Furthermore, A_1 makes its decision not by coordinating with A_2, but by noting that electrical fitting is a dominant strategy. On the other hand, if the system is at a state where both foundation and electrical fitting is done, then agents coordinate to avoid doing plumbing simultaneously.
- Another interesting feature of the policies occurs when agents are idle, *e.g.*, in Problem 1, if foundation, electric and plumbing are done, then agent A_1 repeatedly performs the foundation task (yielding a -10 reward at every time step). This action choice avoids a chain reaction starting from the foundation of the house. Checking the rewards, there is actually a higher expected loss from the chain reaction than the cost of repeatedly checking the foundation of the house.

For small problems with one house, we can compute the optimal policy exactly. In Table 10.2, we present the optimal values for two such problems. Additionally, we can compute the actual value of acting according to the policy generated by our method. As the table shows, these values are very close, indicating that the policies generated by our method are very close to optimal in these problems.

10.5 Discussion and related work

We provide a principled and efficient approach for planning in multiagent domains where the required interactions vary from one situation to another. We show that the task of finding an optimal joint action in our approach leads to a very natural communication pattern, where agents send messages along a *coordination* graph determined by the structure of the value rules, as in the previous chapter. However, the coordination structure now dynamically changes according to the state of the system, and even on the actual numerical values assigned to the value rules. Furthermore, the coordination graph can be adapted incrementally as the agents learn new rules or discard unimportant ones.

Our empirical evaluation shows that our methods scale to very complex problems, including problems where traditional table-based representations of the value function blow up exponentially. In problems where the optimal value could be computed analytically for comparison purposes, the value of the policies generated by our approach was within 0.05% of the optimal value. We also empirically observed the variable coordination properties of our approach. Our algorithm thus provides an effective method for acting in dynamic environments with a varying coordination structure.

From a representation perspective, the factored MDP model used in this chapter extends the rule-based representation described in Chapter 7 to the multiagent case. Boutilier [1996] suggests that the algorithms developed in Boutilier *et al.* [1995] can be extended to this collaborative multiagent case. The tradeoffs between our methods and those of Boutilier *et al.* have been discussed in detail in Section 7.8.1. In particular, their methods exploit only context-specific structure, while our approach can additionally exploit additive structure. On the other hand, their methods do not require basis functions to be defined *a priori*. We believe that, arguably, additive structure is even more important in multiagent systems. In our house building domain, for example, the interaction between agents with the same skill is context-specific, but the one between agents with different skills is probably better captured with an additive model.

Interestingly, Kok *et al.* [2003] applied our variable coordination graph to select the actions for a team of robots, where the weights of the rules were tuned by hand, rather than with our factored LP-based algorithm. Their team used this policy to win first place (out of 46 teams) in the 2003 RoboCup simulation league, winning all games, scoring a total of 177 goals with only 7 goals against them. Although the results of Kok *et al.* [2003] do not evaluate our planning algorithms, they show that our factored Q-function representation along with our variable coordination graph can capture very complex and effective policies.

We believe that this graph-based coordination mechanism will provide a well-founded schema for other multiagent collaboration and communication approaches in many environments, such as RoboCup, where the coordination structure must change over time.

Chapter 11

Coordinated reinforcement learning

In the previous chapters, we presented approaches that combine value function approximation with a message passing scheme by which multiple agents efficiently determine the jointly optimal action with respect to an approximate value function. We have also presented efficient planning algorithms for computing these approximate value functions, in multiagent settings.

Unfortunately, in many practical situations, a complete model of the environment, *i.e.*, of the transition probabilities, $P(\mathbf{x}' \mid \mathbf{x}, a)$ or of the reward function, $R(\mathbf{x}, a)$, is not know. Typically, there are two possible courses of action in such cases: to consult a domain expert who can provide an estimate of the model, or to estimate (learn) the model or a policy directly from data obtained from the real world. The latter process is called *reinforcement learning* (RL), as the agents are learning to act by responding to the reinforcement signals (rewards) they receive from the environment. For an in-depth presentation of the reinforcement learning problem and of some possible solution methods, we refer the reader to books on this topic by Sutton and Barto [1998] and by Bertsekas and Tsitsiklis [1996], and the review by Kaelbling *et al.* [1996].

At the high-level, there are two typical approaches to reinforcement learning. In a *model-based* approach [Moore & Atkeson, 1993; Kearns & Singh, 1998; Brafman & Tennenholtz, 2001], the environment is represented by a particular parametric model, and the agents learn the parameters of this model from their experience in this environment. This approximate model is then used to obtain approximate policies for this environment. Typically, the agents may choose to *explore* the environment further in order to improve the model, in the hope that an improved model will lead to a better policy in the future. Alternatively, the agents may choose to *exploit* what they have learned thus far, and select a policy that maximizes the reward with respect to this approximate model. Kearns and Singh [1998] present an algorithm that makes this choice between exploring and exploiting explicit. Brafman and Tennenholtz [2001] describe a simple algorithm that makes this choice implicitly, but that still leads to near-optimal policies in polynomial time (in the number of states). These methods tend to be effective, if an appropriate parameterization of the model is chosen.

An alternative is to choose a *model-free* approach [Sutton, 1988; Watkins, 1989; Williams, 1992], where no assumptions are made about a particular parametric model of the environment. Here, either the value function or the policy are parameterized. The agents then optimize these parameters directly from experience. Model-free approaches are often simpler, require less assumptions about the environment, and are often more easily combined with function approximation methods. Model-based approaches are often more stable and allow us to obtain more effective exploration strategies. Atkeson and Santamaria [1997] provide a more detailed discussion on the trade-off between these two approaches.

Most reinforcement learning methods have focused on single agent settings. Although some algorithms have been proposed for collaborative multiagent settings, these methods often do not directly consider interactions between agents in the parameterized solutions [Peshkin *et al.*, 2000], or use heuristic methods for combining values or rewards from different agents [Schneider *et al.*, 1999].

In this chapter, we show how our coordination graph action selection mechanism can be applied to the design of efficient reinforcement learning algorithms for collaborative multiagent problems, by building on existing single agent RL methods. We call our approach *coordinated reinforcement learning*, as structured coordination between agents is used both in the core of our learning algorithms and in our execution architectures. Interestingly, in the context of reinforcement learning, we will no longer require a factored model of the environment or even a discrete state space.

We begin by presenting two methods for computing an approximate value function through reinforcement learning in multiagent settings, building on two existing algorithms: Q-learning [Watkins, 1989; Watkins & Dayan, 1992] and Least Squares Policy Iteration (LSPI) [Lagoudakis & Parr, 2001]. We also demonstrate how parameterized value functions of the form acquired by our reinforcement learning variants can be combined in a very natural fashion with algorithms that attempt to optimize the policy directly, such as those of Williams [1992], Jaakkola *et al.* [1995], Sutton *et al.* [2000], Konda and Tsitsiklis [2000], Baxter and Bartlett [2000], Ng and Jordan [2000], and Shelton [2001]. The same communication and coordination structures used in the value function approximation phase are used in the policy search phase to sample from and update a factored stochastic policy function.

Our framework will approximate the global Q-function using the same type of factored Q-functions described in the previous chapters. Specifically, agent i is associated with a local Q-function $Q_i^{\mathbf{w}_i}$ that is parameterized by an independent set of parameters $\mathbf{w}_i$. The global Q-function is again given by:

$$Q^{\mathbf{w}}(\mathbf{x}, \mathbf{a}) = \sum_i Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a}).$$

In our RL setting, we will optimize the parameters $\mathbf{w}$ from the agents' experience with the environment. This experience is described by quadruples of the form (state, action, reward, next-state), which we will henceforth refer to as $(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, r^{(t)}, \mathbf{x}^{(t+1)})$, for the tth time step, where $r^{(t)} = R(\mathbf{x}^{(t)}, \mathbf{a}^{(t)})$ is the reward associated with the current state of the system. Note that, in general, the rewards may depend stochastically on the state and action. That is, at every visit the agents observe some reward sampled according to some unknown distribution. Furthermore, in this chapter, as the experience with the environment is represented by samples rather than a model, we no longer need to assume that the state is discrete. We can thus consider both discrete and continuous state spaces.

11.1 Coordination structure in Q-learning

Q-learning is a standard approach for solving an MDP through reinforcement learning [Watkins, 1989; Watkins & Dayan, 1992]. In Q-learning, the agent directly learns the values of state-action pairs from experience with the environment. The algorithm starts with some estimate with the Q-function. This estimated is then updated at each iteration using the following update rule:

$$Q(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}) \leftarrow Q(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}) + \alpha \left[r^{(t)} + \gamma \mathcal{V}(\mathbf{x}^{(t+1)}) - Q(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}) \right], \tag{11.1}$$

where $\alpha > 0$ is the "learning rate," or step size parameter, and

$$\mathcal{V}(\mathbf{x}^{(t+1)}) = \max_{\mathbf{a}} Q(\mathbf{x}^{(t+1)}, \mathbf{a}).$$

With a suitable decay schedule for the learning rate, a policy that ensures that every state-action pair is experienced infinitely often, and a representation for $Q(\mathbf{x}, \mathbf{a})$ which can assign an independent value to every state-action pair, Q-learning will converge to estimates for $Q(\mathbf{x}, \mathbf{a})$ which reflect the expected, discounted value of taking action $\mathbf{a}$ in state $\mathbf{x}$ and proceeding optimally thereafter, *i.e.*, the optimal Q-function [Watkins & Dayan, 1992].

In practice the formal convergence requirements for Q-learning almost never hold, because the state space is too large to permit an independent representation of the value of every state [Gordon, 2001]. Typically, a parametric function approximator such as a neural network is used to represent the Q-function for each action. The following gradient-based update scheme is often used to adapt Q-learning to this setting:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \alpha \left[r^{(t)} + \gamma \mathcal{V}(\mathbf{x}^{(t+1)}) - Q(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, \mathbf{w}^{(t)}) \right] \nabla_{\mathbf{w}} Q(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, \mathbf{w}^{(t)}), \tag{11.2}$$

where $\mathbf{w}$ is a weight vector for our function approximation architecture and, again, the value $\mathcal{V}(\mathbf{x}^{(t+1)})$ of the next state is again:

$$\mathcal{V}(\mathbf{x}^{(t+1)}) = \max_{\mathbf{a}} Q(\mathbf{x}^{(t+1)}, \mathbf{a}). \tag{11.3}$$

The Q-learning update mechanism is completely generic and requires only that the approximation architecture be differentiable. We are free to choose an architecture that is compatible with our action selection mechanism. Therefore, as described above, we can assume that every agent i maintains a local Q-function Q_i defined over some subset of the (possibly continuous) state variables, its own actions, and possibly actions of some other agents. The global Q-function is again a function of the global state $\mathbf{x}$ and the joint action vector $\mathbf{a}$:

$$Q^{\mathbf{w}}(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{g} Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a}),$$

where the dependence on $\mathbf{w}_i$ of $Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a})$ indicates the parametric (possibly non-linear) nature of our local Q-function representation. There are some somewhat subtle consequences of this representation. The first is that determining $\mathcal{V}(\mathbf{x}^{(t+1)})$ in Equation (11.3) requires a maximization over an exponentially-large action space that can be computed efficiently using the coordination graph procedure from Section 9.1 (or the rule-based version in Section 10.2).

The Q_i functions themselves can be maintained locally by each agent as an arbitrary, differentiable function of a set of local weights $\mathbf{w}_i$. Each Q_i can be defined over the entire state space, or just some subset of the state variables visible to agent i. It is important to note that the dependence on the state variables does not affect the complexity of our coordination graph algorithms, as the current state is always instantiated before these procedures are applied.

Once we have defined the local Q_i functions, we must compute the weight update in Equation (11.2). Each agent must compute:

$$\Delta(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, r^{(t)}, \mathbf{x}^{(t+1)}, \mathbf{w}^{(t)}) = \left[r^{(t)} + \gamma \mathcal{V}(\mathbf{x}^{(t+1)}) - Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}) \right], \tag{11.4}$$

the difference between the current Q-value and the discounted value of the next state. Thus, each agent needs access to $r^{(t)}$, $\mathcal{V}(\mathbf{x}^{(t+1)})$, and $Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)})$. Both the global reward $r^{(t)}$ and the Q value for the current state, $Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)})$, can be computed by a simple message passing scheme similar to the one in the coordination graph, by fixing the action of every agent to the one assigned in $\mathbf{a}^{(t)}$. A more elaborate process is required in order to compute $\mathcal{V}(\mathbf{x}^{(t+1)})$. However, as mentioned above, this term can be computed efficiently using our coordination graph maximization procedures.

Therefore, after the coordination step, each agent will have access to the value of $\Delta(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, r^{(t)}, \mathbf{x}^{(t+1)}, \mathbf{w}^{(t)})$. At this point, the weight update equation is entirely local:

```
COORDINATEDQLEARNING($Q$, $\mathbf{w}^{(0)}$ $\gamma$, $n$, $\alpha$, $\mathcal{O}$)
    // $Q = \{Q_1, \ldots, Q_g\}$ is the set of local Q-functions, each $Q_i$ is parameterized by $\mathbf{w}_i$.
    // $\mathbf{w}^{(0)}$ is the initial value for the parameters.
    // $\gamma$ is the discount factor.
    // $n$ is the number of iterations.
    // $\alpha = \{\alpha^{(0)}, \ldots, \alpha^{(n)}\}$ is the set of learning rates for each iteration.
    // $\mathcal{O}$ stores the elimination order.
    // Return the parameters of Q-function after $n$ iterations.
  FOR ITERATION $t = 0$ TO $n - 1$:
      // Observe the current transition.
      OBSERVE $(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, r^{(t)}, \mathbf{x}^{(t+1)})$.
      // Compute the action which maximizes the Q-function at the next state and its value using
        the variable elimination algorithm in Figure 9.2.
      LET
          $\left[\mathbf{a}^{(t+1)}, \mathcal{V}(\mathbf{x}^{(t+1)})\right] = \text{ARGVARIABLEELIMINATION}(Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t+1)}, \mathbf{a}),$
                                  $\mathcal{O}, \text{MAXOUT}, \text{ARGMAXOUT}),$
        WHERE $Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t+1)}, \mathbf{a}) = \{Q_1^{\mathbf{w}_1^{(t)}}(\mathbf{x}^{(t+1)}, \mathbf{a}), \ldots, Q_g^{\mathbf{w}_g^{(t)}}(\mathbf{x}^{(t+1)}, \mathbf{a})\}$.
      // Compute gradient for current state.
      COMPUTE THE GRADIENT $\nabla_{\mathbf{w}_i} Q_i^{\mathbf{w}_i^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)})$ FOR EACH LOCAL Q-FUNCTION $Q_i$.
      // Update parameters.
      UPDATE Q-FUNCTION PARAMETERS $\mathbf{w}_i$ FOR EACH LOCAL Q-FUNCTION $Q_i$ BY:
        $\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} + \alpha^{(t)} \left[r^{(t)} + \gamma\mathcal{V}(\mathbf{x}^{(t+1)}) - Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)})\right] \nabla_{\mathbf{w}_i} Q_i^{\mathbf{w}_i^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}),$
      // Take action $\mathbf{a}^{(t+1)}$ which maximizes $Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t+1)}, \mathbf{a})$. If an exploration policy is used,
        the action should be computed appropriately.
      EXECUTE ACTION $\mathbf{a}^{(t+1)}$.
  RETURN THE PARAMETERS $\mathbf{w}^{(n)}$.
```

Figure 11.1: Coordinated Q-learning algorithm.

$$\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} + \alpha\, \Delta(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}, r^{(t)}, \mathbf{x}^{(t+1)}, \mathbf{w}^{(t)}) \nabla_{\mathbf{w}_i} Q_i^{\mathbf{w}_i^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}). \tag{11.5}$$

Note that, in this equation, agent i only needs to compute the local gradient

$$\nabla_{\mathbf{w}_i} Q_i^{\mathbf{w}_i^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)}),$$

rather than the global gradient $\nabla_{\mathbf{w}} Q^{\mathbf{w}^{(t)}}(\mathbf{x}^{(t)}, \mathbf{a}^{(t)})$ used in Equation (11.2). The reason for this simplification is that the gradient decomposes linearly as no two local Q-functions Q_i and Q_j share parameters in $\mathbf{w}$. The locality of the weight updates in this formulation of Q-learning makes it very attractive for a distributed implementation. Each agent can maintain an entirely local Q-function and does not need to know anything about the structure of the neighboring agents' Q-functions. Different agents can even use different architectures, *e.g.*, one might use a neural network and another might use a CMAC [Albus, 1975]. The only requirement is that the joint Q-function be expressed as a sum of the these individual Q-functions.

Our complete multiagent extension of the Q-learning algorithm is shown in Figure 11.1. Note that Q-learning is usually implemented with an *exploration policy*: instead of always taking the action that maximizes the Q-function at every time step, the agent also takes actions that lead to rarely visited states. For examples of such policies and its effect on the convergence of Q-learning, see the book by Sutton and Barto [1998].

A negative aspect of this Q-learning formulation is that, like almost all forms of Q-learning with function approximation, it is difficult to provide any kind of formal convergence guarantees.

11.2 Multiagent LSPI

An often effective approach to RL is to optimize a value function or policy using a stored corpus of data, such methods are called *batch reinforcement learning* algorithms. Specifically, batch methods use a set of samples,

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{a}_i, \mathbf{x}'_i, r_i) |\, i = 1, 2, \ldots, L\},$$

collected from the environment to optimize a value function estimate or a policy. In this section, we propose a multiagent batch RL approach that builds on Least Squares policy iteration (LSPI) [Lagoudakis & Parr, 2001], a batch algorithm that uses the stored corpus of samples instead of a model to perform approximate policy iteration.

Given a policy π, the Q-function for this policy is again given by the following set of linear equations:

$$Q_\pi(\mathbf{x}, \mathbf{a}) = R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \mathcal{V}_\pi(\mathbf{x}'), \;\; \forall \mathbf{x} \in \mathbf{X}, \; \forall \mathbf{a} \in \mathbf{A}, \tag{11.6}$$

where $\mathcal{V}_\pi(\mathbf{x}) = Q_\pi(\mathbf{x}, \pi(\mathbf{x}))$. In matrix notation, we can express this fixed-point equation as:

$$Q_\pi = R + \gamma P \mathcal{V}_\pi, \tag{11.7}$$

where Q_π and R are $|\mathbf{X}| \cdot |\mathbf{A}|$ vectors, $\mathcal{V}$ is a $|\mathbf{X}|$ vector, and P is a $|\mathbf{X}| \cdot |\mathbf{A}| \times |\mathbf{X}|$ matrix.

LSPI approximates the Q-functions using a linear combination of basis functions (features). Specifically, given a policy π, the Q-function for this policy is approximated by:

$$\widehat{Q}^{\mathbf{w}}_\pi(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{k} w_i \phi_i(\mathbf{x}, \mathbf{a}) = \mathbf{w}^\mathsf{T} \phi(\mathbf{x}, \mathbf{a}), \tag{11.8}$$

where we use $\phi_i(\mathbf{x}, \mathbf{a})$ to denote a basis function used for approximating the Q-function, whose scope thus includes both state $\mathbf{X}$ and agent $\mathbf{A}$ variables. This notation differentiates this type of basis functions from our usual basis function $h_i(\mathbf{x})$, whose scope includes only state variables.

For a policy, $\pi^{(i)}$, associated with the ith iteration of the algorithm, LSPI computes an approximation to the Q-function, $\widehat{Q}_{\pi^{(i)}}$, satisfying the fixed-point conditions in Equation (11.6) with respect to our sample set. The new $\widehat{Q}_{\pi^{(i)}}$ then implicitly defines a greedy policy $\pi^{(i+1)}$:

$$\pi^{(i+1)}(\mathbf{x}) = \arg\max_{\mathbf{a}} \widehat{Q}_{\pi^{(i)}}(\mathbf{x}),$$

and the process is repeated until some form of convergence is achieved.

We briefly review the mathematical operations required for LSPI. For convenience we express our basis functions in matrix form:

$$\mathbf{\Phi} = \begin{pmatrix} \phi(\mathbf{x}_1, \mathbf{a}_1)^\intercal \\ \vdots \\ \phi(\mathbf{x}, \mathbf{a})^\intercal \\ \vdots \\ \phi(\mathbf{x}_{|\mathbf{X}|}, \mathbf{a}_{|\mathbf{A}|})^\intercal \end{pmatrix};$$

where $\mathbf{\Phi}$ is matrix $(|\mathbf{X}| \cdot |\mathbf{A}| \times k)$, where each row corresponds to a state-action pair, and each column corresponds to a basis function. We can represent our approximation of the Q-function by $\mathbf{\Phi w}$. Additionally, we can define a matrix $\mathbf{\Phi}_\pi$, with one row for each state $\mathbf{x}$, where the action choice for this row is the one specified by our policy, $\pi(\mathbf{x})$:

$$\mathbf{\Phi}_\pi = \begin{pmatrix} \phi(\mathbf{x}_1, \pi(\mathbf{x}_1))^\intercal \\ \vdots \\ \phi(\mathbf{x}, \pi(\mathbf{x}))^\intercal \\ \vdots \\ \phi(\mathbf{x}_{|\mathbf{X}|}, \pi(\mathbf{x}_{|\mathbf{X}|}))^\intercal \end{pmatrix},$$

note that $\mathbf{\Phi}_\pi$ is a $(|\mathbf{X}| \times k)$ matrix. Similarly, $\mathbf{\Phi}_\pi \mathbf{w}$ forms our approximation of $\mathcal{V}_\pi$, the value of policy π.

If we knew the transition matrix, P, and the reward function, R, we could, in principle, compute the weights of our Q-function approximation by solving a least-squares approximation to the fixed point in Equation (11.7):

$$\begin{aligned} Q_\pi &= R + \gamma P \mathcal{V}_\pi; \\ \mathbf{\Phi w} &\approx R + \gamma P \mathbf{\Phi}_\pi \mathbf{w}; \\ \mathbf{\Phi}^\intercal \mathbf{\Phi w} &\approx \mathbf{\Phi}^\intercal R + \gamma \mathbf{\Phi}^\intercal P \mathbf{\Phi}_\pi \mathbf{w}; \end{aligned}$$

Rearranging, we obtain the weights $\mathbf{w}$ by solving the following system of linear equations:

$$\mathbf{C w}_\pi = b, \tag{11.9}$$

where $\mathbf{C} = \mathbf{\Phi}^{\intercal}(\mathbf{\Phi} - \gamma P\mathbf{\Phi}_{\pi}) \quad \text{and} \quad b = \mathbf{\Phi}^{\intercal}R$.

Unfortunately, the matrices in Equation (11.7) contain entries for each state and action, and are thus exponentially-large. Furthermore, in our RL setting, we do not have models of R and P. However, we can use the samples in our corpus to construct approximate versions of the fixed point in Equation (11.9) by using an approximate version of $\mathbf{\Phi}$, $P\mathbf{\Phi}_{\pi}$, and R as follows:

$$\widehat{\mathbf{\Phi}} = \begin{pmatrix} \phi\left(\mathbf{x}_1, \mathbf{a}_1\right)^{\intercal} \\ \vdots \\ \phi\left(\mathbf{x}_i, \mathbf{a}_i\right)^{\intercal} \\ \vdots \\ \phi\left(\mathbf{x}_L, \mathbf{a}_L\right)^{\intercal} \end{pmatrix}; \quad \widehat{P\mathbf{\Phi}_{\pi}} = \begin{pmatrix} \phi\left(\mathbf{x}'_1, \pi(\mathbf{x}'_1)\right)^{\intercal} \\ \vdots \\ \phi\left(\mathbf{x}'_i, \pi(\mathbf{x}'_i)\right)^{\intercal} \\ \vdots \\ \phi\left(\mathbf{x}'_L, \pi(\mathbf{x}'_L)\right)^{\intercal} \end{pmatrix};$$

$$\widehat{R} = \begin{pmatrix} r_1 \\ \vdots \\ r_i \\ \vdots \\ r_L \end{pmatrix}.$$

Note that $\widehat{\mathbf{\Phi}}$ still has one column for each basis function, but now we only have one row for each sample, rather than a row for each state and action. Similarly, the vector $\widehat{R}$ contains one element per sample indicating the reward associated with this sample. Finally, $\widehat{P\mathbf{\Phi}_{\pi}}$ is an approximation of the backprojection of the basis functions. This matrix contains one row per sample with the value $\phi\left(\mathbf{x}'_i, \pi(\mathbf{x}'_i)\right)^{\intercal}$, which is an unbiased estimate of the backprojections, when action $\mathbf{a}_i$ is taken at state $\mathbf{x}_i$.

We can now compute the weights of our Q-function approximation for any policy π by solving a linear system of equations:

$$\widehat{\mathbf{\Phi}}^{\intercal}(\widehat{\mathbf{\Phi}} - \gamma\widehat{P\mathbf{\Phi}_{\pi}})\mathbf{w}_{\pi} = \widehat{\mathbf{\Phi}}^{\intercal}\widehat{R}. \tag{11.10}$$

The complete algorithm, starting from some initial set of weights $\mathbf{w}^{(0)}$ and some sample set $\mathcal{S}$, is given by repeating the following steps:

1. Generate the backprojection matrix, $\widehat{P\boldsymbol{\Phi}}_{\pi^{(t+1)}}$, for the greedy policy $\pi^{(t+1)}$ with respect to our previous Q-function estimate defined by the weights $\mathbf{w}^{(t)}$, where each row of $\widehat{P\boldsymbol{\Phi}}_{\pi^{(t+1)}}$ is given by:

$$\phi\left(\mathbf{x}'_i, \pi^{(t+1)}(\mathbf{x}'_i)\right)^\mathsf{T},$$

   and $\pi^{(t+1)}(\mathbf{x}'_i) = \arg\max_{\mathbf{a}} \widehat{Q}_\pi(\mathbf{x}'_i, \mathbf{a}, \mathbf{w}^{(t)})$.

2. Compute weights $\mathbf{w}^{(t+1)}$ of Q-function estimate for new policy $\pi^{(t+1)}$ by solving the linear system in Equation (11.10).

Lagoudakis and Parr [2001] present a simple incremental update rule for to generate the matrices $\widehat{\mathbf{C}}$ and $\widehat{b}$ directly. Thus, we never need to generate the intermediate matrices $\widehat{\boldsymbol{\Phi}}$, $\widehat{R}$, and $\widehat{P\boldsymbol{\Phi}}_\pi$. Specifically, assume that we initialize $\widehat{\mathbf{C}}^{(t)} = \mathbf{0}$ and $\widehat{b}^{(t)} = \mathbf{0}$. For policy $\pi^{(t)}$, each sample $(\mathbf{x}, \mathbf{a}, r, \mathbf{x}')$ contributes to the approximation according to the following update equation :

$$\widehat{\mathbf{C}}^{(t)} \leftarrow \widehat{\mathbf{C}}^{(t)} + \phi(\mathbf{x}, \mathbf{a})\Big(\phi(\mathbf{x}, \mathbf{a}) - \gamma\phi(\mathbf{x}', \pi^{(t)}(\mathbf{x}'))\Big)^\mathsf{T}, \qquad (11.11)$$

and $\widehat{b} \leftarrow \widehat{b} + r\phi(\mathbf{x}, \mathbf{a})$. We can then obtain the parameters for the Q-function estimate associated with this policy by solving the linear system $\widehat{\mathbf{C}}^{(t)}\mathbf{w} = \widehat{b}^{(t)}$.

Importantly, LSPI is able to reuse the same set of samples even as the policy changes. For example, suppose the corpus contains a tuple $(\mathbf{x}, \mathbf{a}_1, r, \mathbf{x}')$, *i.e.*, a transition from state $\mathbf{x}$ to state $\mathbf{x}'$ under action $\mathbf{a}_1$. Now suppose that the current policy $\pi^{(t)}$ takes action $\mathbf{a}_2$ at state $\mathbf{x}'$, *i.e.*, $\pi^{(t)}(\mathbf{x}') = \mathbf{a}_2$. This tuple is thus entered into the $\mathbf{C}^{(t)}$ matrix using Equation (11.11) as if a transition were made from $\phi(\mathbf{x}, \mathbf{a}_1)$ to $\phi(\mathbf{x}', \mathbf{a}_2)$:

$$\widehat{\mathbf{C}} \leftarrow \widehat{\mathbf{C}} + \phi(\mathbf{x}, \mathbf{a}_1)\Big(\phi(\mathbf{x}, \mathbf{a}_1) - \gamma\phi(\mathbf{x}', \mathbf{a}_2)\Big)^\mathsf{T}.$$

If $\pi^{(t+1)}(\mathbf{x}')$ changes the action for $\mathbf{x}'$ from $\mathbf{a}_2$ to $\mathbf{a}_3$, then the next iteration of LSPI enters a transition from $\phi(\mathbf{x}, \mathbf{a}_1)$ to $\phi(\mathbf{x}', \mathbf{a}_3)$ into the $\mathbf{C}$ matrix:

$$\widehat{\mathbf{C}} \leftarrow \widehat{\mathbf{C}} + \phi(\mathbf{x}, \mathbf{a}_1)\Big(\phi(\mathbf{x}, \mathbf{a}_1) - \gamma\phi(\mathbf{x}', \mathbf{a}_3)\Big)^\mathsf{T}.$$

The sample can be reused because the dynamics for state $\mathbf{x}$ under action $\mathbf{a}_1$ have not changed, only the greedy action defining $\mathcal{V}_\pi(\mathbf{x}') = Q(\mathbf{x}', \pi(\mathbf{x}'))$ has changed from $\mathbf{a}_2$ to $\mathbf{a}_3$.

Extending LSPI to a multiagent setting is surprisingly straightforward if we use our collaborative action selection mechanism, as shown in Figure 11.2. We first note that since LSPI is a linear method, any set of Q-functions produced by LSPI will, by construction, be of the right form for collaborative action selection. Each agent is assigned a local set of basis functions which define its local Q-function. The scope of these basis functions can be defined over the agent's own actions as well as the actions of a small number of other agents. As with ordinary LSPI, the current policy $\pi^{(t)}$ is defined implicitly by the current set of Q-functions, $\widehat{Q}_{\pi^{(t)}}$. However, in the multiagent case, we cannot enumerate each possible action to determine the policy in Step 1 of the algorithm, because this set of actions is exponential in the number of agents. Fortunately, we can again exploit the structure of the coordination graph to determine the optimal actions relative to $\widehat{Q}_{\pi^{(t)}}$: For each transition from state $\mathbf{x}$ to state $\mathbf{x}'$ under joint action $\mathbf{a}$ the coordination graph is used to determine the optimal greedy action $\mathbf{a}' = \pi(\mathbf{x}') = \arg\max_{\mathbf{a}} \widehat{Q}_{\pi^{(t)}}(\mathbf{x}', \mathbf{a})$ for $\mathbf{x}'$. The transition is added to the $\mathbf{C}$ matrix in Equation (11.11) as a transition from $Q(\mathbf{x}, \mathbf{a})$ to $Q(\mathbf{x}', \mathbf{a}')$.

A disadvantage of LSPI is that it is not currently amenable to a distributed implementation during the learning phase: The construction of the $\mathbf{C}$ matrix requires knowledge of the evaluation of each agent's basis functions for every state in the corpus, not only for every action that is actually taken by the agents, but also for the action selected by the each policy for the next time step. Thus, as most batch methods, multiagent LSPI is most useful as an offline technique for computing value function approximations from samples, without estimating a model of the environment.

11.3 Coordination in direct policy search

Value function-based reinforcement learning methods have recently come under some criticism as being unstable and difficult to use in practice [Gordon, 1999]. A function approximation architecture that is not well-suited to the problem can diverge or produce poor results with little meaningful feedback that is directly useful for modifying the function

```
MULTIAGENTLSPI($\mathbf{\Phi}$, $\mathbf{w}^{(0)}$, $\gamma$, $\mathcal{S}$, $T_{max}$, $\varepsilon$, $\mathcal{O}$)
        // $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_k\}$ is the set of basis functions.
        // $\mathbf{w}^{(0)}$ is the initial value for the weights.
        // $\gamma$ is the discount factor.
        // $\mathcal{S}$ is the sample set.
        // $T_{max}$ is the maximum number of iterations.
        // $\varepsilon$ is a precision parameter.
        // $\mathcal{O}$ stores the elimination order.
        // Return the weights for basis functions.
    LET ITERATION $t = 0$.
    REPEAT:
        // Initialization.
        LET $\widehat{\mathbf{C}} = \mathbf{0}$ AND $\widehat{b} = \mathbf{0}$.
        // Iterate over samples.
        FOR EACH $(\mathbf{x}_i, \mathbf{a}_i, \mathbf{x}'_i, r_i) \in \mathcal{S}$:
            // Compute the action assigned by the policy $\pi^{(t)}$ to the next state $\mathbf{x}'_i$, that is the action
              which maximizes the current Q-function $\arg\max_{\mathbf{a}} \sum_i w_i^{(t)} \phi(\mathbf{x}'_i, \mathbf{a})$ at the next state
              $\mathbf{x}'_i$ using the variable elimination algorithm in Figure 9.2.
            LET

                $\mathbf{a}' = $ ARGVARIABLEELIMINATION($\{w_1^{(t)}\phi_1(\mathbf{x}'_i, \mathbf{a}), \ldots, w_k^{(t)}\phi_k(\mathbf{x}'_i, \mathbf{a})\}$,
                                                  $\mathcal{O}$, MAXOUT, ARGMAXOUT),

            // Add this sample to $\mathbf{C}$ matrix and to the $b$ vector.
            LET $\widehat{\mathbf{C}} \leftarrow \widehat{\mathbf{C}} + \phi(\mathbf{x}_i, \mathbf{a}_i)\Big(\phi(\mathbf{x}_i, \mathbf{a}_i) - \gamma\phi(\mathbf{x}'_i, \mathbf{a}')\Big)^{\mathsf{T}}$.
            LET $\widehat{b} \leftarrow \widehat{b} + r_i\phi(\mathbf{x}_i, \mathbf{a}_i)$.
        // Compute new basis function weights $\mathbf{w}^{(t+1)}$.
        LET $\mathbf{w}^{(t+1)}$ BE THE SOLUTION TO THE LINEAR SET OF EQUATIONS: $\widehat{\mathbf{C}}\mathbf{w} = \widehat{b}$.
        LET $t = t + 1$.
    UNTIL $\left\|\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}\right\|_\infty \le \varepsilon$ OR $t = T_{max}$.
    RETURN THE WEIGHTS $\mathbf{w}^{(t)}$.
```

Figure 11.2: Multiagent LSPI algorithm.

approximator to achieve better performance.

LSPI was designed to address some of the concerns with Q-learning-based value function approximation. It is more stable than Q-learning and since it is a linear method, it is somewhat easier to debug. However, LSPI is still an approximate policy iteration procedure and can be quite sensitive to small errors in the estimated Q-values for policies [Bertsekas & Tsitsiklis, 1996]. In practice, LSPI takes large, coarse steps in policy space.

The shortcomings of value function-based methods have led to a surge of interest in direct policy search methods [Williams, 1992; Jaakkola *et al.*, 1995; Sutton *et al.*, 2000; Konda & Tsitsiklis, 2000; Baxter & Bartlett, 2000; Ng & Jordan, 2000; Shelton, 2001]. These methods use gradient ascent to search a space of parameterized stochastic policies. As with all gradient methods, local optima can be very problematic. Furthermore, gradient estimates are often very noisy and susceptible to plateaus, where the gradient magnitude is very small and becomes difficult to follow. Defining a relatively smooth but expressive policy space and finding reasonable starting points within this space are all important elements of any successful application of gradient ascent.

11.3.1 REINFORCE

In this section, we briefly review one of the simplest single agent policy search algorithms, REINFORCE[Williams, 1992]. We refer the reader to the presentation of Meuleau *et al.* [2001] for a more detailed derivation. The REINFORCE algorithm tackles *episodic problems*, where the agent start from an initial state $\mathbf{x}^{(0)}$ distributed according to $P\left(\mathbf{x}^{(0)}\right)$, and collects rewards for τ_{max} steps. Then a new starting state is sampled, and the process is repeated. In the episodic formulation, the expected value $V_{\rho^{\mathbf{w}}}$ of a stochastic policy ρ parameterized by $\mathbf{w}$ is given by:

$$V_{\rho^{\mathbf{w}}} = \mathbf{E}_{\rho^{\mathbf{w}}}\left[\sum_{t=0}^{\tau_{max}} \gamma^t R\left(\mathbf{X}^{(t)}, \mathbf{A}^{(t)}\right)\right]. \tag{11.12}$$

If we differentiate the value in Equation (11.12) with respect to a particular policy parameter $w \in \mathbf{w}$, we obtain:

$$\frac{\partial}{\partial w} V_{\rho^{\mathbf{w}}} = \mathbf{E}_{\rho^{\mathbf{w}}} \left[\sum_{t=0}^{\tau_{max}} \gamma^t R\left(\mathbf{X}^{(t)}, \mathbf{A}^{(t)}\right) \left(\sum_{t'=0}^{t} \frac{\partial}{\partial w} \ln \rho^{\mathbf{w}} \left(\mathbf{A}^{(t')} \mid \mathbf{X}^{(t')} \right) \right) \right]. \tag{11.13}$$

Unfortunately, the derivative in Equation (11.13) requires us to compute a sum over all possible assignments to the state and agent variables for the whole episode, an exponentially-large summation. The REINFORCE algorithm addresses this issue by using L sampled episodes:

$$\mathcal{S} = \left\{ \left(\mathbf{x}_1^{(0-t)}, \mathbf{a}_1^{(0-t)}, r_1^{(0-t)} \right), \ldots, \left(\mathbf{x}_L^{(0-t)}, \mathbf{a}_L^{(0-t)}, r_L^{(0-t)} \right) \right\}, \tag{11.14}$$

where $\left(\mathbf{x}_i^{(0-t)}, \mathbf{a}_i^{(0-t)}, r_i^{(0-t)} \right)$ denotes, respectively, the states visited from time 0 to t, the actions taken, and the rewards accrued. Using these samples, we obtain an unbiased estimate of the derivative:

$$\widehat{\frac{\partial}{\partial w} V_{\rho^{\mathbf{w}}}} = \frac{1}{L} \sum_{i=1}^{L} \sum_{t=0}^{\tau_{max}} \gamma^t r_i^{(t)} \left[\sum_{t'=0}^{t} \frac{\partial}{\partial w} \ln \rho^{\mathbf{w}} \left(\mathbf{a}_i^{(t')} \mid \mathbf{x}_i^{(t')} \right) \right]. \tag{11.15}$$

We thus need two operations in order to compute the gradient using the REINFORCE algorithm: a method to sample from our stochastic policy, in order to collect the samples, and an efficient algorithm for computing $\frac{\partial}{\partial w} \ln \rho^{\mathbf{w}} \left(\mathbf{a}_i^{(t')} \mid \mathbf{x}_i^{(t')} \right)$, the partial derivative of the policy, for some fixed state and action choice.

11.3.2 Multiagent factored soft-max policy

In order to extend the reinforce algorithm to collaborative multiagent settings, we must first choose an appropriate policy parameterization. We now show how to seed a gradient ascent procedure with a multiagent policy generated by Q-learning or LSPI as described above. Of course, the methods presented in this section also apply for policy search methods that start from any initial policy estimate.

To guarantee that the gradient is well-defined, policy search methods require us to use stochastic policies (see definition in Chapter 2). Our first task is to convert the deterministic policy implied by our approximate Q-function into a stochastic policy, $\rho(\mathbf{a}|\mathbf{x})$, *i.e.*, a distribution over actions given the state. A natural way to do this, which also turns out to be compatible with most policy search methods, is to create a soft-max policy over the Q-values:

Definition 11.3.1 (soft-max policy) *Let the* soft-max policy $\mathit{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})$ *associated with the local Q-functions* $Q^{\mathbf{w}} = \{Q_1^{\mathbf{w}_1}(\mathbf{x}, \mathbf{a}), \dots, Q_g^{\mathbf{w}_g}(\mathbf{x}, \mathbf{a})\}$ *be defined as:*

$$\mathit{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}}) = \frac{e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{a})}}{\sum_{\mathbf{b}} e^{\frac{1}{T}\sum_k Q_k^{\mathbf{w}_k}(\mathbf{x},\mathbf{b})}}; \tag{11.16}$$

where T is a temperature parameter *indicating how stochastic we want to make the initial policy.* ∎

Note that once we optimize the $Q_j^{\mathbf{w}_j}$ in the policy description, they no longer form an estimate of the Q-function. We now view them simply as a parameterization of the policy.

To be able to apply policy search methods for such a policy representation, we must address two additional issues: First, to act according to our policy $\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})$, agents must coordinate to sample an action according to the soft-max distribution in Equation (11.16). Second, for gradient ascent purposes, we need an efficient method for computing the derivative of our stochastic policy $\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})$ with respect to the parameters $\mathbf{w}$.

11.3.3 Sampling from a multiagent soft-max policy

The standard approach for sampling from a soft-max policy at some state $\mathbf{x}$ is to compute the value of the numerator for each action $\mathbf{a}$. These values are then normalized, and an action is chosen at random according to these normalized values. Sampling from our multiagent soft-max policy may appear problematic, because the size of the joint action space makes such action enumeration procedure intractable. Fortunately, we can again use a variable elimination-style algorithm on our coordination graph to sample our multiagent policy.

Instantiating the current state $\mathbf{x}$ into Q is again easy: each agent needs to observe only the variables in $\mathsf{Obs}[Q_j]$ and instantiate each Q_j, as $Q_j^{\mathbf{x}}$, appropriately. At this point, we need to generate a sample from a soft-max of $Q_j^{\mathbf{x}}$ functions that depend only on the action choice. In order to illustrate the general sampling procedure [Cowell *et al.*, 1999], we use an example that follows the same structure as the one we used for action selection in Example 9.1.2:

Example 11.3.2 *Following our earlier example, our task is now to sample from the potential corresponding to the numerator of* $\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})$. *Suppose, for example, that, after instantiating the state* $\mathbf{x}$, *the individual agent's Q-functions have the following form:*

$$Q^{\mathbf{x}} = Q_1(a_1, a_2) + Q_2(a_2, a_4) + Q_3(a_1, a_3) + Q_4(a_3, a_4),$$

and that we wish to sample from the potential function for

$$e^{\sum_j Q_j^{\mathbf{x}}(\mathbf{a})} = e^{Q_1(a_1,a_2)} e^{Q_2(a_2,a_4)} e^{Q_3(a_1,a_3)} e^{Q_4(a_3,a_4)}.$$

To sample actions one at a time, we will follow a strategy of marginalizing out actions until we are left with a potential over a single action. We then sample from this potential and propagate the results backwards to sample actions for the remaining agents.

Suppose we begin by eliminating A_4. *Agent 4 can summarize its impact on the rest of the distribution by combining its potential function with that of agent 2 and defining a new potential:*

$$f_4(a_2, a_3) = \sum_{a_4} e^{Q_2(a_2,a_4)} e^{Q_4(a_3,a_4)}.$$

The problem now reduces to sampling from

$$e^{Q_1(a_1,a_2)} e^{Q_3(a_1,a_3)} f_4(a_2, a_3),$$

having one fewer agent. Next, agent 3 communicates its contribution giving:

$$f_3(a_1, a_2) = \sum_{a_3} e^{Q_3(a_1,a_3)} f_4(a_2, a_3).$$

```
SUMOUT ($\mathcal{E}, A_l$)
        // $\mathcal{E} = \{e_1, \ldots, e_m\}$ is the set of functions.
        // $A_l$ variable to be summed out.
    LET $f = \prod_{j=1}^{L} e_j$.
    IF $A_l = \emptyset$:
        LET $e = f$.
    ELSE:
        DEFINE A NEW FUNCTION $e = \sum_{a_l} f$; NOTE THAT
         $\text{SCOPE}[e] = \cup_{j=1}^{L} \text{SCOPE}[e_j] - \{X_l\}$.
    RETURN $e$.
```

Figure 11.3: SUMOUT operator for variable elimination, procedure that sums out a variable A_l from functions $\prod_i e_i$.

Agent 2 now communicates its contribution, giving

$$f_2(a_1) = \sum_{a_2} e^{Q_1(a_1,a_2)} f_3(a_1, a_2).$$

Agent 1 can now sample its action from the potential $P(a_1) \propto f_2(a_1)$. Let us denote this sample by a_1^*.

We can now sample actions for the remaining agents by reversing the direction of the messages and sampling from the distribution for each agent, conditioned on the choices of the previous agents. For example, when agent 2 is informed of the action selected by agent 1, agent 2 can sample actions from the distribution:

$$P(a_2|a_1^*) \propto e^{Q_1(a_1^*,a_2)} f_3(a_1^*, a_2).$$

After agent 2 samples action a_2^*, agent 3 can sample from:

$$P(a_3|a_1^*, a_2^*) \propto e^{Q_3(a_1^*,a_3)} f_4(a_2^*, a_3).$$

Finally, after agent 3 samples action a_3^*, agent 4 can sample its action a_4^* according to:

```
SAMPLEOUT $(\mathcal{E}, A_l)$
    // $\mathcal{E} = \{e_1, \ldots, e_m\}$ is the set of functions that depend only on $A_l$.
    // $A_l$ variable to be sampled.
  RETURN A SAMPLE $a_l^*$ DISTRIBUTED PROPORTIONALLY TO $\prod_{j=1}^{L} e_j$.
```

Figure 11.4: SAMPLEOUT operator for variable elimination, procedure that returns a sample of the variable A_l distributed according to $\prod_i e_i$.

$$P(a_4|a_1^*, a_2^*, a_3^*) \propto e^{Q_2(a_2^*,a_4)} e^{Q_4(a_3^*,a_4)}. \quad \blacksquare$$

The general algorithm has the same message passing topology as our original action selection mechanism. The only difference is the content of the messages: The forward pass messages are probability potentials and the backward pass messages are used to compute conditional distributions from which actions are sampled. The generic variable elimination algorithm in Figure 9.2 can be used to obtain a centralized version of this algorithm, all we need to do is use different operators, as in the example above. Specifically, we substitute the ELIMOPERATOR with SUMOUT from Figure 11.3, and ARGOPERATOR with SAMPLEOUT from Figure 11.4. The distributed version is analogous to the one in Figure 9.4. The correctness of this approach is guaranteed by the correctness of variable elimination:

Theorem 11.3.3 *For any ordering $\mathcal{O}$ on the variables, the* ARGVARIABLEELIMINATION *in Figure 9.2 procedure produces samples from the soft-max policy:*

$$\begin{aligned} &\text{ARGVARIABLEELIMINATION}(\{e^{\frac{1}{T}Q_1^{\mathbf{x}}}, \ldots, e^{\frac{1}{T}Q_g^{\mathbf{x}}}\}, \mathcal{O}, \text{ SUMOUT}, \text{ SAMPLEOUT}) \\ &\qquad\qquad \sim \frac{e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{a})}}{\sum_{\mathbf{b}} e^{\frac{1}{T}\sum_k Q_k^{\mathbf{w}_k}(\mathbf{x},\mathbf{b})}}, \\ &\qquad\qquad = \mathit{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}}), \end{aligned}$$

for each state $\mathbf{x}$.

Proof: *see for example the book by Lauritzen and Spiegelhalter [1988].* ∎

As with the basic variable elimination procedure in Section 4.2, the cost of this sampling algorithm is linear in the number of new “function values” introduced, or in our multiagent coordination case, only exponential in the *induced width* of the coordination graph.

11.3.4 Gradient of a multiagent policy

The next key operation in our multiagent policy search framework is the computation of the gradient of a multiagent soft-max policy function, a key operation in a REINFORCE style [Williams, 1992] policy search algorithm.[1]

First, recall that the global Q-function is the sum of the local Q_j-functions:

$$Q^{\mathbf{w}}(\mathbf{x},\mathbf{a}) = \sum_{j=1}^{g} Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{a}),$$

and our soft-max policy is given by:

$$\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}}) = \frac{e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{a})}}{\sum_{\mathbf{b}} e^{\frac{1}{T}\sum_k Q_k^{\mathbf{w}_k}(\mathbf{x},\mathbf{b})}}.$$

As discussed in Section 11.3.1, most policy search approaches require us to compute the gradient of the log of the stochastic policy. Consider the derivative of the log of our soft-max policy with respect to a particular parameter $w_i \in \mathbf{w}_i$ of agent i's local Q-function:

$$\begin{aligned}\frac{\partial}{\partial w_i}\ln\left[\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})\right] &= \frac{\partial}{\partial w_i}\ln\left(\frac{e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{a})}}{\sum_{\mathbf{b}} e^{\frac{1}{T}\sum_k Q_k^{\mathbf{w}_k}(\mathbf{x},\mathbf{b})}}\right); \\ &= \frac{\partial}{\partial w_i}\ln e^{\frac{1}{T}\sum_j Q_k^{\mathbf{w}_k}(\mathbf{x},\mathbf{a})} - \frac{\partial}{\partial w_i}\ln\sum_{\mathbf{b}} e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{b})}.\end{aligned} \tag{11.17}$$

Using the fact that $\frac{\partial}{\partial w_i}\ln f = \frac{\frac{\partial}{\partial w_i} f}{f}$, Equation (11.17) becomes:

$$\frac{\partial}{\partial w_i}\ln\left[\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})\right] = \frac{\partial}{\partial w_i}\frac{1}{T}\sum_k Q_k^{\mathbf{w}_k}(\mathbf{x},\mathbf{a}) - \frac{\sum_{\mathbf{b}}\frac{\partial}{\partial w_i} e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{b})}}{\sum_{\mathbf{b}'} e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{b}')}}. \tag{11.18}$$

[1]Most policy search algorithms are of this style.

```
MULTIAGENTPOLICYDERIVATIVE ($Q^{\mathbf{w}}$, $T$, $\mathbf{a}^*$, $i$,$w_i$,$Z(\mathbf{x})$,$\mathcal{O}$)
        // $Q^{\mathbf{w}} = \{Q_1^{\mathbf{w}_1}, \ldots, Q_g^{\mathbf{w}_g}\}$ is the set of local Q-functions.
        // $T$ is the temperature parameter.
        // $\mathbf{a}^*$ is the current action.
        // $i$ is the agent we are considering.
        // $w_i$ is the parameter we are differentiating.
        // $Z(\mathbf{x})$ is the partition function $Z(\mathbf{x}) = \sum_{\mathbf{b}} e^{\frac{1}{T}\sum_j Q_j(\mathbf{x},\mathbf{b},\mathbf{w}_j)}$ computed at state $\mathbf{x}$: .
        // $\mathcal{O}$ stores the elimination order.
        // Return the derivative $\frac{\partial}{\partial w_i} \ln \left[\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})\right]$ computed at action $\mathbf{a}^*$.
    // Collect set of functions to be summed to compute numerator of the second term in the righthand
      side of Equation (11.19).
    LET $\mathcal{F} = \{e^{\frac{1}{T}Q_1^{\mathbf{w}_1}(\mathbf{x},\mathbf{b})}, \ldots, e^{\frac{1}{T}Q_g^{\mathbf{w}_g}(\mathbf{x},\mathbf{b})}, \frac{1}{T}\frac{\partial}{\partial w_i}Q_i^{\mathbf{w}_i}(\mathbf{x},\mathbf{b})\}$.
    LET $Num$ = VARIABLEELIMINATION($\mathcal{F}$,$\mathcal{O}$, SUMOUT).
    // We can now compute the desired derivative.
    LET $\delta(\mathbf{a}) = \frac{1}{T}\frac{\partial}{\partial w_i}Q_i(\mathbf{x},\mathbf{a},\mathbf{w}_i) - \frac{Num}{Z(\mathbf{x})}$.
    RETURN DERIVATIVE $\delta(\mathbf{a}^*)$.
```

Figure 11.5: Procedure for computing the derivative of the log of our multiagent soft-max policy: $\frac{\partial}{\partial w_i} \ln \left[\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})\right]$, computed at action $\mathbf{a}^*$.

Using the fact that $\frac{\partial}{\partial w_i}e^f = e^f \frac{\partial f}{\partial w_i}$, and the linearity of derivatives, Equation (11.18) becomes:

$$\frac{\partial}{\partial w_i} \ln \left[\mathsf{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})\right] = \frac{1}{T}\frac{\partial}{\partial w_i}Q_i^{\mathbf{w}_i}(\mathbf{x},\mathbf{a}) - \frac{\sum_{\mathbf{b}} e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{b})} \frac{1}{T}\frac{\partial}{\partial w_i}Q_i^{\mathbf{w}_i}(\mathbf{x},\mathbf{b})}{\sum_{\mathbf{b}'} e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{b}')}}. \tag{11.19}$$

The first term in the righthand side of Equation (11.19) is just the local derivative of the agent's local Q-function. The denominator of the second term is the partition function of our multiagent soft-max policy:

$$Z(\mathbf{x}) = \sum_{\mathbf{b}'} e^{\frac{1}{T}\sum_j Q_j^{\mathbf{w}_j}(\mathbf{x},\mathbf{b}')}, \tag{11.20}$$

computed at state $\mathbf{x}$. We obtain the partition function as a side product of our efficient sampling algorithm using the variable elimination algorithm in Figure 9.2.

Therefore, the only term that remains to be computed is the numerator of the second term in the righthand side of Equation (11.19). We can again use a variable elimination procedure to compute this term. Specifically, this numerator can be rewritten as:

$$\sum_{\mathbf{a}} \frac{1}{T} \frac{\partial}{\partial w_i} Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a}) \prod_j e^{\frac{1}{T} Q_j^{\mathbf{w}_j}(\mathbf{x}, \mathbf{a})}. \tag{11.21}$$

Note that the term inside the sum is the product of restricted-scope functions: the product of $\frac{\partial}{\partial w_i} Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a})$, whose scope is $\mathsf{Scope}[Q_i]$, with each $e^{\frac{1}{T} Q_j^{\mathbf{w}_j}(\mathbf{x}, \mathbf{a})}$, whose scope is $\mathsf{Scope}[Q_j]$. Thus, computing the numerator in Equation (11.21) is equivalent to computing the sum over all action of a product of functions, which is exactly a partition function. This task can again be performed efficiently using variable elimination, analogously to the sampling method that relies on variable elimination.

Figure 11.5 shows the complete algorithm for computing the derivative of the log of our multiagent soft-max policy with respect to a particular parameter w_i. The derivation in this section proves the correctness of this procedure:

Theorem 11.3.4 *For any ordering $\mathcal{O}$ on the variables, the* MULTIAGENTPOLICYDERIVATIVE *procedure computes the derivative of the log of the soft-max policy with respect to parameter $w_i \in \mathbf{w}_i$ of agent's i local Q-function:*

$$\begin{aligned} \textsc{MultiagentPolicyDerivative}(Q^{\mathbf{x}}, T, \mathbf{a}^*, i, w_i, Z(\mathbf{x}), \mathcal{O}) \quad = \\ \frac{\partial}{\partial w_i} \ln\left[\textit{SoftMax}(\mathbf{a} \mid \mathbf{x}, Q^{\mathbf{w}})\right], \end{aligned}$$

computed at action $\mathbf{a}^*$, for each state $\mathbf{x}$, where the partition function $Z(\mathbf{x})$ is defined in Equation (11.20). ▮

If we want to compute the derivative of the log of the policy with respect to every parameter $w \in \mathbf{w}$ using the algorithm in Figure 11.5, we would be applying variable elimination once for each parameter. However, by using the clique tree algorithm [Lauritzen & Spiegelhalter, 1988], it is possible to compute all of these derivatives in time equivalent to about two passes of variable elimination. Specifically, we would start by building a clique tree representation for our soft-max policy, conditional on the current state $\mathbf{x}$. Now note that we can interpret the second term in the righthand side of Equation (11.19) as the expectation of $\frac{1}{T}\frac{\partial}{\partial w_i}Q_i(\mathbf{x}, \mathbf{a}, \mathbf{w}_i)$ with respect to our soft-max policy. Given a clique tree, this expectation can be computed efficiently by just using the calibrated potential in a clique that includes the agent variables in $\mathsf{Agents}[Q_i]$, without any further variable elimination steps.

11.3.5 Multiagent REINFORCE

In the previous sections, we presented efficient algorithms for sampling and computing the gradient of a multiagent soft-max policy. We can now revisit the REINFORCE, described in Section 11.3.1, to obtain a new collaborative multiagent policy search algorithm, where the policy represents explicit correlations between the actions of our agents.

In Figure 11.5, we present an efficient algorithm for computing the derivative of the log of our multiagent soft-max policy. We can now use this algorithm to compute an REINFORCE-style approximation to the gradient of the value of our multiagent policy using the formulation in Equation (11.15). Using this estimate of the gradient, we can use any of the standard gradient ascent procedures to optimize the parameters of our multiagent soft-max policy.

We have presented a centralized version of our policy search algorithm. As in the case of Q-learning, a global error signal must be shared by the entire set of agents in a distributed implementation. Apart from this, the gradient computations and stochastic policy sampling procedures involve a message passing scheme with the same topology as the action selection mechanism. We believe that these methods can be incorporated into any of a number of policy search methods to fine tune a policy derived by a value function method, such as Q-learning or by LSPI.

11.4 Empirical evaluation

We validated our coordinated RL approach on two domains: multiagent SysAdmin and power grid [Schneider *et al.*, 1999].

We first evaluated our multiagent LSPI algorithm on the multiagent SysAdmin problem

```
MULTIAGENTREINFORCE ($Q$, $\mathbf{w}$, $T$, $L$, $\tau_{max}$, $\mathcal{O}$)
    // $Q = \{Q_1, \ldots, Q_g\}$ is the set of local Q-functions parameterized by $\mathbf{w}$.
    // $\mathbf{w}$ is the current value of the parameters.
    // $T$ is the temperature parameter.
    // $L$ is the number of trajectories.
    // Return an unbiased estimate of the gradient of our multiagent soft-max policy:

        $$\nabla_{\mathbf{w}} V_{\text{SoftMax}(\mathbf{a}|\mathbf{x},Q^{\mathbf{w}})}.$$

  // For each trajectory.
  FOR $l = 1$ TO $L$:
      // Initialization.
      LET $\Delta_l(\mathbf{w}) = \mathbf{0}$.
      LET $\delta_l(\mathbf{w}) = \mathbf{0}$.
      SAMPLE INITIAL STATE $\mathbf{x}^{(0)}$.
      // For each step.
      FOR $t = 0$ TO $\tau_{max}$:
          // Sample action from soft-max policy, and get partition function for free.
          LET $\left[\mathbf{a}^{(t)}, Z(\mathbf{x}^{(t)})\right] = \text{MULTIAGENTSOFTMAXPOLICY}(\{e^{\frac{1}{T}Q_1^{\mathbf{x}^{(t)}}}, \ldots, e^{\frac{1}{T}Q_g^{\mathbf{x}^{(t)}}}\}, \mathcal{O})]$.
          // Execute action, and observe reward and next state.
          EXECUTE ACTION $\mathbf{a}^{(t)}$, AND OBSERVE REWARD $r^{(t)}$ AND NEXT STATE $\mathbf{x}^{(t+1)}$.
          // Compute the derivative of the log of the policy for each parameter $w \in \mathbf{w}$.
          FOR EACH AGENT $i$ AND EACH PARAMETER $w_i \in \mathbf{w}_i$, LET:

            $$\delta_l(w_i) = \delta_i(w_i) + \text{MULTIAGENTPOLICYDERIVATIVE}(Q^{\mathbf{x}^{(t)}}, T, \mathbf{a}^{(t)}, i, w_i, Z(\mathbf{x}^{(t)}), \mathcal{O}).$$

          // Update the gradient of the value.
          LET $\Delta(w) = \Delta(w) + \gamma^t r^{(t)} \delta(w)$, FOR EACH PARAMETER $w \in \mathbf{w}$.
  RETURN GRADIENT $\Delta(\mathbf{w}) = \frac{1}{L}\sum_l \Delta_l(\mathbf{w})$.
```

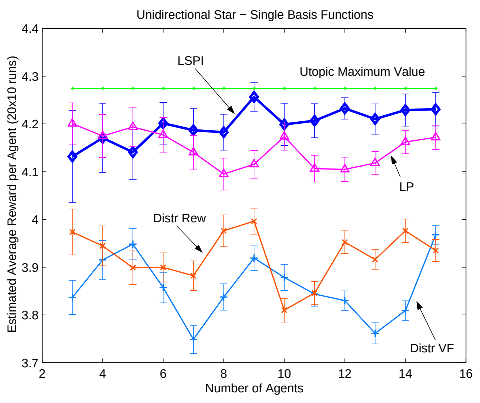
Figure 11.6: Procedure for the multiagent REINFORCE algorithm for computing an estimate to the gradient of the value of our multiagent soft-max policy.

for a variety of network topologies. Figure 11.7 shows the estimated value of the resulting policies for problems with increasing number of agents. For comparison, we also plot the results for three other methods: our planning algorithm using the factored LP-based approximation (LP); and the algorithms of Schneider *et al.* [1999], distributed reward (DR) and distributed value function (DVF). Note, the LP-based approach is a planning algorithm, *i.e.*, uses full knowledge of the (factored) MDP model. On the other hand, coordinated RL, DR and DVF are all model-free reinforcement learning approaches.

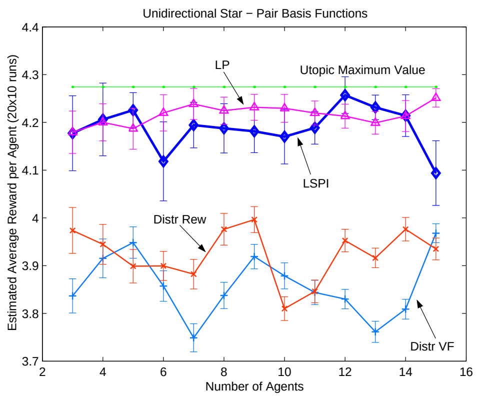
We experimented with two sets of multiagent LSPI basis functions corresponding to the backprojections of the “single” and of the “pair” basis functions in Section 9.3. For n machines, we found that about $600n$ samples are sufficient for multiagent LSPI to learn a good policy. Samples were collected by starting at the initial state (with all working machines) and following a purely random policy. To avoid biasing our samples too heavily by the stationary distribution of the random policy, each episode was truncated at 15 steps. Thus, samples were collected from $40n$ episodes each one 15 steps long. The resulting policies were evaluated by averaging performance over 20 runs of 100 steps. The entire experiment was repeated 10 times with different sample sets and the results were averaged. Figure 11.7 shows the results obtained by LSPI compared with the results of LP, DR, and DVF. We also plot the “Utopic maximum value”, a loose upper bound on the value of the optimal policy.

The results in all cases clearly indicate that multiagent LSPI learns very good policies comparable to the LP approach using the same basis functions, but *without* any use of the model. Note that these policies are near-optimal, as their values are very close to the upper bound on the value of the optimal policy. It is worth noting that the number of samples used grows linearly in the number of agents, whereas the joint state-action space grows exponentially. For example, a problem with 15 agents has over 205 trillion states and 32 thousand possible actions, but required only 9000 samples.

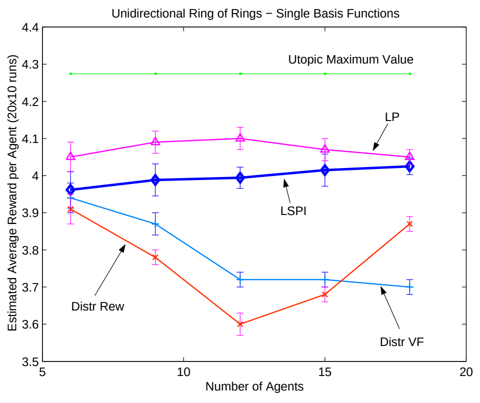
We also tested our multiagent LSPI approach on the power grid domain of Schneider *et al.* [1999]. Here, the grid is composed of a set of nodes. Each node is either a Provider (a fixed voltage source), a Customer (with a desired voltage), or a Distributor. Links from distributors to other nodes are associated with resistances and no customer is connected directly to a provider. The distributors must set the resistances to meet the demand of the



(a)



(b)



(c)

Figure 11.7: Comparing multiagent LSPI with factored LP-based approximation (LP), and with the distribute reward (DR) and distributed value function (DVF) algorithms of [Schneider *et al.*, 1999], on the SysAdmin problem. Estimated discounted reward per agent of resulting policies are presented for topologies: (a) star with "single" basis; (b) star with "pair" basis; (c) ring of rings with "single" basis.

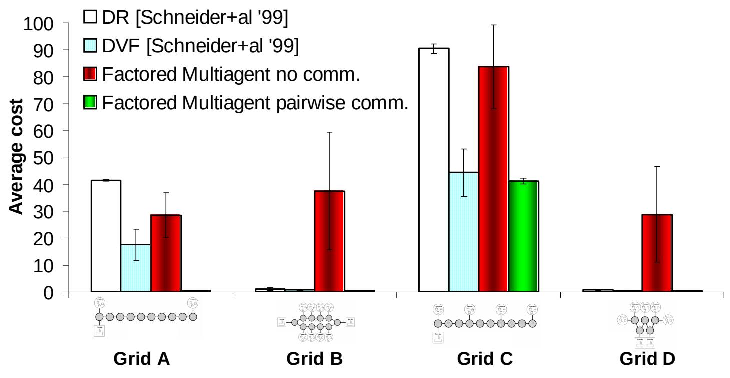


Figure 11.8: Comparison of our multiagent LSPI algorithm with the DR and DVF algorithms of [Schneider *et al.*, 1999] on their power grid problem: average cost over 10 runs of 60000 steps and 95% confidence intervals. DR and DVF results as reported in [Schneider *et al.*, 1999].

customers. If the demand of a particular customer is not met, then the grid incurs a cost equal to the demand minus the supply. At every time step, each distributor can decide whether to double, halve or maintain the value of the resistor at each of its links. If two distributors are linked, they share the same resistance and their action choices may conflict. In such case, a conflict resolution schema is applied, *e.g.*, if distributor 1 is connected to distributor 2, and distributor 1 wants to halve the resistance and distributor 2 wants to double it, then the value is maintained. We refer to the presentation of Schneider *et al.* [1999] for further details.

Schneider *et al.* [1999] proposed a set of algorithms, including DR and DVF, and applied them to this problem. In their set up, each distributor observes a set of state variables, including the value of the resistance at each of its links, the sign of the voltage differential to the neighbors, etc; then, it makes a local decision for each of its links. We applied our multiagent LSPI algorithm to the same problem with two simple types of state-action basis functions: “no comm.”, which is composed of indicators for each assignment of the state of the resistor and the action choice, with a total of 9 indicator bases for each end of a link; and “pair comm.”, which has indicator bases for each assignment of the resistance level, action of distributor i and action of distributor j, for each pair (i, j) of directly connected distributors (27 indicators per pair). Thus, our agents observe a much smaller part of the state than those of Schneider *et al.* [1999]. The quality of the resulting policies are shown in Figure 11.8. Multiagent LSPI used $10,000$ samples with different sample sets for each run. The multiagent LSPI results with the “no comm.” basis set are sub-optimal. Although some of the policies obtained with this basis set were near-optimal, most were close to random and the resulting average cost was high (with large confidence intervals). However, the very simple pairwise coordination strategy obtained from the “pair comm.” basis set yielded near-optimal policies. The DR and DVF agents must communicate during the learning process, but not during action selection. Our “pair comm.” basis set requires a coordination step in both steps. These agents incur a lower average cost than the DR and DVF agents for all grids and observe a much smaller part of the state space.

11.5 Discussion and related work

We propose a new approach to reinforcement learning: *coordinated RL*. In this approach, agents make coordinated decisions and share information to achieve a principled learning strategy. Our method successfully incorporates the cooperative action selection mechanisms described in Chapters 9 and 10 into the reinforcement learning framework to allow for structured communication between agents, each of which has only partial access to the state description. A feature of our method is that the structure of the communication between agents is not fixed *a priori*, but derived directly from the value function or policy architecture.

We believe our coordination mechanism can be applied to almost any reinforcement learning method. In this chapter, we applied the coordinated RL approach to Q-learning [Watkins, 1989; Watkins & Dayan, 1992], LSPI [Lagoudakis & Parr, 2001], and policy search [Williams, 1992]. With Q-learning and policy search, the learning mechanism can be distributed; agents communicate reinforcement signals, utility values, and conditional policies. In LSPI, some centralized coordination is required to compute the projection of the value function. The resulting policies can always be executed in a distributed manner. In our view, a batch algorithm, such as LSPI, can provide an offline estimate of the Q-function. Subsequently, Q-learning or direct policy search can be applied online to refine this estimate. By using our coordinated RL method, we can smoothly shift between these two phases, in collaborative multiagent settings.

We evaluate our coordinated RL methods, comparing the results both to our planning algorithm, and to other RL approaches. In these experiments, we reliably learned policies that were comparable to the best policies achieved by our planning algorithm with full knowledge of the model, and that were better than other state-of-the-art RL approaches. The amount of data required scaled linearly with the number of state and action variables even though the state and action spaces were growing exponentially.

Our coordinated RL experiments involved discrete state spaces. These domains were chosen primarily to compare learning performance with our planning algorithm. However, the methods discussed in this chapter will also apply to collaborative multiagent planning problems in continuous state spaces.

Learning in the context of collaborative multiagent problems has also been widely explored in the past. Claus and Boutilier [1998] partition methods into *independent learners* (IL), where each agent learns to optimize ignoring the existence of other agents, and *joint action learners* (JAL), where agents learn the value of their actions in conjunction with other agents through coordination. The policy search method of Peshkin *et al.* [2000] can be seen as an example of IL, as the gradient is decomposed into an independent term for each agent. On the other hand, reward or value sharing methods, such as those of Schneider *et al.* [1999] and Wolpert *et al.* [1999], enforce some coordination between agents when learning the parameters of the value function. The method of Sallans and Hinton [2001], discussed in more detail in Section 9.4, requires an approximate action selection step. Our method seeks to optimize the global Q-function or policy, through an efficient distributed agent coordination mechanism. This optimization is achieved with our coordination graph and a particular choice of approximation architecture, *i.e.*, a sum of local Q_i's for each agent, where each Q_i uses a linear architecture in multiagent LSPI, and each Q_i can use any approximation architecture in multiagent Q-learning and multiagent REINFORCE. Our empirical evaluation demonstrates that such coordination can significantly improve the quality of the policies obtained. Our approach will, of course, be most advantageous when the true Q-function can be approximated reasonably by such a linear combination of local Q-functions defined over subsets of the agents.

In this part of the thesis, we developed a suite of algorithm for coordination, planning and learning in large-scale systems. We believe that these methods will provide a strong foundation for solving complex real-world dynamic decision-making problems involving multiple agents.

Part IV

Generalization to new environments

Chapter 12

Relational Markov decision processes

Most planning methods, including the ones presented thus far in this thesis, are designed to optimize the plan of an agent in a fixed environment. However, in many real-world settings, an agent will face multiple environments over its lifetime, and its experience with one environment should help it to perform well in another.

Consider, for example, an agent designed to play a strategic computer war game, such as the *Freecraft* game shown in Figure 12.1 (an open source version of the popular *Warcraft*® game). In this game, the agent is faced with many scenarios. In each scenario, it must control a set of agents (or units) with different skills in order to defeat an opponent. Most scenarios share the same basic elements: *resources*, such as gold and wood; *units*, such as peasants, who collect resources and build structures, and footmen, who fight with enemy units; and *structures*, such as barracks, that are used to train footmen. To avoid competitive multiagent settings, as described in Chapter 1, we are assuming that the Freecraft controlled enemies are part of the environment and do not respond strategically to our policy choice. Each scenario is composed of these same basic building blocks, but they differ in terms of the map layout, types of units available, amounts of resources, etc. We would like the agent to learn from its experience with playing some scenarios, enabling it to tackle new scenarios without significant amounts of replanning. In particular, we would like the agent to generalize from simple scenarios, allowing it to deal with other scenarios that are too complex for any effective planner.

The idea of generalization has been a longstanding goal in traditional planning [Fikes

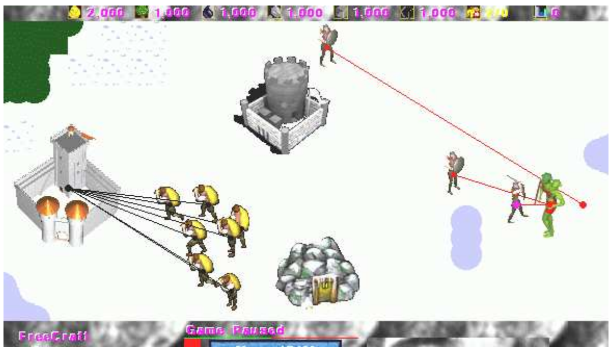


Figure 12.1: Freecraft strategic domain with 9 peasants, a barrack, a castle, a forest, a gold mine, 3 footmen, and an enemy; executing the generalized policy computed by our algorithm.

et al., 1972], and later in Markov decision processes and reinforcement learning research [Sutton & Barto, 1998; Thrun & O'Sullivan, 1996]. This problem is a challenging one, because it is often unclear how to translate the solution obtained for one domain to another. MDP solutions assign values and/or actions to states. Two different MDPs (*e.g.*, two Freecraft scenarios), are typically quite different, in that they have a different set (and even number) of states and actions. In cases such as this, the mapping of one solution to another is not obvious.

Our approach is based on the insight that many domains can be described in terms of objects and the relations between them. A particular domain will involve multiple objects from several classes. Different tasks in the same domain will typically involve different sets of objects, related to each other in different ways. For example, in Freecraft, different tasks might involve different numbers of peasants, footmen, enemies, etc. We therefore define a notion of a *relational MDP (RMDP)*, based on the *probabilistic relational model (PRM)* framework of Koller and Pfeffer [1998]. An RMDP for a particular domain provides a general schema for an entire suite of environments, or worlds, in that domain. It specifies a set of classes, and how the dynamics and rewards of an object in a given class depend on the state of that object and of related objects.

We use the class structure of the RMDP to define a value function that can be generalized from one domain to another. We begin with the assumption that the value function is approximated with our factored value function representation. Thus, the value of a global Freecraft state is approximated as a sum of terms corresponding to the state of individual peasants, footmen, gold, etc. We then assume that individual objects in the same class have a very similar value function. Thus, we define the notion of a *class-based value function*, where each class is associated with a *class value subfunction*. All objects in the same class have the value subfunction of their class. The overall value function for a particular environment is the sum of value subfunctions for the individual objects in the domain.

A set of value subfunctions for the different classes immediately determines a value function for any new environment in the domain, and can be used for acting. Thus, we can compute a set of class subfunctions based on some a subset of environments, and apply them to a new environment without replanning for it.

In addition to a computer game, there are many other domains where this relational framework could be applied. In Chapter 1, we describe a few such application domains. For example, in manufacturing settings, modern factories are often composed of cells, where each cell is of one of a few "types", or classes in our relational model. In sensor networks, a large-scale sensing task is performed by a large collection of a few types of sensors. These tasks could potentially be addressed effectively using relational MDPs to generalize from small scenarios to the large-scale ones required in practice.

12.1 Relational representation

A *relational MDP* defines the system dynamics and rewards at the level of a template for a task domain. Given a particular environment within that domain, it defines a specific factored MDP instantiated for that environment.

12.1.1 Class template

As in the *probabilistic relational model* (PRM) framework of Koller and Pfeffer [1998], the domain in a relational MDP is defined via a *schema* that specifies a set of *object classes* $\mathcal{C} = \{C_1, \ldots, C_c\}$. Each class C is associated with a set of *state variables* $\mathcal{X}[C] = \{C.X_1, \ldots, C.X_k\}$ that describe the state of an object in that class. As in a factored

MDP, each state variable $C.X$ has a *domain* of possible values $\mathrm{Dom}[C.X]$. We define $\mathbf{X}_C$ to be the random variables defining the state of an object in C, *i.e.*, the assignment to the state variables $\mathcal{X}[C]$ of class C. Each cell is of one of a few "types", or classes in our relational model. For each class, the schema also specifies a set of *action variables* $\mathcal{A}[C] = \{C.A_1, \ldots, C.A_g\}$. Each action variable $C.A_i$ can take on one of several assignments $\mathrm{Dom}[C.A_i]$, and we use $\mathbf{A}_C$ to define the set of possible assignments to all action variables of class C.

Example 12.1.1 (Freecraft classes) *The classes in our Freecraft domain might include: Peasant, Footman, Resource, etc. The class Peasant may have a state variable Task whose domain is* $\mathrm{Dom}[\textsf{Peasant}.\textit{Task}] = \{\textit{Waiting, Mining, Harvesting, Building}\}$*, and a state variable Health whose domain has three values, indicating the peasant's health level. In this case,* $\mathbf{X}_{\textsf{Peasant}}$ *would have* $4 \cdot 3 = 12$ *assignments, one for each combination of values for Task and Health.*

Additionally, a peasant can decide to collect resources, by mining or harvesting wood, or to build a structure. Thus, the peasant class Peasant is associated with a single action variable whose domain is $\mathrm{Dom}[\textsf{Peasant}.A] = \{\textit{Wait, Mine, Harvest, Build}\}$. ▮

12.1.2 Links

The schema also specifies a set of *links* $\mathcal{L}[C] = \{L_1, \ldots, L_l\}$ for each class C representing links between objects in the domain. Each link $C.L$ has a *range* $\rho[C.L] = C'$, indicating that an object of class C is linked to one object of class C'. In a more complex situation, a link may relate a class C to many instances of a class C' simultaneously. We denote such a *set link* by $\rho[C.L] = \textsf{SetOf}\{C'\}$, *i.e.*, every object of class C is linked to zero, one, or (possibly) many objects of class C'.

Example 12.1.2 (Freecraft links) *In our Freecraft example, a barrack can be built by a peasant if enough resources are available. Thus, objects of class Barrack might be linked to Peasant objects –* $\rho[\textsf{Barrack}.\textit{BuiltBy}] = \textsf{Peasant}$. *In addition a barrack is linked to two instances of the resource class:* $\rho[\textsf{Barrack}.\textit{MyWood}] = \textsf{Resource}$*, and* $\rho[\textsf{Barrack}.\textit{MyGold}] = \textsf{Resource}$.

The relationship between footmen and enemies is more complex, as multiple footmen can attack an enemy at same time. In this case, an object of the class Enemy *may be linked to multiple objects of the class* Footman*, which we denote by* $\rho[\mathsf{Enemy}.\mathit{My\_Footmen}] = \mathsf{SetOf}\{\mathsf{Footman}\}$. ▮

12.1.3 A world

A particular instance of the schema is defined via a *world* ω, specifying the set of objects of each class, and the links between them. For a particular world ω, we use $\mathcal{O}[\omega][C]$ to denote the objects of class C, and $\mathcal{O}[\omega]$ to denote the total set of objects in ω. A state $\mathbf{x}$ of the world ω at a given point in time is a vector defining the states of the individual objects in the world. We use $\mathbf{x}_o$ for an object o to denote $\mathbf{x}[\mathbf{X}_o]$, *i.e.*, the instantiation in $\mathbf{x}$ to the state variables of object o. Similarly, an action $\mathbf{a}$ in the world ω defines $\mathbf{a}_o$, the assignment to the action variables of object o.

The world ω also specifies the domain of possible values of the links between objects. Thus, for each link $C.L$, and for each $o \in \mathcal{O}[\omega][C]$, ω specifies $\mathrm{Dom}_\omega[o.L]$, the set of possible values of $o.L$. Each value $o.\ell \in \mathrm{Dom}_\omega[o.L]$ specifies a set of objects $o' \in \rho[C.L]$. We assume that the domain of values $\mathrm{Dom}_\omega[o.L]$ is fixed throughout time, but the particular value $o.\ell$ of the link may change.

Example 12.1.3 (Freecraft world) *Consider a Freecraft scenario containing 2 peasants, a barrack, and a gold mine. In order to specify a world for this scenario, we would first define two instances of class* Peasant*, which we denote by* $\mathcal{O}[\omega][\mathsf{Peasant}] = \{\text{Peasant1}, \text{Peasant2}\}$*, an instance of the barrack class, denoted by* $\mathcal{O}[\omega][\mathsf{Barrack}] = \{\text{Barrack1}\}$*, and, finally,* $\mathcal{O}[\omega][\mathsf{Gold}] = \{\text{Gold1}\}$*. If* Peasant1 *is responsible for building the barrack, we would specify the link* $\text{Barrack1}.\mathit{BuiltBy} = \text{Peasant1}$*, whose domain has a single value, thus does not change over time. We describe a Freecraft domain with a changing relational structure later in this chapter.* ▮

12.1.4 Transition model template

This section presents the basic elements forming the relational representation of the transition model.

Class transition model: The dynamics and rewards of an RMDP are also defined at the schema level. Each class C is associated with a *class transition model* P^C that specifies the probability distribution over the next state of an object o in class C, given the current state $\mathbf{x}_o$ of this object, the assignment to its action variables $\mathbf{a}_o$, and the states and actions of all of the objects linked to o:

$$P^C(\mathbf{X}'_C \mid \mathbf{X}_C, \mathbf{A}_C, \mathbf{X}_{C.L_1}, \mathbf{A}_{C.L_1}, \ldots, \mathbf{X}_{C.L_l}, \mathbf{A}_{C.L_l}). \tag{12.1}$$

As discussed by Koller and Pfeffer [1998], in addition to depending on the state of linked objects $L_i \in \mathcal{L}[C]$, such a relational representation can recursively include dependencies on objects linked to objects in L_i, *e.g.*, objects in $L_i.L_j$, for $L_j \in \mathcal{L}[C']$ such that $\rho[C.L_i] = C'$, as long as the recursion is guaranteed to be finite. We refer the reader to the presentation of Koller and Pfeffer [1998] for further details.

In general, $\mathbf{X}'_C$ is a set of state variables. We can thus represent P^C compactly using a dynamic decision network (DDN), as in Section 8.1.1. In the graph for this DDN, the parents of each state variable $C.X'_i$ for class C will be a subset of the state and action variables of C and of the objects linked to this class, which we denote by:

$$\mathsf{Parents}(C.X'_i) \subseteq \{\mathcal{X}[C], \mathcal{A}[C], \mathcal{X}[C.L_1], \mathcal{A}[C.L_1], \ldots, \mathcal{X}[C.L_l], \mathcal{A}[C.L_l]\}. \tag{12.2}$$

The conditional probability distribution (CPD) for $C.X'_i$ will thus be given by:

$$P^{C.X'_i}(C.X'_i \mid \mathsf{Parents}(C.X'_i)). \tag{12.3}$$

Using this factored representation, the class transition probabilities become:

$$P^C(\mathbf{X}'_C \mid \mathbf{X}_C, \mathbf{A}_C, \mathbf{X}_{C.L_1}, \mathbf{A}_{C.L_1}, \ldots, \mathbf{X}_{C.L_l}, \mathbf{A}_{C.L_l}) = \prod_i P^{C.X'_i}(C.X'_i \mid \mathsf{Parents}(C.X'_i)). \tag{12.4}$$

Example 12.1.4 (Freecraft class transition model) *In Freecraft, a peasant can choose to build a barrack. If there are enough resources (gold and wood), this barrack will be built with high probability in the next time step. Thus, the transition model for the status of a barrack in the next time step,* $\mathsf{Barrack}.Status'$*, depends on its status in the current time* *step, on the task performed by any peasant that could build it (*$\mathsf{Barrack}$*.BuiltBy.Task), and on the amount of wood and gold.* ▮

Aggregators: The transition model for a class C is conditioned on the state of the objects in $C.L_i$. In general, when $\rho[C.L_i] = \mathsf{SetOf}\{C'\}$, L_i links an object of class C to a set of objects of class C' (*e.g.*, the set of footmen that can attack an enemy). Thus, our class template must provide a compact specification of the transition model that can depend on the state of an unbounded number of variables. We can deal with this issue using the idea of *aggregation* [Koller & Pfeffer, 1998]. Note that every object o' in $C.L_i$ belongs to the same class C'. Thus, these objects have the same set of state and action variables. Intuitively, aggregation summarizes the state of the objects o' linked to an object o of class C. The transition model will then depend on this summary, rather than on the state of every object in $C.L_i$ in isolation.

More specifically, we define a *counting function* that counts the number of objects in a particular state:

Definition 12.1.5 (counting function) *Let $\mathcal{B} = \{o_1, \ldots, o_m\}$ be a set of objects of a class C. Also, let $\mathbf{Y} \subseteq \{\mathcal{X}[C], \mathcal{A}[C]\}$ be a subset of the state and action variables of class C. We define the* counting function $\sharp$ *for some assignment $\mathbf{y}$ to $\mathbf{Y}$ in a world state $\mathbf{x}$ and action $\mathbf{a}$ by:*

$$\sharp(\mathcal{B}, \mathbf{x}, \mathbf{a}, \mathbf{y}) = \sum_{i=1}^{m} \mathbb{1}((\mathbf{x}_{o_i}, \mathbf{a}_{o_i})[\mathbf{Y}] = \mathbf{y}),$$

where $(\mathbf{x}_{o_i}, \mathbf{a}_{obj_i})[\mathbf{Y}]$ is the instantiation to the variables in $\mathbf{Y}$ in $(\mathbf{x}_{o_i}, \mathbf{a}_{obj_i})$, the state and action of object o_i defined in $\mathbf{x}$ and $\mathbf{a}$. ▮

Using this notion of aggregation we can formalize the class transition probabilities in Equation (12.1) for cases where $C.L_i$ is a set link to elements of class C'. In such cases, the probability of $\mathbf{X}'_C$ will depend on $\sharp(C.L_i, \mathbf{x}, \mathbf{a}, \mathbf{y})$, where $\mathbf{y}$ is an assignment to the variables in $\mathbf{X}_{C'}, \mathbf{A}_{C'}$. For simplicity of exposition, this notion of aggregation is only a special case of the one defined by Pfeffer [2000]. The more general notion will also apply in our relational MDP framework.

Example 12.1.6 (Freecraft aggregation) *In our Freecraft example, the transition model* *for an enemy's health depends on an aggregation of the footmen attacking it. Specifically, the probability that the assignment of* $\textsf{Enemy}.\mathit{Health}$ *transitions from Healthy to Dead depends on:*

$$\sharp\left(\textsf{Enemy}.\mathit{My\_Footmen},\ \mathbf{x},\ \mathbf{a},\ \ \textsf{Footman}.\mathit{Health} = \mathit{Healthy} \wedge \textsf{Footman}.\mathit{Action} = \mathit{Attack}\right),$$

that is, the number *of footmen in* $\textsf{Enemy}.\mathit{My\_Footmen}$ *who are healthy and attacking in the current setting to the state variables* $\mathbf{x}$ *and to the action variables* $\mathbf{a}$. ▮

Dynamic relational structure: The class-level transition model in Equation (12.1) is defined in terms of the links $C.L_i$. As discussed in Section 12.1.3, in any particular world a link $o.L$ for object o may take one of many values in $\mathrm{Dom}_\omega[o.L]$. Thus, the relational structure of the world may potentially change over time. To simplify our presentation, we assume that this evolution is deterministic. Specifically, at every time step, the current (joint) state and action will uniquely specify the particular value $o.\ell$ of $o.L$.

To allow agents or state variables to affect the evolution of links, we define the notion of a selector variable:

Definition 12.1.7 (selector variable) *If a variable* B *of class* C *is a* selector variable*,* $C.B = \textsf{Selector}[C'.L]$*, for some link* $C'.L$*; then, in a world* ω*, the domain of* $o.B$*, where* $o \in \mathcal{O}[\omega][C]$*, is given by:*

$$\mathrm{Dom}[o.B] = \mathrm{Dom}_\omega[o'.L],$$

for some $o' \in \mathcal{O}[\omega][C']$*. This instantiated selector variable is denoted by*

$$o.B = \textsf{Selector}_\omega[o'.L]. \quad ▮$$

In other words, the domain of a variable $o.B$ will be the domain of possible values of a link $o'.L$ of some (potentially different) object o'. Note that both state and action variables can be selectors in this formulation. Using a selector action an object can, for example, choose which object it will influence in the next time step. Such settings include a computer network, where a machine can choose to send packets to one of a few other machines, and Freecraft, where a footman can choose to attack one of several enemies.

To simplify our models, we assume that an object can only have a selector variable over its own links or the links of related objects. Specifically, if $o.B = \textsf{Selector}_\omega[o'.L]$ we assume that either $o = o'$, or $o \in \mathrm{Dom}_\omega[o'.L]$. For example, a computer selecting which machine to receive packets from is an example where $o = o'$, as the machine is selecting the computers that will influence its state. In Freecraft, if we allow a footman to select an enemy to attack, we are considering the second case, $o \in \mathrm{Dom}_\omega[o'.L]$, as the footman is setting an object that influences this enemy. This restriction ensures that the links of each object o can only be influenced by its own state and action, or by the state and action of objects in $\mathrm{Dom}_\omega[o.L]$.

Returning to our dynamic relational structure, we assume that the value of a link is deterministically specified by each state and action. Specifically, for each link $o.L$, the value $o.\ell$ at some time step will be deterministically specified by object o and the objects in $\mathrm{Dom}_\omega[o.L]$. We will not introduce further notation to represent the actual function specifying the value $o.\ell$ of the link. We will simply assume that object o is linked to every object in $\mathrm{Dom}[o.L]$, and that our CPD will select the specific target according to the current state and action. We can view this formulation as a form of context-specific structure, where the context, specified by the current action and state, defines which linked objects o' will influence o in the next time step.

Pfeffer [2000] defines relations as first class objects, and thus considers uncertainty about the target of a link. Using his formulation, we could define a more general notion of relations that change over time. However, such models could significantly increase the computational cost of our algorithm.

12.1.5 Reward function template

Finally, we must also define rewards at the class level. We assume for simplicity of notation that rewards are associated only with the states of individual objects; adding dependencies on linked objects is straightforward. We define a reward function $R^C(\mathbf{X}_C, \mathbf{A}_C)$ that represents the contribution to the reward of any object in C. We assume that the reward for *each object* is bounded by R^o_{max}, or equivalently,

$$R^o_{max} \geq R^C(\mathbf{x}_C, \mathbf{a}_C) \geq 0, \;\; \forall \mathbf{x}_C \in \mathbf{X}_C, \; \forall \mathbf{a}_C \in \mathbf{A}_C, \; \forall C \in \mathcal{C}.$$

Example 12.1.8 (Freecraft class reward function) *In Freecraft, we may have a reward function associated with the* Enemy *class that specifies a reward of 10 if the state of an enemy object is Dead:*

$$R^{\textsf{Enemy}}(\textsf{Enemy}.Health) = \begin{cases} 10, & \textit{if } \textsf{Enemy}.Health = Dead; \\ 0, & \textit{otherwise.} \end{cases} \quad ▮$$

More elaborate models may include more global reward functions; for example, the player may only receive a reward when all enemies are dead. Although such reward functions can be represented compactly using a relational model, the efficiency of our planning algorithm can be hindered by factors that depend on many objects simultaneously.

12.2 From templates to factored MDPs

Given a world ω, the RMDP representation uniquely defines a *ground* factored MDP Π_ω, whose transition model is specified (as usual) as a dynamic decision network (DDN) [Dean & Kanazawa, 1988]. The random variables in this factored MDP are the state variables of the individual objects $o.X$, for each $o \in \mathcal{O}[\omega][C]$ and for each $X \in \mathcal{X}[C]$. Similarly, the action variables will be $o.A$, for each $o \in \mathcal{O}[\omega][C]$ and for each $A \in \mathcal{A}[C]$.

Next, we must define the transition graph associated with the ground DDN of our factored MDP. This graph specifies the dependence of the variables at time $t+1$ on the variables at time t. Consider the parents of a state variable $o.X'_i$, where o is an instance of class C, *i.e.*, $o \in \mathcal{O}[\omega][C]$. In Equation (12.2), our template for the class transition probabilities defines the set of parents for $C.X'_i$, the class-level state variable corresponding to $o.X'_i$. Our world ω specifies the assignment to the links $\mathcal{L}[C]$, *i.e.*, the objects o' that are linked to o. Once we set these assignments to the links into Equation (12.2), we obtain the set of parents $\textsf{Parents}(o.X'_i)$ of $o.X'_i$ in our ground DDN.

A template for the conditional probability distribution (CPD) for the class state variable $C.X'_i$ was specified at the class-level in Equation (12.3). Using this template, we can specify the CPD for $o.X'_i$:

$$P^{C.X'_i}(o.X'_i \mid \textsf{Parents}(o.X'_i)). \tag{12.5}$$

Example 12.2.1 (Freecraft ground DDN) *In a Freecraft world with two peasants, the random variables in the ground DDN include:*

$$\text{Peasant1}.\textit{Task},\ \text{Peasant2}.\textit{Task},\ \text{Barrack1}.\textit{Status},\ \textit{etc.}$$

The parents of the time $t+1$ variable Barrack1.*Status*$'$ *are the time t variables:*

$$\text{Barrack1}.\textit{Status},\ \text{Peasant1}.\textit{Task},\ \text{Gold1}.\textit{Amount and } \text{Wood1}.\textit{Amount}.$$

The transition model is the same for all instances in the same class, as in Equation (12.1). Thus, all of the o.Status variables for barrack objects share the same conditional probability distribution. Note, however, that each specific barrack depends on the particular peasants linked to it. Thus, the actual parents in the DDN of the status variables for two different barrack objects can be different. That is, the parents of the status variable of a particular barrack will only include the task variables of peasants linked to this barrack. ▮

The reward function in our ground factored MDP Π_ω is simply the sum of the reward functions for the individual objects:

$$R^\omega(\mathbf{x}, \mathbf{a}) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} R^C(\mathbf{x}[\mathbf{X}_o], \mathbf{a}[\mathbf{A}_o]).$$

In our simple Freecraft example, our overall reward function in a given state will be 10 times the number of dead enemies in that state.

It remains to specify the actions in the ground MDP. The RMDP specifies a set of possible action variables for every object in the world. In a setting where only a single action can be taken at any time step (single agent case), the agent must choose both an object to act on, and which action to perform on that object. In this case, the set of actions in the ground MDP is simply the union of the possible actions for each object:

$$\bigcup_{o \in \omega} \mathrm{Dom}[\mathbf{A}_o].$$

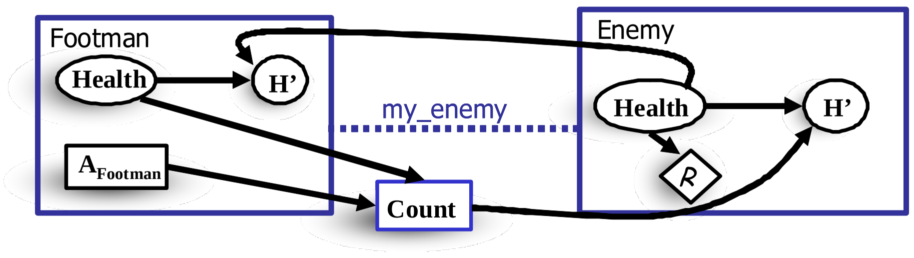


Figure 12.2: Schema for Freecraft tactical domain.

In a setting where multiple actions can be performed in parallel (say, in a collaborative multiagent setting), it might be possible to perform an action on every object in the domain at every step. Here, the set of actions in the ground MDP is a vector specifying an action for every object:

$$\bigotimes_{o \in \omega} \mathrm{Dom}[\mathbf{A}_o].$$

Intermediate cases, allowing degrees of parallelism, are also possible. For simplicity of presentation, we focus on the multiagent case. Freecraft is an example of a multiagent problem, where an action is an assignment to the action of every unit in the game.

Example 12.2.2 (Freecraft tactical domain) *Consider a simplified version of the Freecraft problem, whose schema is illustrated in Figure 12.2, where only two classes of units participate in the game:* $\mathcal{C} = \{\mathsf{Footman}, \mathsf{Enemy}\}$*. Both the footman and the enemy classes have only one state variable each, Health, with domain:*

$$\mathrm{Dom}[\mathit{Health}] = \{\mathit{Healthy, Wounded, Dead}\}.$$

The footman class contains one single-valued link: $\rho[\mathsf{Footman}.\mathit{My\_Enemy}] = \mathsf{Enemy}$*. Thus, the transition model for a footman's health will depend on the health of its enemy:*

$$P^{\mathsf{Footman}}(\mathbf{X}'_{\mathsf{Footman}} \mid \mathbf{X}_{\mathsf{Footman}}, \mathbf{X}_{\mathsf{Footman}.\mathit{My\_Enemy}}),$$

that is, if a footman's enemy is not dead, then the probability that a footman will become wounded, and eventually die, is significantly higher.

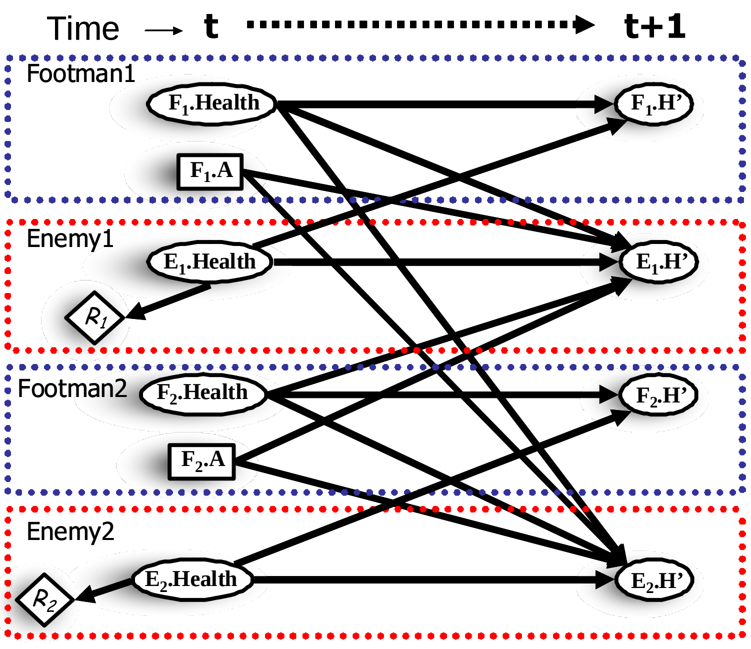


Figure 12.3: Resulting factored MDP for Freecraft tactical domain for a world with 2 footmen and 2 enemies.

A footman can choose deterministically to attack any enemy. thus, each footman is associated with an action $\mathsf{Footman}.A$ *that selects the enemy it is attacking:* $\mathsf{Footman}.A = \mathsf{Selector}\,[\mathsf{Enemy}.My\_Footmen]$. *The actual value of* $\mathsf{Enemy}.My\_Footmen$ *at some point in time will be deterministically specified as the union of the footmen who select to attack this enemy. As a consequence, an enemy could end up being linked to a set of footmen,* $\rho[\mathsf{Enemy}.My\_Footmen] = \mathsf{SetOf}\{\mathsf{Footman}\}$. *In this case, the transition model of the health of an enemy may depend on the number of footmen who are not dead and whose action choice is to attack this enemy:*

$$P^{\mathsf{Enemy}}\left(\mathbf{X}'_{\mathsf{Enemy}} \mid \mathbf{X}_{\mathsf{Enemy}}, \sharp\left(\mathsf{Enemy}.My\_Footmen, \mathbf{X}, \mathbf{A},\ \neg\mathsf{Footman}.Health = Dead \wedge \mathsf{Footman}.A = this\ Enemy\right)\right).$$

Finally, we must define the template for the reward function. Here there is only a reward when an enemy is dead: $R^{\mathsf{Enemy}}(\mathbf{X}_{\mathsf{Enemy}})$.

We now have a template to describe any instance of the tactical Freecraft domain. In a particular world, we must define the instances of each class and the links between these instances. For example, a world with 2 footmen and 2 enemies has 4 objects: $\{\textit{Footman1}, \textit{Footman2}, \textit{Enemy1}, \textit{Enemy2}\}$*. Each footman is linked to an enemy:*

$$\textit{Footman1.My\_Enemy} = \textit{Enemy1}, \ \ \textit{and} \ \ \textit{Footman2.My\_Enemy} = \textit{Enemy2}.$$

Each enemy can potentially be linked to both footmen: $\mathrm{Dom}_{2vs2}[\textit{Enemy1.My\_Footmen}] = \mathrm{Dom}_{2vs2}[\textit{Enemy2.My\_Footmen}] = \{\{\emptyset\}, \{\textit{Footman1}\}, \{\textit{Footman2}\}, \{\textit{Footman1}, \textit{Footman2}\}\}$. *At each time step the action choices of the two footmen will specify the actual value of these links.*

The template, along with the number of objects and the links in this specific ("$2vs2$*") world, yields a well-defined factored MDP,* Π_{2vs2}*, as shown in Figure 12.3.* ▮

12.3 Relational value functions

In our relational setting, the state space is exponentially large, with one state for each joint assignment to the random variables $o.X$ of every object (*e.g.*, exponential in the number of units in the Freecraft scenario). In a multiagent problem, the number of actions is also exponential in the number of agents. Thus, it is infeasible to represent the exact value function for such problems, and we must resort to an approximate solution.

12.3.1 Object value subfunctions

We again address the problem of exponential growth in the value function representation by using our factored linear value function, where the value function of a world is approximated as a sum of *local object value subfunctions* associated with the individual objects in the model. Here, we associate a value subfunction $\mathcal{V}_o$ with every object in ω. Most simply, this local value function can depend only on the state of the individual object $\mathbf{X}_o$. A richer approximation might associate a value function with pairs, or even small subsets, of closely related objects. Each object value subfunction $\mathcal{V}_o$ can be further decomposed into a linear combination of a set of *object basis functions*:

Definition 12.3.1 (object basis function, object value subfunction) *An* object basis function h_i^o *for object* o *is a function* $h_i^o : \mathbf{T}_{o,i} \mapsto \mathbb{R}$, *whose scope,* $\mathsf{Scope}[h_i^o] = \mathbf{T}_{o,i}$, *is a subset of the state variables of this object, and of related objects; formally, we have that:*

$$\mathbf{T}_{o,i} \subseteq \{\mathbf{X}_o, \mathbf{X}_{o.L_1}, \ldots, \mathbf{X}_{o.L_l}\}.$$

An object value subfunction $\mathcal{V}_o$ *for object* o *is a function* $\mathcal{V}_o : \mathbf{T}_o \mapsto \mathbb{R}$, *such that:*

$$\mathcal{V}_o(\mathbf{T}_o) = \sum_{h_i^o \in \mathsf{Basis}[o]} h_i^o w_i^o(\mathbf{T}_o),$$

where $\mathsf{Basis}[o]$ *is the set of basis functions associated with object* o*. Thus, the scope of* $\mathcal{V}_o$ *is given by:*

$$\mathbf{T}_o = \mathsf{Scope}[\mathcal{V}_o] = \bigcup_{h_i^o \in \mathsf{Basis}[o]} \mathsf{Scope}[h_i^o]. \blacksquare$$

Given a set of local value subfunctions, we approximate the global value function as:

$$\mathcal{V}_\omega(\mathbf{x}) = \sum_{o \in \mathcal{O}[\omega]} \mathcal{V}_o(\mathbf{x}[\mathbf{T}_o]). \tag{12.6}$$

Example 12.3.2 (Freecraft object value subfunctions) *In our Freecraft example, the local value subfunction* $\mathcal{V}_{\mathit{Enemy1}}$ *for enemy object Enemy1 might associate a numeric value for each assignment to the variable Enemy1.Health. We may use a richer approximation for the footman class, where the function* $\mathcal{V}_{\mathit{Footman1}}$ *for Footman1 might be defined over the joint assignments of Footman1.Health and Enemy1.Health, where Footman1.My_Enemy* $=$ *Enemy1. We represent the complete value function for a world as the sum of the local value subfunctions for the individual objects in this world. In our example world (*$\omega = 2vs2$*) with 2 footmen and 2 enemies, the global value function, shown in Figure 12.4(a), will be:*

$$\begin{aligned}\mathcal{V}_{2vs2}(F_1.\mathit{Health}, E_1.\mathit{Health}, F_2.\mathit{Health}, E_2.\mathit{Health}) = \\ \mathcal{V}_{\mathit{Footman1}}(F_1.\mathit{Health}, E_1.\mathit{Health}) + \mathcal{V}_{\mathit{Enemy1}}(E_1.\mathit{Health}) + \\ \mathcal{V}_{\mathit{Footman2}}(F_2.\mathit{Health}, E_2.\mathit{Health}) + \mathcal{V}_{\mathit{Enemy2}}(E_2.\mathit{Health}).\end{aligned}$$

$\blacksquare$

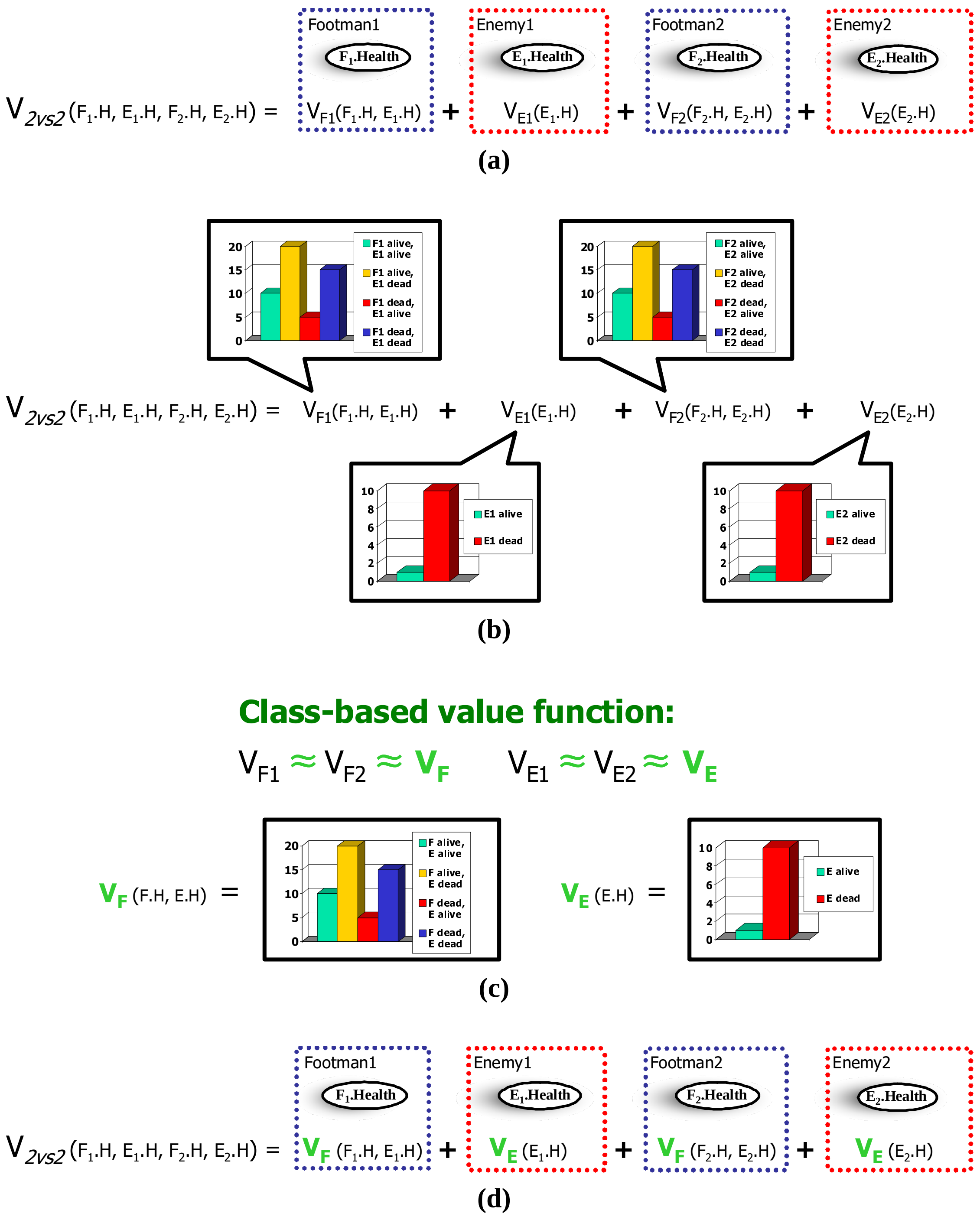


Figure 12.4: Relational value function representation in Freecraft tactical domain: (a) Factored value function in the object level for the $\omega = 2vs2$ world; (b) Illustrative values of the local object value subfunctions, objects of the same class have similar values; (c) Class-based value subfunctions; (d) Class-based value function instantiated in the $2vs2$ world.

12.3.2 Class-based value functions

As for any linear approximation to the value function, the factored algorithms presented thus far in this thesis can be used to compute the coefficients of the object value subfunctions efficiently. Although this approach provides us with a principled way of decomposing a high-dimensional value function in certain types of domains, it does not help us address the generalization problem: A local value function for objects in a world ω does not help us provide a value function for objects in other worlds, especially worlds with different sets of objects.

To obtain generalization, we build on the intuition that different objects in the same class behave similarly: they share the transition model and reward function in the relational MDP. Although they differ in their interactions with other objects, their local contribution to the value function is often similar. Consider our Freecraft example:

Example 12.3.3 (Freecraft class-based value function) *Consider the Freecraft world in Example 12.3.2. If we apply an approximate MDP solution algorithm to this problem, we obtain the actual numeric values of each object value function, such as the ones illustrated in Figure 12.4(b).*

As every footman behaves in a similar manner in the game, the numeric values of $\mathcal{V}_{Footman1}$ and $\mathcal{V}_{Footman2}$ are very similar, as are the values of $\mathcal{V}_{Enemy1}$ and $\mathcal{V}_{Enemy2}$. We can thus define new subfunctions for each class: $\mathcal{V}_{\mathsf{Footman}}$ for footmen, and $\mathcal{V}_{\mathsf{Enemy}}$ for enemies, as shown in Figure 12.4(c). We call these new subfunctions, class-based value subfunctions, *as they are defined for classes of objects.*

We can now use our class-based value subfunctions to represent the value function of the $2vs2$ world, as shown in Figure 12.4(d), by:

$$\begin{aligned}\mathcal{V}_{2vs2}(F_1.Health, E_1.Health, F_2.Health, E_2.Health) = &\\ \mathcal{V}_{\mathsf{Footman}}(F_1.Health, E_1.Health) + \mathcal{V}_{\mathsf{Enemy}}(E_1.Health) + &\\ \mathcal{V}_{\mathsf{Footman}}(F_2.Health, E_2.Health) + \mathcal{V}_{\mathsf{Enemy}}(E_2.Health). &\end{aligned} \tag{12.7}$$

Note that every object of the class Footman *uses the same value subfunction $\mathcal{V}_{\mathsf{Footman}}$, but every individual footman uses this subfunction with a different argument: the contribution of Footman1 depends on Footman1.Health and Enemy1.Health, while the contribution of*

Footman2 depends on Footman2.Health and Enemy2.Health. Thus, despite the fact that these two objects share the same class value subfunction, at every state their contribution to the global value function may be different. For example, in a state where Footman1 is alive and Footman2 is dead, the first object will have a higher contribution to the value function than the second.

In Equation (12.7), we show that the class-based value subfunctions give us a global value function for the $2vs2$ world. Importantly, this class-based representation can also give us a value function for any instance of the Freecraft tactical domain. Thus, we can generalize the value function obtained in a world with 2 footmen and 2 enemies to a world with N footmen and N enemies, without replanning:

$$\mathcal{V}_{NvsN}(F_1.\mathit{Health}, E_1.\mathit{Health}, \ldots, F_N.\mathit{Health}, E_N.\mathit{Health}) = \\ \sum_{i=1}^{N} \mathcal{V}_{\mathsf{Footman}}(F_i.\mathit{Health}, E_i.\mathit{Health}) + \mathcal{V}_{\mathsf{Enemy}}(E_i.\mathit{Health}), \tag{12.8}$$

assuming $F_i.\mathit{Enemy} = E_i$. ▮

This example illustrates our generalization approach: We restrict our space of value functions by requiring that all of the objects in a given class share the same local value subfunction. We can then generalize this type of value function to any world in our domain.

Formally, we define a *class value subfunction* $\mathcal{V}_C$ for each class, where each $\mathcal{V}_C$ is defined by a linear combination of *class basis functions* $\mathcal{V}_C = \sum_i w_i^C h_i^C$. We assume that the parameterization of this class value subfunction is well-defined for every object o in C. This assumption holds trivially if the scope of each h_i^C is restricted to state variables in $\mathcal{X}[C]$, as every instance of class C contains these variables. When the class basis functions can also depend on the state of linked objects, we must define the parameterization accordingly:

Definition 12.3.4 (class basis functions, class value subfunction) *A* class basis function *h_i^C for class C is a function $h_i^C : \mathbf{T}_{C,i} \mapsto \mathbb{R}$, whose scope, $\mathsf{Scope}[h_i^C] = \mathbf{T}_{C,i}$, is a subset of the state variables of this class, and of related objects, formally, we have that:*

$$\mathbf{T}_{C,i} \subseteq \{\mathcal{X}[C], \mathcal{X}[C.L_1], \ldots, \mathcal{X}[C.L_l]\}.$$

A class value subfunction $\mathcal{V}_C$ *for class* C *is a function* $\mathcal{V}_C : \mathbf{T}_C \mapsto \mathbb{R}$*, such that:*

$$\mathcal{V}_C(\mathbf{T}_C) = \sum_{h_i^C \in \mathit{Basis}[C]} w_i^C h_i^C(\mathbf{T}_C),$$

where $\mathit{Basis}[C]$ *is the set of basis functions associated with class* C*. Thus, the scope of* $\mathcal{V}_C$ *is given by:*

$$\mathbf{T}_C = \mathit{Scope}[\mathcal{V}_C] = \bigcup_{h_i^C \in \mathit{Basis}[C]} \mathit{Scope}[h_i^C]. \quad \blacksquare$$

As with the class transition model defined in Section 12.1.4, our class value subfunctions require aggregators to be defined appropriately when $C.L_i$ links an object of class C to a whole set of objects of class C'. Additionally, as with the transition model, class value subfunctions can depend recursively on the state of objects linked to the objects in $C.L_i$, that is, the objects in $C.L_i$, $C.L_i.L_j$, $C.L_i.L_j.L_k$, etc.

12.3.3 Generalization

Our class value subfunctions can be used to define a *class-based value function* specific for each world ω. This value function is represented as the sum of the class value subfunctions instantiated for each object in ω:

$$\mathcal{V}_\omega(\mathbf{x}) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(\mathbf{x}[\mathbf{T}_o]), \tag{12.9}$$

where $\mathbf{T}_o$ is the scope of the class value subfunctions $\mathbf{T}_C$ instantiated with the specific objects in the links defined by the world ω. This value function definition depends both on the set of objects in the world and (when local value functions can involve related objects) on the links between them.

Importantly, although objects in the same class contribute the same class subfunction into the summation of Equation (12.9), the argument of the function for an object is the state of that specific object (and perhaps of its related objects). In any given state, the contributions of different objects of the same class can differ. Thus, as illustrated in Example 12.3.3, every footman has the same local value subfunction parameters, but a dead footman will have a lower contribution than one that is alive.

Therefore, if we compute the coefficients of the class basis functions, we obtain a set of class value subfunctions that allow us to generate a value function for any world ω in our domain.

12.4 Discussion and related work

In this chapter, we present the new framework of relational MDPs. This model seeks to address a longstanding goal in planning research, the ability to generalize plans developed for some set of environments to a new but similar environment, with minimal or no re-planning. An RMDP can model a set of similar environments by representing objects as instances of different classes, building on the probabilistic relational models of Koller and Pfeffer [1998].

In order to generalize plans to multiple environments, we specify an approximate value function in terms of classes of objects and, in a multiagent setting, classes of agents. If we optimize the parameters of this class-level value function, we obtain a set of class value subfunctions that allow us to generate a value function for any world in our domain.

In the next chapter, we present an algorithm that estimates these parameters from a set of sampled environments, allowing us to generalize from these worlds to other worlds in our domain, without replanning. In particular, we can generalize to larger worlds than we can solve even with our factored approximate solution algorithms.

Chapter 13

Generalization to new environments with relational MDPs

In the previous chapter, we defined relational MDPs, a framework that provides a general schema for representing factored MDPs for an entire suite of environments, or worlds, in a domain. It specifies a set of classes, and how the dynamics and rewards of an object in a given class depend on the state of that object and of related objects. We also used the class structure of the RMDP to define a class-based value function that can be generalized from one domain to another.

In this chapter, we provide an optimality criterion for evaluating the quality of a class-based value function for a distribution over environments, and show how it can, in principle, be optimized using an LP. Unfortunately, this formulation requires an optimization over all possible worlds simultaneously. The number of possible worlds is usually too large for this approach to be feasible. Furthermore, if we need to consider all possible worlds, then we will not be achieving the type of generalization we are seeking. To address this problem, we also show how a class-based value function can be "learned" by optimizing it relative to a sample of "small" environments encountered by the agent. We prove that a polynomial number of sampled "small" environments suffices to construct a class-based value function that is close to the one obtainable for the entire distribution over (arbitrarily-large) environments. Finally, we show how we can improve the quality of our approximation by automatically discovering subclasses of objects that have "similar" value subfunctions.

13.1 Finding generalized MDP solutions

With a class-level value function, we can easily generalize from one or more worlds to a new one. To do so, we assume that a single set of class value subfunctions $\mathcal{V}_C$ is a good approximation across a wide range of worlds ω. Assuming we have such a set of value functions, we can act in any new world ω without replanning, as described in Section 12.3.2. We simply define a world-specific value function as in Equation (12.9), and use it to act.

In order for our generalization approach to be successful, we must now optimize $\mathcal{V}_C$ over an entire set of worlds simultaneously. To formalize this intuition, we assume that there is a probability distribution $P(\omega)$ over the worlds that the agent encounters. We want to find a single set of class value subfunctions $\{\mathcal{V}_C\}_{C\in\mathcal{C}}$ that is a good fit for this distribution over worlds. We view this task as one of optimizing for a single "meta-level" MDP $\Pi_{\texttt{meta}}$, where nature first chooses a world ω, and the rest of the dynamics are then determined by the MDP Π_ω.

More formally, the state space of $\Pi_{\texttt{meta}}$ is:

$$\{\mathbf{x}_0\} \cup \bigcup_{\omega} \{(\omega, \mathbf{x}) \ : \ \mathbf{x} \in \mathbf{X}_\omega\}.$$

The transition model is the natural one: From the initial state $\mathbf{x}_0$, nature chooses a world ω according to $P(\omega)$, and an initial state in ω according to some initial starting distribution $P^0_\omega(\mathbf{x})$ over the states in ω. The remaining evolution is then done according to ω's dynamics:

$$\begin{aligned} P((\omega,\mathbf{x}) \mid \mathbf{x}_0) &= P(\omega)\cdot P^0_\omega(\mathbf{x}) \\ P((\omega',\mathbf{x}') \mid (\omega,\mathbf{x}),\mathbf{a}) &= \begin{cases} 0\,, & \omega' \neq \omega\,; \\ P_\omega(\mathbf{x}' \mid \mathbf{x},\mathbf{a})\,, & \text{otherwise.} \end{cases} \end{aligned}$$

In our Freecraft example, nature will choose the number of footmen and enemies, and define the links between them, which then yields a well-defined MDP, *e.g.*, Π_{2vs2}.

13.2 LP formulation

The meta-MDP $\Pi_{\mathtt{meta}}$ allows us to formalize the task of finding a generalized solution to an entire class of MDPs. Specifically, we wish to optimize the class-level parameters for $\mathcal{V}_C$, not for a single ground MDP Π_ω, but for the entire meta-level MDP $\Pi_{\mathtt{meta}}$.

13.2.1 Object-based LP formulation

Consider first the problem of approximate planning for a single world ω. As each world is a factored MDP, we can address this problem using the LP solution algorithms presented thus far in this thesis, the ones in Chapter 5 for the single agent case, and in Chapter 9 for multiagent problems.

Variables: As described in Section 12.3.1, the value function for a particular world is represented by:

$$\mathcal{V}_\omega(\mathbf{x}) = \sum_{o \in \mathcal{O}[\omega]} \sum_{h_i^o \in \mathsf{Basis}[o]} w_i^o h_i^o(\mathbf{x}[\mathbf{T}_{o,i}]).$$

As for any linear approximation to the value function, the LP approach can be adapted to use this value function representation [Schweitzer & Seidmann, 1985]. Our LP variables are now the coefficients of our object basis functions for each object:

$$\{w_i^o \mid \forall h_i^o \in \mathsf{Basis}[o],\ \forall o \in \mathcal{O}[\omega]\}. \tag{13.1}$$

In our Freecraft example, there will be one LP variable for each joint assignment of F_1.*Health* and E_1.*Health* to represent the components of $\mathcal{V}_{\mathit{Footman1}}$. Similar LP variables will be included for the components of $\mathcal{V}_{\mathit{Footman2}}$, $\mathcal{V}_{\mathit{Enemy1}}$, and $\mathcal{V}_{\mathit{Enemy2}}$.

Constraints: As before, we have a constraint for each global state $\mathbf{x}$ and each global action $\mathbf{a}$:

$$\sum_{o \in \mathcal{O}[\omega]} \sum_{h_i^o \in \mathsf{Basis}[o]} w_i^o h_i^o(\mathbf{x}[\mathbf{T}_{o,i}]) \geq \\ \sum_{o \in \mathcal{O}[\omega]} R^o(\mathbf{x}[\mathbf{X}_o], \mathbf{a}[\mathbf{A}_o]) + \gamma \sum_{\mathbf{x}'} P_\omega(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_{o \in \mathcal{O}[\omega]} \sum_{h_i^o \in \mathsf{Basis}[o]} w_i^o h_i^o(\mathbf{x}'[\mathbf{T}_{o,i}{}']). \tag{13.2}$$

Objective function: Finally, our objective function is to minimize:

$$\sum_{o \in \mathcal{O}[\omega]} \sum_{h_i^o \in \mathsf{Basis}[o]} w_i^o \sum_{\mathbf{t}_o \in \mathbf{T}_o} \alpha_o(\mathbf{t}_o) h_i^o(\mathbf{t}_o), \tag{13.3}$$

where the *object state relevance weights* α_o are simply:

$$\alpha_o(\mathbf{t}_o) = \sum_{\mathbf{x} \sim [\mathbf{t}_o]} \alpha_\omega(\mathbf{x}),$$

and α_ω are the state relevance weights for Π_ω.

This transformation has the effect of reducing the number of free variables in the LP to n (the number of objects) times the number of basis functions in each object value subfunction. However, we still have a constraint for each global state and action, an exponentially-large number. As described in the previous chapter, by using our RMDP formulation, the MDP associated with each world in our domain is represented compactly by a factored MDP. The structure of the DDN representing the process dynamics is often highly factored, defined via local interactions between objects. Similarly, the value functions are local, involving only single objects or groups of closely related objects. Thus, we can use our factored LP decomposition technique to obtain the coefficients of the object-based value function. Often, the induced width of the underlying factored LP in such problems is quite small, allowing our techniques to be applied very efficiently. This induced width depend both on the structure of the relational MDP, and on the values of the relations in the particular world ω. Thus, it is possible that a compact relational MDP may be instantiated into a highly connected world, with large induced width. In such cases, we may exploit context-specific structure, if possible, or need to use additional approximation steps, such as the approximate factorization proposed in Chapter 6 and the future directions discussed in Section 14.2.3.

13.2.2 Class-based LP formulation

In the previous section, we show how our factored algorithms can be applied to optimize the object-based value function for a single ground MDP Π_ω. However, in order to generalize to new worlds, we must optimize the class-level parameters for $\mathcal{V}_C$ for the entire meta MDP $\Pi_{\texttt{meta}}$.

Variables: We can address the problem of optimizing the class-level value function by using a similar LP solution to the one we used for a single world. The variables in the *class-based linear program* are simply the weights of the class basis functions:

$$\{w_i^C \mid \forall h_i^C \in \mathsf{Basis}[C],\ \forall C \in \mathcal{C}\}. \tag{13.4}$$

In our example, there will be one LP variable for each joint assignment of Footman.*Health* and Enemy.*Health* to represent the components of $\mathcal{V}_{\mathsf{Footman}}$ for the footman class. Similar LP variables will be included for the components of $\mathcal{V}_{\mathsf{Enemy}}$. In the $2vs2$ world, the basis functions for *Footman1* and *Footman2* will use the parameters in $\mathcal{V}_{\mathsf{Footman}}$, and the ones for *Enemy1* and *Enemy2* will use the parameters in $\mathcal{V}_{\mathsf{Enemy}}$.

Constraints: Recall that our object-based LP formulation in Equation (13.2) for world ω had a constraint for each state $\mathbf{x} \in \mathbf{X}_\omega$ and each action vector $\mathbf{a} \in \mathbf{A}_\omega$ in this world. In the generalized solution, the state space is the union of the state spaces of all possible worlds, plus the initial state $\mathbf{x}_0$. Our constraint set for $\Pi_{\texttt{meta}}$ will, therefore, be a union of constraint sets, one for each world ω, each with its own actions:

$$\forall \omega \in \Omega, \;\; \forall \mathbf{x} \in \mathbf{X}_\omega, \; \forall \mathbf{a} \in \mathbf{A}_\omega \;\; :$$
$$\sum_{C \in \mathcal{C}} \sum_{h_i^C \in \mathsf{Basis}[C]} \sum_{o \in \mathcal{O}[\omega][C]} w_i^C h_i^C(\mathbf{x}[\mathbf{T}_{o,i}]) \geq$$
$$\sum_{o \in \mathcal{O}[\omega]} R^o(\mathbf{x}[\mathbf{X}_o], \mathbf{a}[\mathbf{A}_o]) + \gamma \sum_{\mathbf{x}'} P_\omega(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_{C \in \mathcal{C}} \sum_{h_i^C \in \mathsf{Basis}[C]} \sum_{o \in \mathcal{O}[\omega][C]} w_i^C h_i^C(\mathbf{x}'[\mathbf{T}_{o,i}']); \tag{13.5}$$

where the class-based value function for world ω is represented by:

$$\mathcal{V}_\omega(\mathbf{x}) = \sum_{C \in \mathcal{C}} \sum_{h_i^C \in \mathsf{Basis}[C]} \sum_{o \in \mathcal{O}[\omega][C]} w_i^C h_i^C(\mathbf{x}[\mathbf{T}_{o,i}]). \tag{13.6}$$

It is important to note that, as each world is represented by a factored MDP, and we can represent the constraints in Equation (13.5) compactly for each world using our LP decomposition technique.

In principle, we should have an additional constraint for the new state $\mathbf{x}_0$:

$$\mathcal{V}(\mathbf{x}_0) \geq R(\mathbf{x}_0) + \gamma \sum_{\omega, \mathbf{x} \in \mathbf{X}_\omega} P(\omega) P_\omega^0(\mathbf{x}) \mathcal{V}_\omega(\mathbf{x}), \tag{13.7}$$

where $R(\mathbf{x}_0) = 0$, and the value function for a world, $\mathcal{V}_\omega(\mathbf{x})$, is defined at the class level as in Equation (13.6). However, as Equation (13.7) is the only inequality involving $\mathcal{V}(\mathbf{x}_0)$, and the objective of our LP is to minimize (a weighted combination of) the values of the states, we can eliminate this constraint by defining $\mathcal{V}(\mathbf{x}_0)$ to have as its value the right hand side of Equation (13.7).

Objective function: The objective function of our class-based LP has the form:

$$\alpha(\mathbf{x}_0)\mathcal{V}(\mathbf{x}_0) + \sum_{\omega} \sum_{\mathbf{x} \in \mathbf{X}_\omega} \alpha(\omega, \mathbf{x}) \mathcal{V}_\omega(\mathbf{x}).$$

As before, we require that the state relevance weights α be positive and sum to 1. By substituting the definition of $\mathcal{V}(\mathbf{x}_0)$ from Equation (13.7), our objective function becomes:

$$\sum_{\omega,\mathbf{x}\in\mathbf{X}_\omega} \left[\alpha(\mathbf{x}_0)\gamma P(\omega)P^0_\omega(\mathbf{x}) + \alpha(\omega,\mathbf{x})\right] \mathcal{V}_\omega(\mathbf{x}).$$

To simplify this objective function, we assume that

$$\alpha(\mathbf{x}_0) = 1/2, \text{ and } \alpha(\omega,\mathbf{x}) = P(\omega)/2\cdot\alpha_\omega(\mathbf{x}),$$

for some set of *world-specific relevance weights* $\alpha_\omega(\mathbf{x}) > 0$, such that $\sum_{\mathbf{x}\in\mathbf{X}_\omega}\alpha_\omega(\mathbf{x}) = 1$. In this case, we can reformulate our objective as:

$$\sum_{\omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)/2[\gamma P^0_\omega(\mathbf{x}) + \alpha_\omega(\mathbf{x})]\mathcal{V}_\omega(\mathbf{x}).$$

Given the form of this objective, if $P^0_\omega(\mathbf{x}) > 0, \forall\mathbf{x}$, a particularly natural choice for the world-specific state relevance weights is: $\alpha_\omega(\mathbf{x}) = P^0_\omega(\mathbf{x})$. Using this choice of weights, which will continue to use in this chapter, the objective function becomes:

$$\text{Minimize:} \quad \frac{1+\gamma}{2}\sum_\omega P(\omega)\sum_{\mathbf{x}\in\mathbf{X}_\omega} P^0_\omega(\mathbf{x})\mathcal{V}_\omega(\mathbf{x});$$

or equivalently:

$$\text{Minimize:} \quad \frac{1+\gamma}{2}\sum_\omega P(\omega)\sum_{C\in\mathcal{C}}\sum_{h_i^C\in\text{Basis}[C]} w_i^C\,\alpha_i^C(\omega), \tag{13.8}$$

where the *class basis function relevance weights* $\alpha_i^C(\omega)$ for a world ω are given by

$$\alpha_i^C(\omega) = \sum_{o\in\mathcal{O}[\omega][C]}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P^0_\omega(\mathbf{x})h_i^C(\mathbf{x}[\mathbf{T}_{o,i}]). \tag{13.9}$$

In some cases, we can further simplify the definition of the class basis function relevance weight $\alpha_i^C(\omega)$. For example, if the initial state distribution is uniform, the basis functions are normalized to sum to one: $\sum_{\mathbf{t}_{o,i}\in\mathbf{T}_{o,i}} h_i^C(\mathbf{t}_{o,i}) = 1$ (*e.g.*, indicator basis functions), and the size of the domain of each basis function $|\mathbf{T}_{o,i}|$ is the same for all objects o of class C, then we can simplify Equation (13.9) as:

$$\alpha_i^C(\omega) = \frac{|\mathcal{O}[\omega][C]|}{|\mathbf{T}_{o,i}|};$$

where $|\mathcal{O}[\omega][C]|$ is the number of objects of class C in world ω.

In some models, the potential number of objects may be infinite, which could make the objective function unbounded. To prevent this problem, we assume that the probability $P(\omega)$ goes to zero sufficiently fast, as the number of objects tends to infinity. To understand this assumption, consider the following generative process for selecting worlds: first, the number of objects is chosen according to $P(\sharp)$; then, the classes and links of each object are chosen according to $P(\omega_\sharp \mid \sharp)$. Using this decomposition, we have that $P(\omega) = P(\sharp)P(\omega_\sharp \mid \sharp)$. The intuitive assumption described above can be formalized as:

Assumption 13.2.1 *The probability that a world ω has n objects is bounded by:*

$$P(\sharp = n) \leq \kappa_\sharp e^{-\lambda_\sharp n}, \;\; \forall n,$$

for some $\kappa_\sharp > 0$, and $\lambda_\sharp > 0$. ▮

If this assumption holds, the objective function becomes bounded, as the reward function grows linearly with the number of objects, while the probability of a world decays exponentially with this number. Note that the distribution $P(\sharp)$ over number of objects can be chosen arbitrarily, as long as it is bounded by some exponentially decaying function. If, for example, we choose $P(\sharp)$ to be an exponential distribution with parameter λ, then $\lambda_\sharp = \kappa_\sharp = \lambda$, and the expected number of objects in a world would be $1/\lambda$.

13.3 Sampling worlds

The main problem with the class-based LP formulation presented in the previous section is that the size of the LP — the size of the objective and the number of constraints — grows with the number of worlds, which, in most situations, grows exponentially with the number of possible objects, or may even be infinite. Furthermore, there may be worlds that are too large to solve, even with our factored approximation algorithms. Finally, this formulation would not fulfill our generalization goal, as we actually need to consider all possible worlds. A practical approach to address this problem is to *sample* some reasonable number of "small" worlds, and solve the LP for these worlds only. The resulting class-based value function can then be used for worlds that were not sampled, and even for worlds that are too large to solve with our factored planning algorithms.

A straightforward approach would be to sample worlds from the distribution $P(\omega)$. Unfortunately, this may lead us to sample very large worlds, albeit relatively low probability due to Assumption 13.2.1. To address this problem, we restrict our sampling to $P_{\leq n}(\omega)$, the distribution over worlds with at most n objects, which we define in the natural way:

$$P_{\leq n}(\omega) = \frac{P(\omega)}{\sum_{\omega' \in \Omega_{\leq n}} P(\omega')},\ \forall \omega \in \Omega_{\leq n}\ , \tag{13.10}$$

where Ω_i is the set of worlds with exactly i objects, and $\Omega_{\leq n} = \bigcup_{i=1}^{n} \Omega_i$ is the set of worlds with at most n objects.

We will start by sampling a set $\mathcal{D}_{\leq n}$ of m i.i.d. "small" worlds according to $P_{\leq n}(\omega)$. We can now define our LP in terms of the worlds in $\mathcal{D}_{\leq n}$, rather than all possible worlds. For each world ω in $\mathcal{D}_{\leq n}$, our LP will contain a set of constraints of the form presented in Equation (13.2). Note that in all worlds these constraints share the variables w_i^C that represent the weights of our class basis functions. The complete LP is given by:

Variables: $\{w_i^C \ \mid\ \forall h_i^C \in \text{Basis}[C],\ \forall C \in \mathcal{C}\};$

Minimize: $\frac{1+\gamma}{2m} \sum_{\omega \in \mathcal{D}_{\leq n}} \sum_{C \in \mathcal{C}} \sum_{h_i^C \in \text{Basis}[C]} w_i^C\ \alpha_i^C(\omega);$

Subject to: $\forall \omega \in \mathcal{D}_{\leq n},\ \forall \mathbf{x} \in \mathbf{X}_\omega,\ \forall \mathbf{a} \in \mathbf{A}_\omega\ :$

$$\begin{aligned}&\sum_{C \in \mathcal{C}} \sum_{h_i^C \in \text{Basis}[C]} \sum_{o \in \mathcal{O}[\omega][C]} w_i^C h_i^C(\mathbf{x}[\mathbf{T}_{o,i}]) \geq \\ &\quad \sum_{o \in \mathcal{O}[\omega]} R^o(\mathbf{x}[\mathbf{X}_o], \mathbf{a}[\mathbf{A}_o]) + \\ &\qquad \gamma \sum_{\mathbf{x}'} P_\omega(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_{C \in \mathcal{C}} \sum_{h_i^C \in \text{Basis}[C]} \sum_{o \in \mathcal{O}[\omega][C]} w_i^C h_i^C(\mathbf{x}'[\mathbf{T}_{o,i}']);\end{aligned} \tag{13.11}$$

where, by using our sampled worlds, the objective function in Equation (13.8) is approximated by: $\frac{1+\gamma}{2m} \sum_{\omega \in \mathcal{D}_{\leq n}} \sum_{C \in \mathcal{C}} \sum_{h_i^C \in \textsf{Basis}[C]} w_i^C \ \alpha_i^C(\omega)$. Our complete LP-based approximation algorithm for computing the class-based value function over the sampled worlds is summarized in Figure 13.1.

The solution obtained by the LP with sampled worlds will, in general, not be equal to the one obtained if all worlds are considered. However, we can show that the quality of the two approximations is close, if a sufficient number of worlds are sampled. Specifically, with a *polynomial* number of sampled worlds, we can guarantee that, with high probability, the quality of the value function approximation obtained when sampling worlds is close to the one obtained when considering all possible (unboundedly-large) worlds. In order to prove this result we need two additional assumptions:

Assumption 13.3.1 *The magnitude of each basis function h_i^C is normalized to 1:*

$$\left\| h_i^C \right\|_\infty \leq 1, \ \ \forall h_i^C \in \textsf{Basis}[C], \ \forall C \in \mathcal{C}.$$

Further, we assume that the weights of our basis functions are bounded by:

$$\left| w_i^C \right| \leq \frac{R^o_{max}}{1-\gamma}, \ \ \forall h_i^C \in \textsf{Basis}[C], \ \forall C \in \mathcal{C}. \ \ ∎$$

These assumptions guarantee that each $w_i^C h_i^C$ has a bounded magnitude, which is necessary to guarantee that the space of class-based value function templates is bounded. Note that we are not assuming a bound on the instantiation of this class-based value function in a world, on the contrary, our theoretical results will hold even in unboundedly-large worlds, where this instantiation will also be unbounded. The assumption on the magnitude of the basis functions can be guaranteed by appropriate construction. The bound on the basis function weights can be enforced by using additional constraints in our LP, though the result of this constrained problem may be suboptimal in the original one. However, in practice, the results of our algorithm usually satisfy this bound, without additional LP constraints, even when we sample worlds.

Under this assumption, we prove the following bound on the quality of our class-based LP:

```
ClassBasedLPA ($P^C$, $R^C$, $\gamma$, $H^C$, $\mathcal{D}_{\leq n}$, $\mathcal{O}^\omega$, $\alpha$)
        // $P^C$ is the class-based transition model.
        // $R^C$ is the set of class-based reward functions.
        // $\gamma$ is the discount factor.
        // $H^C$ is the set of class basis functions $H^C = \{h_i^C \mid \forall h_i^C \in \mathsf{Basis}[C],\ \forall C \in \mathcal{C}\}$.
        // $\mathcal{D}_{\leq n}$ is a set of sampled worlds.
        // $\mathcal{O}^\omega$ stores the elimination order for each sampled world $\omega \in \mathcal{D}_{\leq n}$.
        // $\alpha$ are the class basis functions relevance weights as defined in Equation (13.9).
        // Return the class basis function weights $\{\mathbf{w}^C\}_{C\in\mathcal{C}}$ computed by our linear programming-based
          approximation over the sampled worlds.
    // Generate linear programming-based approximation constraints for each sampled world.
    For sampled world $\omega \in \mathcal{D}_{\leq n}$:
        // Compute backprojection of basis functions for this world.
        For each class $C$; for each basis function in this class $h_i^C \in \mathsf{Basis}[C]$;
          for each object of this class in the world $o \in \mathcal{O}[\omega][C]$:
            Let $g_i^o = \mathit{Backproj}_\omega(h_i^C(\mathbf{T}_{o,i}))$.
        // Generate linear programming constraints for this world.
        Let $\Omega_\omega = \text{FactoredLP}(\{(\gamma g_i^o - h_i^o) \mid \forall h_i^o \in \mathsf{Basis}[o],\ \forall o \in \mathcal{O}[\omega]\}, R^\omega, \mathcal{O}^\omega)$.
        // So far, our constraints guarantee that

          $$\phi_\omega \geq R^\omega(\mathbf{x},a) + \gamma \sum_{\mathbf{x}'} P^\omega(\mathbf{x}' \mid \mathbf{x},a) \sum_{o\in\mathcal{O}[\omega]} \sum_{h_i^o\in\mathsf{Basis}[o]} w_i^o h_i^o(\mathbf{x}') - \sum_{o\in\mathcal{O}[\omega]} \sum_{h_i^o\in\mathsf{Basis}[o]} w_i^o h_i^o(\mathbf{x});$$

          to satisfy the linear programming-approximation solution in (13.11) for world $\omega$, we must
          add a constraint.
        Let $\Omega_\omega = \Omega_\omega \cup \{\phi_\omega = 0\}$.
        // Finally, we must introduce a set of equality constraints that ensure that objects of the
          same class have the same global class basis function coefficients.
        For each class $C$; for each basis function in this class $h_i^C \in \mathsf{Basis}[C]$;
          for each object of this class in the world $o \in \mathcal{O}[\omega][C]$:
            Let $\Omega_\omega = \Omega_\omega \cup \{w_i^o = w_i^C\}$.
    // We can now obtain the weights of the class basis functions by solving an LP.
    Let $\{\mathbf{w}^C\}_{C\in\mathcal{C}}$ be the solution of the linear program:

                Minimize:      $\sum_{\omega\in\mathcal{D}_{\leq n}} \sum_{C\in\mathcal{C}} \sum_{h_i^C\in\mathsf{Basis}[C]} w_i^C\ \alpha_i^C(\omega)$;
                Subject to:    $\Omega_\omega,\ \forall\omega \in \mathcal{D}_{\leq n}$.

    Return $\{\mathbf{w}^C\}_{C\in\mathcal{C}}$.
```

Figure 13.1: Factored class-based LP-based approximation algorithm to obtain a generalizable value function.

Theorem 13.3.2 *Consider the following class-based value functions (each with k parameters): $\widehat{\mathcal{V}}$ obtained from the LP over all possible worlds Ω by minimizing Equation (13.8) subject to the constraints in Equation (13.5); and $\widetilde{\mathcal{V}}$ obtained by solving the class-level LP in (13.11) with constraints only for a set $\mathcal{D}_{\leq n}$ of m worlds sampled from $P_{\leq n}(\omega)$, i.e., only sampled from the set of worlds $\Omega_{\leq n}$ with at most n objects, where*

$$n = \left\lfloor \frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp} \right\rfloor .$$

Let $\mathcal{V}^*$ be the optimal value function of the meta-MDP $\Pi_{\mathtt{meta}}$ over all possible worlds Ω. For any $\delta > 0$ and $\varepsilon > 0$, for a number of sampled worlds m polynomial in $(k, \frac{1}{1-\gamma}, \frac{1}{\varepsilon}, \ln\frac{1}{\delta})$, the error introduced by sampling worlds is bounded by:

$$\left\|\widetilde{\mathcal{V}} - \mathcal{V}^*\right\|_{1,P_\Omega} \leq \left\|\widehat{\mathcal{V}} - \mathcal{V}^*\right\|_{1,P_\Omega} + 18\varepsilon \frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp} \frac{R^o_{max}}{1-\gamma} \frac{\kappa_\sharp}{\lambda_\sharp},$$

with probability at least $1-\delta$, where $\|\mathcal{V}\|_{1,P_\Omega} = \sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)P^0_\omega(\mathbf{x})\left|\mathcal{V}_\omega(\mathbf{x})\right|$, and R^o_{max} is the maximum per-object reward.

Proof: *See Appendix A.5.* ▮

Our theorem states that if we sample a polynomial number of "small" worlds with at most $\left\lfloor \frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp} \right\rfloor$ objects, independently of the number of states or actions, we obtain an approximation to the optimal value function of the meta MDP that is close to the one we would have obtained had we considered all possible (unboundedly-large) worlds in our optimization. If, for example, we again choose $P(\sharp)$ to be an exponential distribution, then $\left\lfloor \frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp} \right\rfloor$ would lead us to sample worlds with a number of objects that is no larger than $\ln\left(\frac{1}{\varepsilon}\right)$ times the expected number of objects in our domain.

The proof uses some of the techniques developed by de Farias and Van Roy [2001b] for analyzing constraint sampling in general MDPs. However, there are some important differences: First, our analysis includes the error introduced when sampling the objective function, which is approximated by a sum only over a sampled subset of "small" worlds rather than over all worlds as in the LP for the full meta-MDP. This issue was not previously addressed. Second, and more important, the algorithm of de Farias and Van Roy relies on the assumption that constraints are sampled according to some “ideal” distribution (the product of a Lyapunov function with the stationary distribution of the optimal policy). In our algorithm, after each world is sampled according to $P_{\leq n}(\omega)$, our algorithm exploits the factored structure in the model to represent the constraints exactly, in closed-form, avoiding the dependency on the “ideal” distribution. Finally, the number of samples in the result of de Farias and Van Roy [2001b] depends on the number of actions in the MDP, which is exponential in multiagent problems. They also present an equivalent formulation where the state space is augmented with a state variable to indicate the choice of each action variable. At every time step, the agent then sets one of these state variables. The number of actions in this modified formulation is now equal to the size of the domain of each action variable, and the theoretical scaling of the number of samples now depends on the log of the number of joint actions, but multiplies the size of the state space by the number of joint actions. The increased number of states will probably increase the number of basis functions needed for a good approximation. Our factored LP decomposition technique allows us to prove a result that has no dependency on the number of actions when each world is represented as a factored MDP. Appendix A.5 also presents a more general (and tighter) version of our result, where in addition to picking ε and δ, the maximum number of objects n can be picked arbitrarily.

13.4 Learning classes of objects

The definition of a class-based value function assumes that all objects in a class have the same local value subfunction. Specifically, our class-based representation forces every object o of a particular class C to have the same class basis function coefficient in every world:

$$w_i^o = w_i^{o'} = w_i^C,\ \forall o, o' \in \mathcal{O}[\omega][C],\ \forall h_i^C \in \mathsf{Basis}[C],\ \forall \omega.$$

However, in many cases, even objects in the same class might play different roles in the model, and therefore have a different impact on the overall value. For example, if only one peasant has the capability to build barracks, his status may have a greater impact. Thus, we may often need to distinguish objects into subclasses. Distinctions of this type are not usually known in advance, but are learned by an agent as it gains experience with a domain and detects regularities.

We propose a procedure that takes exactly this approach to find potential subclasses for each class: Assume that we have been presented with a set $\mathcal{D}$ of worlds. For each world $\omega \in \mathcal{D}$, an approximate value function

$$\mathcal{V}_\omega = \sum_{o \in \mathcal{O}[\omega]} \sum_{h_i^o \in \mathsf{Basis}[o]} w_i^o h_i^o$$

is computed as described in Section 13.2.1. If every object o of class C ($o \in \mathcal{O}[\omega][C]$) is similar, then they must have very similar coefficients w_i^o in every world in $\mathcal{D}$. Otherwise, we need a procedure to split C into subclasses C', C'', etc, such that objects in each subclass have similar coefficients.

In order to differentiate objects into subclasses, we assume that each object in a world is associated with a set of class-based features $\mathcal{F}_\omega^C[o]$. For example, the features may include local information, such as whether the object is a peasant linked to a barrack or not, as well as global information, such as whether this world contains archers in addition to footmen. We use these features, along with the basis function coefficients w_i^o, to differentiate objects of class C into one of the subclasses.

Specifically, we can define our "training data" $\mathcal{D}^C$, for each class C, as

$$\left\{ \left\langle \mathcal{F}_\omega^C[o], \mathbf{w}^o \right\rangle \;:\; \forall o \in \mathcal{O}[\omega][C],\ \forall \omega \in \mathcal{D} \right\},$$

where $\mathbf{w}^o$ is a vector of basis function weights for object o whose ith component is w_i^o. We now have a well-defined learning problem: given this training data, we would like to partition the objects of class C into subclasses, such that objects of the same subclass have similar coefficients w_i^o for each basis function h_i^o in the object value subfunction. Note that this is not a standard learning task, we would like to find a rule to describe objects that have similar coefficients, but we will not use these coefficients in our class-level value function. Once the subclass definitions are obtained, the specific (sub)class coefficients are optimized using our class-level LP.

There are many approaches for tackling our learning task. For each class C, we choose

1. **Learning Subclasses:**
   - **Input:**
     - A set of training worlds $\mathcal{D}$.
     - A set of features $\mathcal{F}_{\omega}^{C}[o]$.
   - **Algorithm:**
     (a) For each $\omega \in \mathcal{D}$, compute an object-based value function, as described in Section 13.2.1.
     (b) For each class C: Apply regression tree learning on
     $$\left\{\left\langle \mathcal{F}_{\omega}^{C}[o], \mathbf{w}^{o}\right\rangle \;:\; \forall o \in \mathcal{O}[\omega][C],\; \forall \omega \in \mathcal{D}\right\}.$$
     (c) Define a subclass of class C for each leaf, characterized by the feature vector associated with its path.
2. **Computing Class-Based Value Function:**
   - **Input:**
     - A set of (sub)class definitions $\mathcal{C}$.
     - A template for $\{\mathcal{V}_C = \sum_{h_i^C \in \mathsf{Basis}[C]} w_i^C h_i^C : C \in \mathcal{C}\}$.
     - A set of training “small” worlds $\mathcal{D}_{\leq n}$ with at most n objects.
   - **Algorithm:**
     (a) Compute the parameters $\{\mathbf{w}^C : C \in \mathcal{C}\}$ that optimize the LP in Equation (13.11) relative to the worlds in $\mathcal{D}_{\leq n}$.
3. **Acting in a New World:**
   - **Input:**
     - A set of class value subfunctions $\{\mathcal{V}_C : C \in \mathcal{C}\}$.
     - A set of (sub)class definitions $\mathcal{C}$.
     - Any world ω.
   - **Algorithm:** Repeat
     (a) Obtain the current state $\mathbf{x}$.
     (b) Determine the appropriate class C for each $o \in \mathcal{O}[\omega]$ according to its features.
     (c) Define $\mathcal{V}_{\omega}$ according to Equation (13.12).
     (d) Use the coordination graph algorithm to compute an action $\mathbf{a}$ that maximizes $R^{\omega}(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P^{\omega}(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \mathcal{V}_{\omega}(\mathbf{x}')$.
     (e) Take action $\mathbf{a}$ in the world.

Figure 13.2: The overall generalization algorithm.

to use decision tree regression [Breiman *et al.*, 1984], so as to construct a tree that predicts the basis function coefficients given the features. Thus, each split in the tree corresponds to a feature in $\mathcal{F}_\omega^C[o]$; each branch down the tree defines a subset of the objects of class C whose feature values are as defined by the path; the leaf at the end of the path contains the average coefficients for this set of objects. We use a squared error criteria to guarantee that objects in a leaf have similar coefficients. As the regression tree learning algorithm tries to construct a tree that is predictive about the basis function coefficients, it will aim to construct a tree where the mean at each leaf is very close to the training data assigned to that leaf. Thus, the leaves tend to correspond to objects in C whose basis function coefficients are similar. We can thus take the leaves in the tree to define our subclasses, where each subclass is characterized by the combination of feature values specified by the path to the corresponding leaf. This algorithm is summarized in Step 1 of Figure 13.2. Note that the mean subfunction at a leaf is not used as the value subfunction for the corresponding class; rather, the parameters of the value subfunction are optimized using the class-based LP in Step 2 of the algorithm. We present a case study of this algorithm in Section 13.5.1.

Once we have our subclass definitions, we define the class-based value function as in Equation (12.9):

$$\mathcal{V}_\omega(\mathbf{x}) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(\mathbf{x}[\mathbf{T}_{o,i}]). \tag{13.12}$$

However, our set of classes $\mathcal{C}$ now includes all subclasses of each class C, and the class of each object o is now the subclass whose branch is consistent with the features of this object $\mathcal{F}_\omega^C[o]$.

13.5 Empirical evaluation

In this section, we present empirical evaluations of our generalization algorithm on two domains: First, we use the multiagent SysAdmin domain to evaluate the scaling properties of our approach, and the effect of learning subclasses on the quality of our policies. Then, we present results on the actual Freecraft game. Here, we evaluate the ability of our algorithm to generalize to problems that are significantly larger than our planning algorithms could address.

13.5.1 Computer network administration

We first experimented with the multiagent SysAdmin problem described in Example 8.1.1. In this problem, we have a single class Comp to represent computers in the network. This class is associated with two state variables $\mathcal{X}[\mathsf{Comp}] = \{\mathsf{Comp}.\mathit{Status}, \mathsf{Comp}.\mathit{Load}\}$, where

$$\begin{aligned} \mathrm{Dom}[\mathsf{Comp}.\mathit{Status}] &= \{\mathit{good}, \mathit{faulty}, \mathit{dead}\}, \text{ and} \\ \mathrm{Dom}[\mathsf{Comp}.\mathit{Load}] &= \{\mathit{idle}, \mathit{loaded}, \mathit{process\ successful}\}. \end{aligned}$$

Each object of the Comp class is also associated with an action variable $\mathcal{A}[\mathsf{Comp}] = \{\mathsf{Comp}.A\}$, where $\mathrm{Dom}[\mathsf{Comp}.A] = \{\mathit{reboot}, \mathit{not\ reboot}\}$. Each object of class Comp has a single set link $\mathcal{L}[\mathsf{Comp}] = \{\mathit{Neighbors}\}$, such that

$$\rho[\mathsf{Comp}.\mathit{Neighbors}] = \mathsf{SetOf}\{\mathsf{Comp}\},$$

i.e., every computer is linked to a set of other computers.

The class transition probabilities for the status variable are described as follows:

$$P^{\mathsf{Comp}.\mathit{Status'}}\left(\mathsf{Comp}.\mathit{Status'} \mid \mathsf{Comp}.\mathit{Status}, \mathsf{Comp}.A, \sharp\left(\mathsf{Comp}.\mathit{Neighbors}.\mathit{Status} = \mathit{Dead}\right)\right),$$

that is, the status of a machine in the next time step depends on its status in the current time step, on the action of its administrator (rebooting causes the machine to be good with probability 1), and on the number of neighbors that are dead, as a dead machine increases the probability that its neighbors will become faulty and eventually die. In our experiments, we use a noisy-or to represent this relationship, where each neighbor has the same noise parameters [Pearl, 1987].

The class transition model for the load variable is simply:

$$P^{\mathsf{Comp}.\mathit{Load'}}\left(\mathsf{Comp}.\mathit{Load'} \mid \mathsf{Comp}.\mathit{Load}, \mathsf{Comp}.\mathit{Status}, \mathsf{Comp}.A\right),$$

as processes take longer to terminate when a machine is faulty, and are lost when the machine dies or the administrator decides to reboot it.

The system receives a reward of 1 if a process terminates successfully. Thus, the class reward template is simply:

$$R^{\mathsf{Comp}}(\mathsf{Comp}.\mathit{Load'}) = \mathbb{1}\left(\mathsf{Comp}.\mathit{Load} = \mathit{process\ successful}\right).$$

A world in this problem is defined by a number of computers and a network topology that defines the objects in $\mathsf{Comp}.\mathit{Neighbors}$. For a world ω with n machines, the number of states in the MDP Π_ω is 9^n and the joint action space contains 2^n possible actions, *e.g.*, a problem with 30 computers has over 10^{28} states and a billion possible actions. We use a discount factor γ of 0.95.

The formulation of our class basis functions was based on the "pair" basis defined in Section 9.3. Each object of class Comp is associated with two sets of basis functions: The first set contains an indicator function over each joint assignment of $\mathsf{Comp}.\mathit{Status}$ and $\mathsf{Comp}.\mathit{Load}$. The second set includes indicators over $\mathsf{Comp}.\mathit{Status}$ and $\mathsf{Comp'}.\mathit{Status}$, for each $\mathsf{Comp'} \in \mathsf{Comp}.\mathit{Neigbourghs}$.

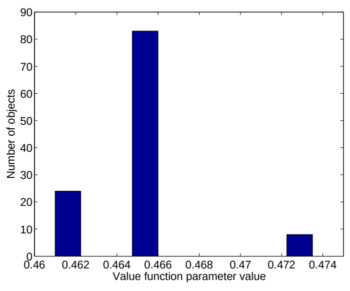
For this problem, we implemented our class-based LP generalization algorithm described in Chapter 13 in Matlab, using CPLEX as the LP solver. Rather than using the full LP decomposition presented in Chapter 4, we used the constraint generation extension proposed in by Schuurmans and Patrascu [2001], described in Section 4.5, as the memory requirements were lower for this second approach.

We first tested the extent to which value functions are shared across objects. In Figure 13.3(a), we plot the value each object gave to the assignment to the indicator basis function $\mathbb{1}(\mathsf{Comp}.\mathit{Status} = \mathit{working})$, for instances of the 'three legs' topology. Clearly, these values cluster into three classes. This is the type of structure that we can extract with our subclass learning algorithm in Section 13.4. We used $CART^{®}$ to learn decision trees for our class partition. Our training data $\mathcal{D}^{\mathsf{Comp}}$ should be of the form:

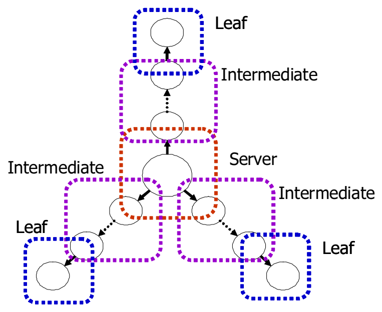
$$\left\{\left\langle \mathcal{F}_\omega^{\mathsf{Comp}}[o], \mathbf{w}^o \right\rangle \,:\, \forall o \in \mathcal{O}[\omega][\mathsf{Comp}],\ \forall \omega \in \mathcal{D}\right\},$$

where $\mathcal{F}_\omega^{\mathsf{Comp}}[o]$ is some set of features evaluated for object o in world ω.

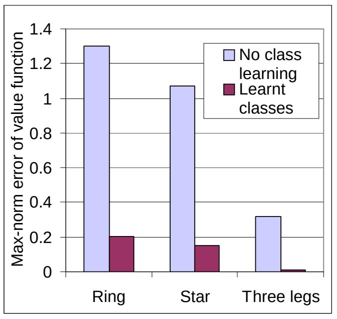
In our 'three legs' network example, we associated each instance of class Comp with a single feature $d(o, \omega)$ that measures the number of hops from the center of the network to



(a)



(b)



(c)

Figure 13.3: Results of learning subclasses for the multiagent SysAdmin problem: (a) training data; (b) classes learned for 'three legs'; (c) advantage of learning subclasses.

computer o. For this particular case, the learning algorithm partitioned the computers into three subclasses illustrated in Figure 13.3(b). Intuitively, we name these subclasses Server, Intermediate, and Leaf. In Figure 13.3(a), we see that the basis function coefficient for the class Server (third column) has the highest value, because a broken server can cause a chain reaction affecting the whole network, while the coefficient of the class Leaf (first column) is lowest, as it cannot affect any other computer.

We then evaluated the generalization quality of our class-based value function by comparing its performance to that of planning specifically for a new environment. For each topology, we computed the class-based value function with 5 sampled networks of up to 20 computers. We then sampled a new larger network of size 21 to 32, and computed for it a value function that used the same factorization, but with no class restrictions. This value function has more parameters – different parameters for each object, rather than for entire classes. These parameters are optimized for each particular network. This process was repeated for 8 sets of networks.

First, we wanted to determine if our procedure for learning classes yields better approximations than the ones obtained from the default classes. Figure 13.3(c) compares the max-norm error between our class-based value function and the one obtained by replanning in each domain, without any class restrictions. The graph suggests that, by learning classes using our decision tree regression procedure, we obtain a much better approximation of the value function than we would have, had we replanned.

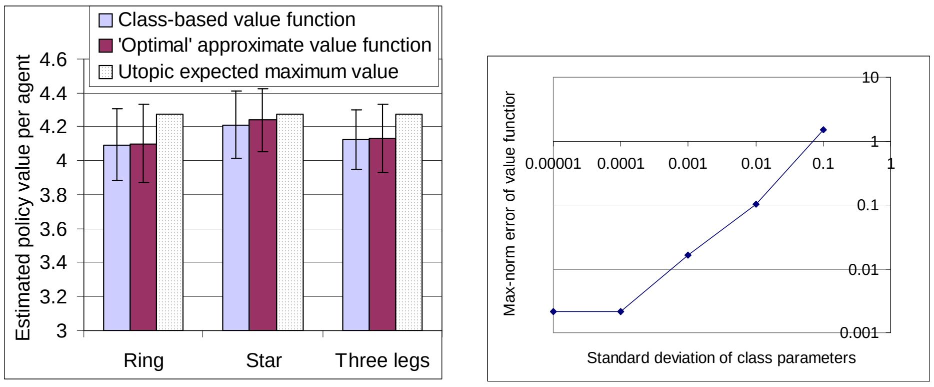


Figure 13.4: Generalization results for the multiagent SysAdmin problem: (a) generalization quality (evaluated by 20 Monte Carlo runs of 100 steps); (b) adding noise to instantiated object parameters.

Next, we evaluate the quality of the greedy policies obtained from our class-level value function, as compared to replanning in each world. The results, shown in Figure 13.4(a), indicate that the value of the policy from the class-based value function is very close to the value of replanning, suggesting that we can generalize well to new problems. We also computed a utopic upper bound on the *expected* value of the optimal policy by removing the (negative) effect of the neighbors on the status of the machines. Although this bound is loose, our approximate policies still achieve a value close to the bound, indicating that our generalized policies are near-optimal for these problems.

In practice, objects may not have exactly the same transition model as the one defined by the class template. To evaluate the effects of such uncertainty, we used a hierarchical Bayes approach. Thus, rather than giving each object the same transition probabilities as the class, we sampled the parameters of each object independently from a class Dirichlet distribution whose mean is determined by the class parameter. Figure 13.4(b) shows the error between our class-based approximation versus the value function we obtain for replanning with the *particular instantiated objects*, without class restriction. Note, the error grows linearly in a log-log scale, that is, only polynomially with the standard deviation of the Dirichlet, indicating that our approach is robust to such noise.

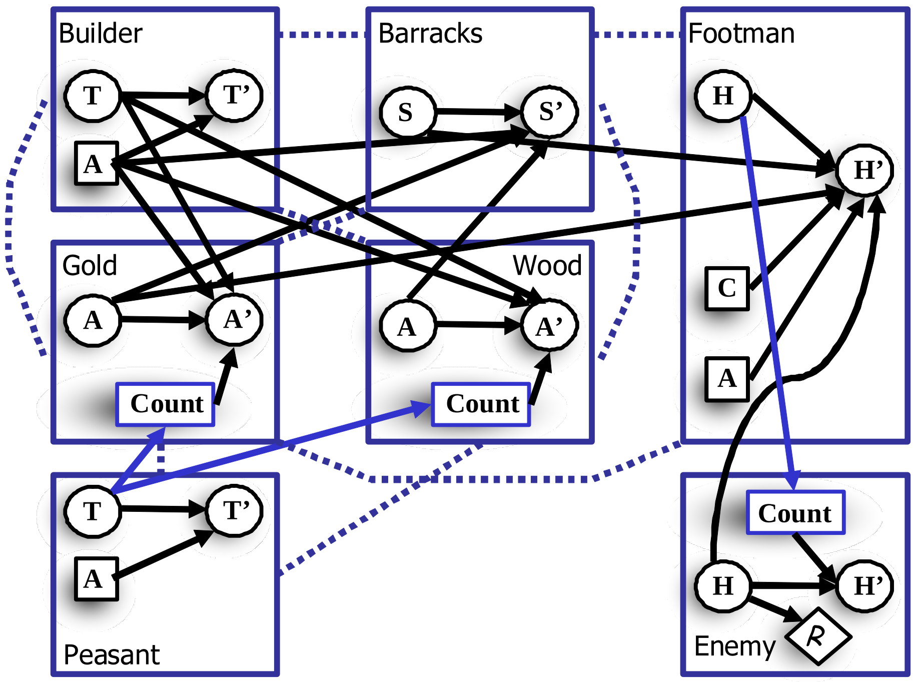


Figure 13.5: RMDP schema for Freecraft.

13.5.2 Freecraft

We also evaluated the quality of our class-based approximations on the actual Freecraft game. For this evaluation, we implemented our methods in C++ and used CPLEX as the LP solver. We created two tasks, which assess our policies in two different aspects of the game: *strategic domain* – evaluating long-term strategic decision making, and *tactical domain* – testing coordination in local tactical battle maneuvers. Our Freecraft interface, and scenarios for these and other more complex tasks are publicly available at:

http://dags.stanford.edu/Freecraft/ .

For each task we designed an RMDP model to represent the system by consulting "domain experts". In this model, we have 6 classes:

$$\mathcal{C} = \{\text{Peasant}, \text{Wood}, \text{Gold}, \text{Barrack}, \text{Footman}, \text{Enemy}\}.$$

The state and action variables of each class are:

$$\begin{aligned}
&\mathcal{X}[\textsf{Peasant}] = \{\mathit{Task}\}, \quad && \mathcal{A}[\textsf{Peasant}] = \{\mathit{Action}\}.\\
&\mathcal{X}[\textsf{Gold}] = \{\mathit{Amount}\}. && \\
&\mathcal{X}[\textsf{Wood}] = \{\mathit{Amount}\}. && \\
&\mathcal{X}[\textsf{Barrack}] = \{\mathit{Status}\}. && \\
&\mathcal{X}[\textsf{Footman}] = \{\mathit{Health}\}, && \mathcal{A}[\textsf{Footman}] = \{\mathit{Create}, \mathit{Attack}\}.\\
&\mathcal{X}[\textsf{Enemy}] = \{\mathit{Health}\}. &&
\end{aligned}$$

An object of the class Peasant can have one of 4 tasks,

$$\mathrm{Dom}[\textsf{Peasant}.\mathit{Task}] = \{\mathit{Waiting}, \mathit{Harvesting}, \mathit{Mining}, \mathit{Building}\},$$

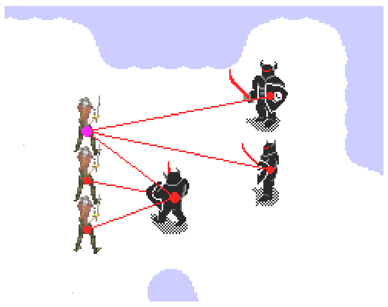
and one of 4 actions: $\mathrm{Dom}[\textsf{Peasant}.\mathit{Action}] = \{\mathit{Wait}, \mathit{Harvest}, \mathit{Mine}, \mathit{Build}\}$. At every time step, a peasant's task will be set according to its action, with high probability.

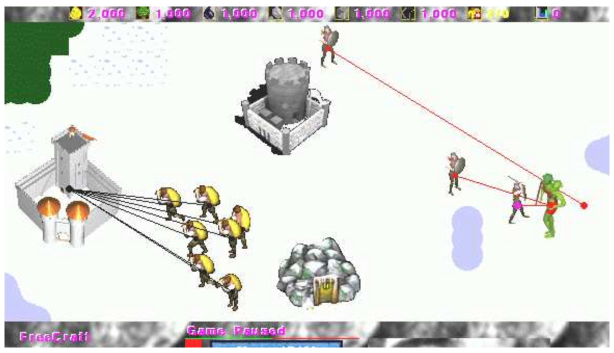
The amount of gold or wood is discretized into 3 levels. The value of Gold.*Amount* at each time step increases with a probability that depends monotonically on the number of peasants whose task is *Mining*. The class transition model for Wood is analogous.

In our model, the status of a barrack takes one of 2 values: $\{\mathit{Unbuilt}, \mathit{Built}\}$. A barrack will transition from unbuilt to built with high probability, if enough gold and wood are available, and the task of a peasant linked to this barrack is *Building*.

The state variable Footman.*Health* is discretized into 5 "health points", a footman with no health points is considered "dead". If a dead footman takes action Footman.*Create*, a barrack is built, and there is enough gold, this footman's health points will be set to its maximum level. At every time step, a footman's health points may decrease, if it is attacked by an enemy who is not dead. In addition, the footman's second action variable, Footman.*Attack*, is used to select which enemy this footman is attacking in the next time step, as described in Example 12.2.2.

Objects of class Enemy are described similarly to those of class Footman. The state variable Enemy.*Health* is discretized into 5 "health points". At every time step, an enemy's health points may decrease with a probability that increases monotonically with the number of footmen whose Footman.*Attack* action variable selects this enemy. In our Freecraft


(a)



(b)

Figure 13.6: Freecraft problem domains: (a) tactical; (b) strategic.

model, only the Enemy class is associated with a reward function:

$$R^{\mathsf{Enemy}} = \mathbb{1}\left(\mathsf{Enemy}.\mathit{Health} = \mathit{Dead}\right).$$

After solving a number of small problems, we learned that the Peasant class needed to be divided into 2 subclasses: Peasants that are linked to objects of the class Barrack, *i.e.*, peasants that can build a barrack, are defined to belong to class Builder, while other peasants are included in the standard Peasant class.

Figure 13.5 illustrates our complete relational MDP representation for the general Freecraft domain. We use this relational representation to obtain class-level value functions. After planning, our policies were evaluated on the actual game. To better visualize our results, we direct the reader to view videos of our policies at a website:

http://robotics.stanford.edu/∼guestrin/Research/Generalization/ .

This website also contains more details of our RMDP model. It is important to note that our policies were constructed relative to this very approximate model of the game, but evaluated against the real game.

Tactical domain: In the *tactical domain*, also described in Example 12.2.2, the goal is to take out an opposing enemy force with an equivalent number of units. At each time step, each footman decides which enemy to attack. The enemies are controlled using Freecraft's hand-built strategy. We modelled footmen and enemies as described above. To encourage coordination, each footman was also linked to a "buddy" in a ring structure.

Our class value function for Footman was defined by a set of indicators over the assignments of this footman's health, his buddy's health, and the health of the enemy that attacks this footman. Additionally, the class value function for Enemy was defined by a set of indicators over the assignments of this enemy's health, the health of the footman this enemy is attacking, and the health of the enemy that attacks this footman's buddy.

We solved this model for a world with 3 footmen and 3 enemies, illustrated in Figure 13.6(a). The resulting policy demonstrates successful coordination between our footmen: initially all three footmen focus on one enemy. When the enemy becomes injured, one footman switches its target. Finally, when the enemy is very weak, only one footman continues to attack it, while the other two tackle a different enemy. Our full policy is fairly complex, with action choices depending both on the state of the footmen and of the enemies. Using this policy, our footmen defeat the enemies in Freecraft.

The scope of our class value function for Footman includes the health of the enemy. Unfortunately, the scope of the backprojection of this function includes the *Health* and *Attack* variables of all footmen, as every footman can choose to attack this enemy. Thus, the size of the backprojection grows exponentially in the number of objects in the world. Therefore, we cannot solve large models using our standard factored LP approach. It may be possible to exploit CSI in this model, as the CPD of an enemy only depends on a footman in contexts where the footman's action chooses to attack this enemy. However, solving a factored MDP with this formulation requires extensions to the CSI methods presented in this thesis to address the aggregation in the enemy's CPD. Although some such extensions are discussed by Pfeffer [2000], we have not yet pursed them in the context of MDPs.

Fortunately, when executing a policy, we first instantiate the state at every time step. Thus, after instantiation, the factors become significantly smaller, and action selection is performed efficiently. Thus, even though we cannot execute Step 2 in Figure 13.2 of our algorithm for larger scenarios, we can generalize our class-based value function to a world with 4 footmen and enemies without replanning, using only Step 3 of our approach. The policy continues to demonstrate successful coordination between footmen, and we again beat Freecraft's policy. However, as the number of units increases, the position of enemies becomes increasingly important. Specifically, one of our footmen may choose to attack an enemy that is not close to the footman's current position. As our footman moves towards that enemy, it is attacked by other enemies along the way, wasting valuable health points. Currently, our model does not consider this feature, and in a world with 5 footmen and enemies, our policy loses to Freecraft in a close battle.

Strategic domain: The goal in the *strategic domain* is to kill a strong enemy. The player starts with a few peasants, who can collect gold or wood, or attempt to build a barrack, a task requiring both gold and wood. All resources are consumed after each *Build* action. With a barrack and gold, the player can train a footman. The footmen can choose to attack the enemy. When attacked, the enemy loses "health points", but fights back and may kill the footmen.

In addition to the standard links in our RMDP model, we included links between every peasant and a "central" object of the new subclass Builder defined above. Each footman was also linked to a "buddy" in a ring structure, as described above. Our local value subfunctions for each class were composed of indicators for the assignment of the state variables of each triple of linked objects in our model.

We solved a world with 2 peasants, 1 barrack, 2 footmen, and an enemy. The resulting policy for this instance of the strategic problem is quite interesting: the peasants gather gold and wood to build a barrack, then gold to build a footman. Rather than attacking the enemy at once, this footman waits until the peasants collect enough gold to build a second footman. Then, these two footmen attack the enemy together. Unfortunately, the stronger enemy is able to kill both of these footmen, becoming quite weak in the process. When the next footman is trained, rather than waiting for a second one, it attacks the now weak enemy, and is able to kill him.

As with the tactical domain, planning in large instances of the strategic domain is infeasible, as every peasant can influence the amount of the gold and wood. Fortunately, at every time step, every peasant's task and the amount of gold are observed. Thus, the instantiated Q-function is very compact, and action selection can be performed very efficiently. Therefore, we can use our generalized value function to tackle a world with 9 peasants and 3 footmen, without replanning.

Interestingly, the policy in the larger scenario is qualitatively different from the one in the smaller scenario: As before, the 9 peasants coordinate to gather resources, and build a barrack. However, rather than attacking with 2 footmen, as in the smaller scenario, the policy now waits for 3 footmen to be trained before attacking. The 3 footmen are able to kill the enemy, and only one of these footmen dies. This problem shows successful generalization from a problem with about 10^6 joint state-action pairs to one with over 10^{13} pairs.

13.6 Discussion and related work

We present an algorithm that is able to generalize plans to new environments represented by relational MDPs. Such a generalization has two complementary uses: First we can tackle new environments with minimal or no replanning. Second it allows us to generalize plans from smaller tractable environments to significantly larger ones that could not be solved directly with our planning algorithm. Our theoretical analysis proves that, by solving a linear program over a sampled set of "small" worlds, we obtain a solution that is close to the one we would have obtained if we had sampled all possible worlds. This LP can be solved by our factored LP decomposition technique, allowing us to obtain the weights of our class-level basis functions very efficiently. Finally, we present an approach for learning subclass structure by finding regularities in the value functions of a set of small worlds.

We present empirical evaluations of our generalization algorithm, demonstrating the two complementary uses of generalization: First, in a multiagent network management task, we showed that the quality of the generalized policies are very close to those obtained by replanning in each world, without class restrictions in the value function. We have also empirically demonstrated that our approximations can be significantly enhanced by learning subclasses of objects.

Second, we demonstrated in the actual Freecraft game that our class-based value functions allow us to generalize plans from smaller tractable environments to significantly larger ones that could not be solved directly with our planning algorithm. This real strategic computer game contains many characteristics present in real-world dynamic resource allocation problems. Our generalized policies for this the Freecraft domain demonstrated the long-term planning, and elaborate coordination between agents required to solve such general resource allocation problems.

13.6.1 Comparisons and limitations

Several other authors have considered the generalization problem, first in traditional planning [Fikes *et al.*, 1972], and later in stochastic domains [Sutton & Barto, 1998; Thrun & O'Sullivan, 1996]. Several approaches can represent value functions in general terms, but usually require it to be hand-constructed for the particular task. Others have focused on reusing solutions from isomorphic regions of state space [Parr, 1998; Hauskrecht *et al.*, 1998; Dietterich, 2000]. By comparison, our method exploits similarities between objects evolving in parallel. It would be very interesting to combine these two types of decomposition, as discussed in Section 14.2.8.

The work of Boutilier *et al.* [2001] on symbolic value iteration computes first-order value functions that generalize over objects in a world. However, it focuses on computing exact value functions, which are unlikely to generalize to a different world. Furthermore, it relies on the use of theorem proving tools, which adds to the complexity of the approach.

Methods in deterministic planning have focused on generalizing from compactly described policies learned from many domains to incrementally build a first-order policy [Khardon, 1999; Martin & Geffner, 2000]. Closest in spirit to our approach is the recent work of Džeroski *et al.* [2001] and of Yoon *et al.* [2002] that extends these deterministic approaches to stochastic domains. Their methods find regularities in exact policies or Q-functions obtained for small environments. These approaches then use *inductive logic programming* (ILP) methods [De Raedt (ed.), 1995] to obtain relational representations of the policy or Q-function. We thus view these methods as attempting to find generalized solutions from the compact policies or value functions obtained from goal-based algorithms, such as those described in Section 7.8.2, where a policy or value function can sometimes be represented compactly using a propositional description.

Our procedure for discovering subclasses by finding structure in the factored value function is, in some sense, analogous to the ILP approaches. Once we have learned these subclasses, we perform a global planning step, taking into account many problem domains simultaneously. More fundamentally, we are attempting to find an approximate factored value functions that generalize across environments, rather than similarities between parts of exact policies or value functions. The comparison between our generalization approach and those of Džeroski *et al.* [2001] and of Yoon *et al.* [2002] is analogous to the comparisons between our factored planning algorithms and the planning methods of Kushmerick *et al.* [1995] and of Blum and Langford [1999], presented in Section 7.8.2. In multiagent settings, however, exact compact propositional descriptions of the policy or of the value function are often very difficult to obtain. Thus, we expect that the algorithms of Džeroski *et al.* [2001] and of Yoon *et al.* [2002] will be less successful in these settings.

The key assumption in our method is interchangeability between objects of the same class. Our mechanism for learning subclasses allows us to deal with cases where objects in the domain can vary, but our generalizations will not be successful in very heterogeneous environments, where most objects have very different influences on the overall dynamics or rewards. Additionally, the efficiency of our LP solution algorithm depends on the connectivity of the underlying problem. In a domain with strong and constant interactions between many objects (*e.g.,* RoboCup), or when the reward function depends arbitrarily on the state of many objects (*e.g.,* Blocksworld), the solution algorithm will probably not be efficient, as discussed in Section 7.8.2.

Although our experiments show that we can successfully apply our class-based value functions to new environments without replanning, there are domains where such direct application would not be sufficient to obtain a good solution. In such domains, our generalized value functions can provide a good initial policy, which could be refined using a variety of local search methods.

We have assumed that relations only change deterministically over time. In many domains (*e.g.,* Blocksworld or RoboCup), this assumption is false. In Chapter 10, we showed that *context-specific independence* can allow for dynamically changing coordination structures in multiagent environments. Similar ideas may allow us to tackle stochastically changing relational structures, as discussed further in Section 14.2.9.

13.6.2 Summary

This part of the thesis presents: RMDPS, a relational representation for MDPs; a class-based value functions that, once optimized, can be instantiated to generate approximate solutions to new environments without replanning; an efficient LP-based algorithm for optimizing the weights of these class-based basis functions by considering a polynomial sample of "small" environments; and a method for discovering subclass structure that can improve the quality of our approximations. Our empirical results support both RMDPs as a model for generalization in dynamic environments, and our LP-based algorithm for learning the parameters of the class-level value function from a set of sampled environments. The choice of learning algorithm is, of course, orthogonal to our RMDP representation. However, we believe that our combined methods will provide a strong framework for obtaining generalized approximate solutions to large-scale stochastic planning domains.

Part V

Conclusions and future directions

Chapter 14

Conclusions

This thesis demonstrates that, by exploiting problem-specific structure, we can scale up automated methods for planning under uncertainty to complex large-scale domains. This chapter provides a summary of the methods and algorithms presented in this thesis, along with a discussion of some limitations of our work and future questions that remain open.

14.1 Summary and contributions

This sections presents a high-level summary of our results, emphasizing the connections between the parts of the thesis.

14.1.1 Foundation

This thesis provides a formal framework for exploiting problem specific structure in complex planning problems under uncertainty. Factored MDPs [Boutilier *et al.*, 1995] allow us to represent such structured problems very compactly. Unfortunately, even with this compact representation, optimal planning is still intractable [Mundhenk *et al.*, 2000; Liberatore, 2002; Allender *et al.*, 2002]. We thus focus on approximate solutions for such problems. Specifically, we choose a linear approximation architecture [Bellman *et al.*, 1963], where the value function is approximated as linear combination of a set of basis functions:

$$\mathcal{V}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}).$$

This architecture provides simple and often effective methods for obtaining approximate solutions for planning problems. More specifically, in the context of factored MDPs, we focus on factored value functions [Koller & Parr, 1999], where each basis function h_i is restricted to depend only on a small set of state variables $\mathbf{C}_i$:

$$\mathcal{V}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}[\mathbf{C}_i]).$$

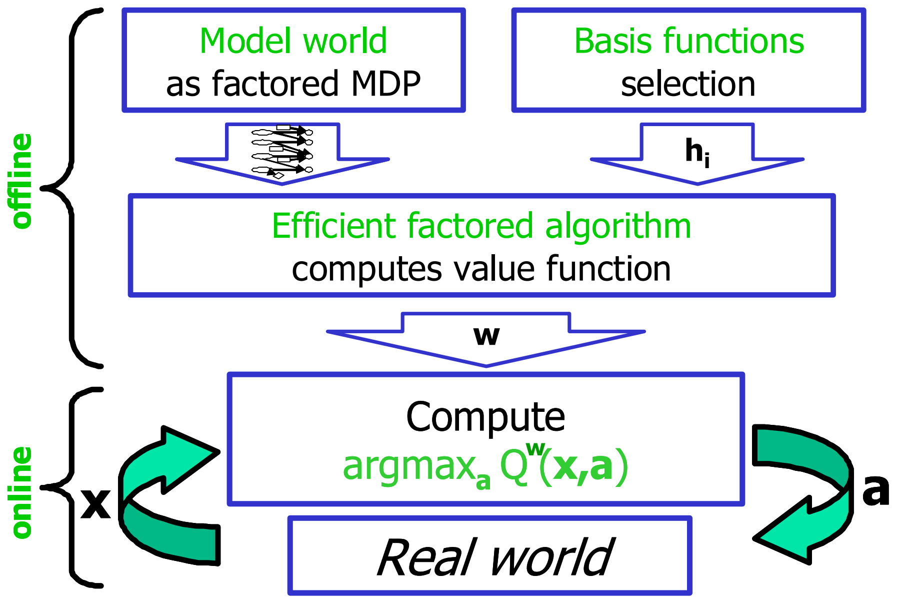
Factored value functions are sometimes able to represent near-optimal approximations of the true value function very compactly, even in some exponentially-large problems, as demonstrated in this thesis.

This thesis builds on this basic factored representation of the MDP and of the value function to design very efficient planning algorithms and multiagent coordination strategies. Additionally, we extend our approach to address multiagent reinforcement learning problems where a model of the world is not known, and to obtain multiagent coordination structures, which may vary with the state of the system. Finally, we describe a relational representation for MDPs, which allows us to generalize solutions devised from a sample of small worlds to larger worlds, without replanning.

Our algorithms leverage on factored LPs, a novel LP decomposition technique, analogous to variable elimination in cost networks [Bertele & Brioschi, 1972], that reduces an exponentially-large LP to a provably equivalent, polynomial-sized one. This algorithm, described in Chapter 4, is a central element in almost every method in this thesis.

14.1.2 Factored single agent planning

The basic factored MDP representation addresses single agent problems. For such problems, we have developed three planning algorithms, which build on our factored LP decomposition technique. These algorithms follow the structure described in Figure 14.1.



Model world as factored MDP
Basis functions selection
h_i
Efficient factored algorithm computes value function
w
offline
online
x
Compute argmax_a Q^w(x,a)
a
Real world

Figure 14.1: Overview of our framework for efficient planning in structured single agent problems.

We assume that the world is modelled by a factored MDP and that a set of (factored) basis functions has been selected. Offline, one of our planning algorithms is used to compute the weights $\mathbf{w}$ of our basis functions. Then, online, the agent follows a closed-loop policy, by observing the current state $\mathbf{x}$ from the real world, computing the greedy action:

$$\arg\max_a Q^{\mathbf{w}}(\mathbf{x}, a) = \arg\max_a R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i h_i(\mathbf{x}') \, .$$

This greedy action can be computed efficiently using the backprojection algorithm in Section 3.3. Once the maximizing action is obtained, the agent executes this action in the real world, observes the next state, and the process is repeated.

We proposed three algorithms for optimizing the weights $\mathbf{w}$ of the basis functions: approximate policy iteration, LP-based approximation, and the factored dual algorithm. Our approximate policy iteration algorithm is motivated by error analyses showing the importance of minimizing $\mathcal{L}_\infty$ error. This algorithm is more efficient and substantially easier to implement than previous methods based on the $\mathcal{L}_2$-projection. Our experimental results suggest that our $\mathcal{L}_\infty$ method also performs better in practice. Both of these algorithms require a default action assumption, stating that an action only modifies the CPDs of a few state variables.

Our factored LP-based approximation algorithm is simpler, easier to implement and more general than the policy iteration approach. Unlike our policy iteration algorithm, the LP-based approximation algorithm does not rely on the default action assumption stating that actions only affect a small number of state variables. Although the LP-based approximation algorithm does not have the same theoretical guarantees as max-norm projection approaches, empirically it seems to be a favorable option. Our experiments suggest that approximate policy iteration tends to generate better policies for the same set of basis functions. However, due to the computational advantages, we can add more basis functions to the LP-based approximation algorithm, obtaining a better policy and still maintaining a much faster running time than approximate policy iteration approach.

The complexity of our planning algorithms is only exponential in the induced width of a cost network formed by the backprojections of the basis functions, rather than exponential in the number of variables. Thus, these algorithms will be very efficient in sparsely connected factored MDPs that can be well-approximated by basis functions whose scope is restricted to small sets of variables.

We also present the factored dual algorithm. This novel formulation provides an approximate version of our factored LP decomposition technique. We can thus potentially address problems with large induced width. Additionally, this formulation provides an anytime version of our factored planning approach by incrementally improving the approximation of the LP decomposition. Although we currently cannot provide theoretical bounds on the quality of this approximation of our factored LP decomposition, we believe that this novel factored dual approach may provide effective solutions to many complex problems that could not be solved, even with our other factored planning algorithms.

Our experimental results on single agent problems demonstrate polynomial scaling for problems with fixed induced width, as expected by our complexity analysis. For some small problems, where the optimal solution can be computed exactly, we show that the actual long-term reward received by the policies obtained by our approximate methods using

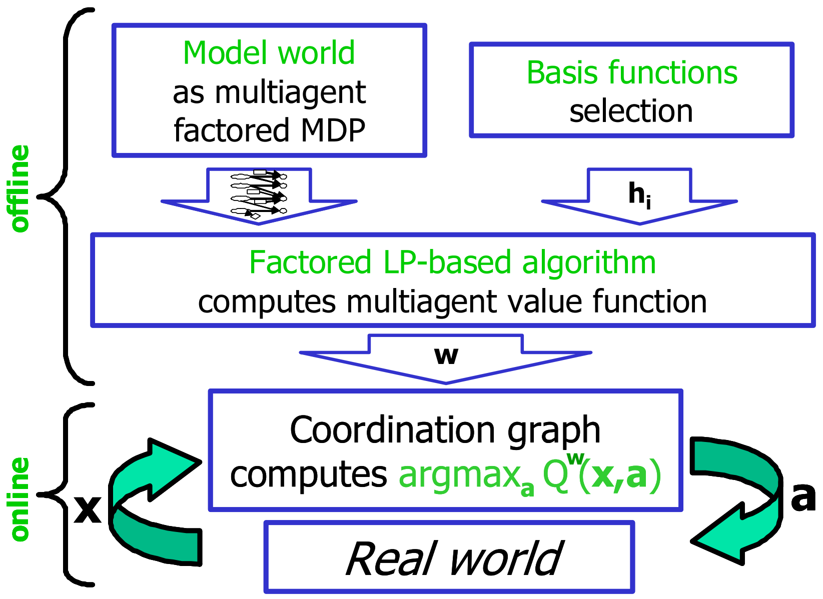

offline
Model world as multiagent factored MDP
Basis functions selection
h_i
Factored LP-based algorithm computes multiagent value function
w
online
x
Coordination graph computes $\mathrm{argmax}_a Q^w(x,a)$
a
Real world

Figure 14.2: Overview of our algorithm for efficient planning and coordination in structured multiagent problems.

simple basis functions are within 6% of those obtained by the optimal policy. For larger problems, we compute bounds on the quality of our policies showing that our solutions do not degrade significantly as the problem size increases.

14.1.3 Multiagent coordination and planning

In this thesis, we also address planning under uncertainty problems involving multiple collaborating agents. We approximate the value function for such problems using the same factored value function representation described above. Unfortunately, our approximate dynamic programming algorithms do not apply in these multiagent problems, as the policies can no longer be represented compactly by a decision list. Fortunately, the computation of the weights of the approximate value function can be performed by using a simple extension of the factored LP-based approximation algorithm. Similarly, our factored dual algorithm is also appropriate for solving such multiagent problems, thus allowing us to tackle some domains with large induced width.

As outlined in Figure 14.2, we approach multiagent problems by modelling the system as a multiagent factored MDP, and then selecting a set of factored basis functions. The weights of our factored value function are then computed offline by one of these two factored planning algorithms. As in the single agent case, the multiple agents then follow a closed-loop policy, where the maximizing joint action is computed online for each state $\mathbf{x}$ visited by the system according to:

$$\arg\max_{\mathbf{a}} Q^{\mathbf{w}}(\mathbf{x}, \mathbf{a}) = \arg\max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_i w_i h_i(\mathbf{x}') \ ,$$

where $\mathbf{a}$ is a joint action defining the specific action of each agent. Unfortunately, the number of joint actions is exponential in the number of agents. Furthermore, this maximization requires a centralized optimization procedure, which, as discussed in Chapter 1, is not desirable in many practical real world problems.

In Chapter 9, we address the action selection problem by proposing the *coordination graphs* framework, a novel, simple, distributed message passing algorithm based on variable elimination. This procedure leads to the selection of the optimal maximizing action in multiagent problems, while only requiring agents to observe a small set of state variables, and to communicate with a small number of other agents. This limited observability and communication properties should increase the applicability of our methods to many complex problems, such as the application to the RoboCup presented by Kok *et al.* [2003].

Interestingly, the communication structure between agents is not defined *a priori*, as in many existing methods, but is derived directly from the structure of the factored MDP and of the factored value function. The communication bandwidth required between agents is exactly the induced width of the coordination graph derived from our formulation. We thus present an unified view of multiagent coordination and value function approximation: A particular factored value function structure induces a particular coordination structure between agents. If the we choose to increase the scope of the basis functions, for example, we can probably increase the quality of our approximations. On the other hand, this new representation will probably form a coordination graph of higher induced width, thus requiring more communication between agents.

It is interesting to compare the complexity of our planning and coordination algorithms.

Our factored LP-based approximation algorithm is exponential in the induced width of a cost network formed by the backprojection of our basis functions. In contrast, our coordination graph is formed by the same backprojections, but where the state of the system has been instantiated. Thus, the induced width of the coordination graph depends only on the action variables of our system. Therefore, our coordination step can be exponentially faster than our planning algorithm. This difference allows us to coordinate agents even in very highly connected complex environments, where our planning algorithms are infeasible, such as the larger game scenarios described in Section 13.5 that can only be solved by generalizing solutions from smaller problems.

Our multiagent experimental results again demonstrate polynomial time scaling for problems with fixed induced width, and near-optimal policies in small problems that can be solved exactly. For larger problems, with simple basis functions, we obtain solutions that are within 5% of a loose upper bound on the quality of the optimal policy. We also compare our methods to state-of-the-art algorithms of Schneider *et al.* [1999], demonstrating that, at least for these problems, our solutions are about 10% better than those obtained by previous approaches. Finally, a (more general) alternative to our factored LP algorithm is to sample a subset of the exponential number of constraints present in our LPs, as analyzed by de Farias and Van Roy [2001b]. When compared to our closed-form factored LP algorithm, uniform sampling of constraints yielded policies whose quality degraded significantly as the problem size increased. As discussed by de Farias and Van Roy [2001b], sampling methods can be quite sensitive to the choice of sampling distribution. Thus, these sampling results could potentially be improved with a different sampling distribution, though, in general, it may be difficult to find such distribution. Even when provided with more expressive basis functions and significantly more running time, the policies obtained by the uniform sampling approach degraded on larger problems to at least 10% lower levels than the policies obtained by our approach with less expressive basis functions.

14.1.4 Context-specific independence and variable coordination structure

Unlike previous approaches, our algorithms can exploit both additive and context-specific structure in the factored MDP model, by using a rule-based representation instead of the standard table-based one. Many real-world systems possess both of these types of structure. Thus, this feature of our algorithms will increase the applicability of factored MDPs to more practical problems.

We demonstrated that exploiting context-specific independence, can yield exponential improvements in computational time when the problem has significant amounts of CSI. However, the overhead of managing sets of rules make it less well-suited for simpler problems. We also compared our approach to the work of Boutilier *et al.* [2000] that exploits only context-specific structure. For problems with significant context-specific structure in the value function, their approach can be faster due to their efficient handling of the ADD representation used by their algorithm. However, there are many problems with significant context-specific structure in the problem representation, rather than in the value function, that require exponentially-large ADDs. In some such problem classes, we demonstrated that by using a linear value function, our algorithm can obtain a polynomial-time near-optimal approximation of the true value function.

By exploiting context-specific structure in multiagent settings, we also provide a principled and efficient approach for planning in multiagent domains, where the required interactions between agents may vary from one situation to another. We show that the task of finding an optimal joint action in our approach leads to a very natural communication pattern, where agents send messages along a coordination graph with a dynamically changing structure that is determined by the value rules representing the value function. This coordination structure changes according to the state of the system, and even according to the actual numerical values assigned to the value rules. Furthermore, the coordination graph can be adapted incrementally as the agents learn new rules or discard unimportant ones.

We show empirically that our results scale to very complex problems, including high induced width problems, where traditional table-based representations of the value function

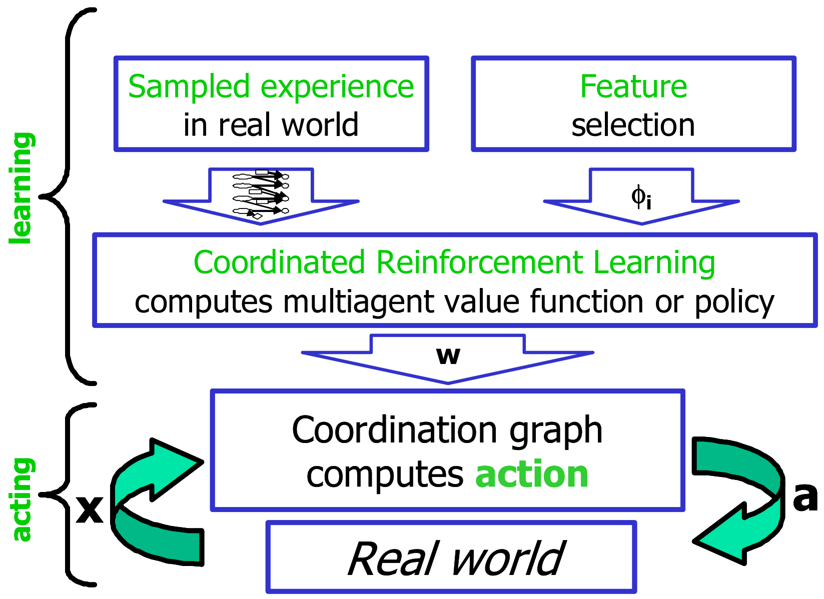


Figure 14.3: Overview of our coordinated reinforcement learning framework.

blow up exponentially. In problems where the optimal value could be computed analytically for comparison purposes, the value of the policies generated by our approach were within 0.05% of the optimal value. Our experiments also verify the variable coordination property of our approach, demonstrating that the coordination structure can vary significantly according to the state of the system.

14.1.5 Coordinated reinforcement learning

Thus far, we have assumed that the system we are tackling has been modelled by a factored MDP. In many practical problems, this model is not known *a priori*. In such cases, the agents must learn effective policies through their interactions with the environment by applying reinforcement learning strategies. In this thesis, we demonstrate that many of the existing RL algorithms that have been successfully applied to single agent problems can be generalized to collaborative multiagent settings by applying simple extensions of our factored value function representation, along with our multiagent coordination algorithm.

This overall framework, which we call coordinated reinforcement learning, is outlined in Figure 14.3. Rather than selecting basis functions over a subset of the state variables, we now define parametric local Q-functions $Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a})$ over state and action variables, which are then used to approximate the global Q-function:

$$Q^{\mathbf{w}}(\mathbf{x}, \mathbf{a}) = \sum_i Q_i^{\mathbf{w}_i}(\mathbf{x}, \mathbf{a}).$$

In contrast to the factored value function, the local $Q_i^{\mathbf{w}_i}$ functions may depend arbitrarily on any set of state variables, including state variables defined over continuous spaces. Furthermore, we no longer require that the underlying model be represented by a factored MDP. We do require that the scope of each $Q_i^{\mathbf{w}_i}$ be restricted to depend only on a small set of action variables; this assumption allows us to apply our coordination graph algorithm to select the maximizing action. Note that we could also utilize a rule-based representation of each $Q_i^{\mathbf{w}_i}$. In such cases, we would have a varying coordination structure both during the learning process and during action selection.

In this thesis, we applied the coordinated RL framework to generalize three existing single agent RL algorithms to multiagent problems: Q-learning [Watkins, 1989; Watkins & Dayan, 1992], LSPI [Lagoudakis & Parr, 2001], and policy search [Williams, 1992]. With Q-learning and policy search, the learning mechanism can be distributed. Agents communicate reinforcement signals, utility values, and conditional policies. In LSPI some centralized coordination is required to compute the projection of the value function. The resulting policies can always be executed in a distributed manner. We believe the coordination mechanism can be applied to almost any reinforcement learning method.

We present two types of experimental comparisons involving our multiagent version of LSPI: First, we compare the multiagent LSPI *learning* algorithm to our factored LP-based *planning* algorithm, which assumes full knowledge of the factored model. In these problems, the quality of the solutions obtained by the planning algorithm tended to be only slightly better policies than that of multiagent LSPI, when using comparable sets of basis functions. The amount of data required by the multiagent LSPI algorithm scaled linearly with the number of state and action variables even though the underlying space was growing exponentially. We also compare multiagent LSPI to the learning algorithms of Schneider *et al.* [1999]. Our multiagent LSPI algorithm obtained better policies both

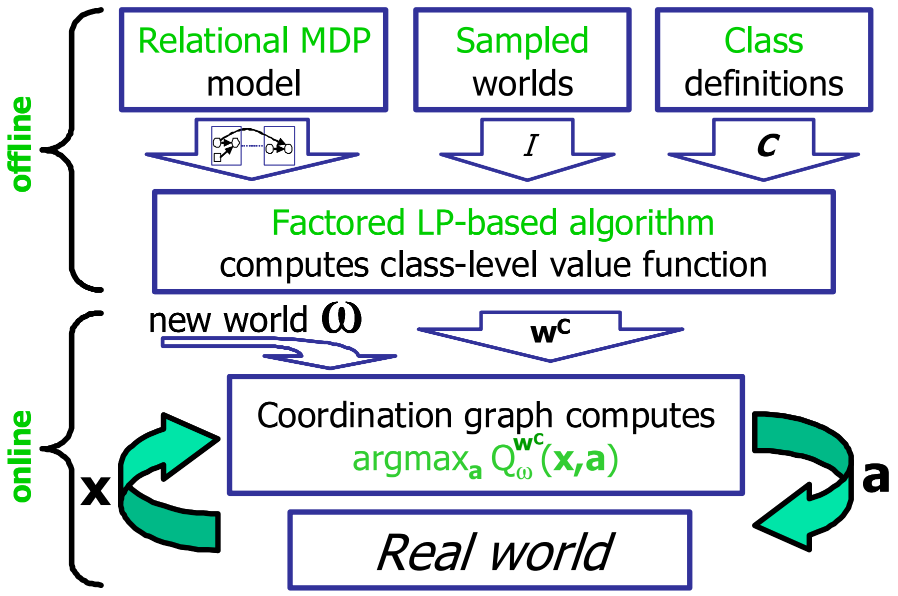

offline
Relational MDP model
Sampled worlds
Class definitions
I
C
Factored LP-based algorithm computes class-level value function
online
new world ω
w^C
Coordination graph computes argmax_a Q_ω^{w^C}(x,a)
x
a
Real world

Figure 14.4: Overview of our generalization approach using relational MDPs.

on the SysAdmin problem used in this thesis, and on the power grid problem described by Schneider *et al.* [1999]. Finally, our experiments demonstrate that coordination between agents can significantly improve the quality of the policies obtained by our approach.

14.1.6 Generalization to new environments

We have also tackled a longstanding goal in planning research, the ability to generalize plans to new environments. Such a generalization has two complementary uses: First, we can tackle new environments with minimal or no replanning. Second, it allows us to generalize plans from smaller tractable environments to significantly larger ones that could not be solved directly with our planning algorithm.

Our generalization approach builds on a novel relational representation of the MDP, where a domain is represented in terms of related objects of various classes. We achieve generalization by defining a value function at the level of object classes. Specifically, every object of class C shares the same set of basis function weights $\mathbf{w}^C$. Using this class-level representation, we can obtain a value function for any world ω in our domain by instantiating the class-level basis functions with the state of each specific object o in this world:

$$\mathcal{V}_\omega^{\mathbf{w}^C}(\mathbf{x}) = \sum_{C\in\mathcal{C}} \sum_{o\in\mathcal{O}[\omega][C]} \mathcal{V}_C^{\mathbf{w}^C}(\mathbf{x}[\mathbf{T}_o]) = \sum_{C\in\mathcal{C}} \sum_{o\in\mathcal{O}[\omega][C]} \sum_{h_i\in\mathsf{Basis}[C]} w_i^C h_i^C(\mathbf{x}[\mathbf{T}_o]).$$

By backprojecting this value function, we obtain a Q-function for this world:

$$Q_\omega^{\mathbf{w}^C}(\mathbf{x},\mathbf{a}) = R_\omega(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) \mathcal{V}_\omega^{\mathbf{w}^C}(\mathbf{x}').$$

In single agent problems, we can obtain the policy by simply selecting the action that maximizes $Q_\omega^{\mathbf{w}^C}$ at the current state. In multiagent problems, we can use our coordination graph algorithm to select this maximizing action.

We also present an optimization algorithm for determining the weights $\mathbf{w}^C$ of our class-level value function. We propose an optimality criteria, where the weights $\mathbf{w}^C$ are optimizing simultaneously for all worlds. For each particular world, we could optimize these weights using one of our efficient planning algorithms. However, optimizing for all worlds simultaneously is both infeasible, and does not fulfill our generalization goal, as all worlds must be considered. Instead we propose a formulation, where the parameters $\mathbf{w}^C$ are optimized over a set of sampled worlds. We prove that, by sampling a polynomial number of "small" worlds, we obtain an estimate of the class-level value function that is close to the one we would obtain had we planned for all worlds simultaneously.

We first assumed that set of classes of objects had been predefined in the model. We then describe a learning method for dividing objects into classes. This method first solves a few sampled environments, where each object belongs to an unique class. We can then use standard learning methods, such as decision tree regression, to find similarities between the value functions of different objects. We can then use the result of this learning algorithm to divide objects into a small set of classes.

Our overall generalization approach is outlined in Figure 14.4. We use a relational representation of the MDP, a sample of small worlds, and the class definitions, perhaps obtained by our learning method, to formulate a compact linear program to optimize the class-level parameters $\mathbf{w}^C$ offline. Then, online, when faced with a new world, we can instantiate our class-level value function, obtaining a Q-function for this new world, thus yielding a policy without any replanning.

Our relational MDP formulation of the generalization problem could be applied in conjunction with any algorithm for optimizing the class-level parameters $\mathbf{w}^C$. Although we believe that our class-level LP provides an effective method for performing such optimization, other algorithms could also be applied. For example, our coordinated RL approach could be used to optimize these parameters in settings where the factored MDP model is not known *a priori*. Other methods could also be applied when the value function representation depends non-linearly on the parameters $\mathbf{w}^C$. Finally, for simplicity, we have assumed that our class-level value function provides an effective policy for new worlds, without any replanning. We could, of course, use the class-level value function as a good starting point from which the specific policy for this new world is optimized.

Our experimental results support the fact that our class-based value function generalizes well to new plans, and that the class and subclass structure discovered by our learning procedure improves the quality of the approximation. Specifically, on a set of simulated problems, we show that the performance of the policy obtained from our generalized value function was within 1% of the policy obtained when replanning in each world, without any class restrictions. Furthermore, we successfully demonstrated our methods on Freecraft, a real strategic computer game that contains many characteristics present in real-world dynamic resource allocation problems. Here, we were able to generalize from a small environment to a very large, highly-connected environment that could not be solved directly by our factored algorithms.

14.2 Future directions and open problems

We now outline some directions that remain open, which we feel could lead to fruitful research topics, and provide some initial thoughts on how these directions could be pursued.

14.2.1 Basis function selection

The success of our algorithms depends on our ability to capture the most important structure in the value function using a linear, factored approximation. This ability, in turn, depends on the choice of the basis functions and on the properties of the domain. The algorithms currently require the designer to specify the factored basis functions. This is a limitation compared to the algorithms of Boutilier *et al.* [2000] that are fully automated. However, our experiments suggest that a few simple rules can be quite successful for designing a basis. First, we ensure that the reward function is representable by our basis. A simple basis that, in addition, contained a separate set of indicators for each variable often did quite well. We can also add indicators over pairs of each variable; most simply, we can choose these according to the DBN transition model, where an indicator is added between variables X_i and each one of the variables in $\mathsf{Parents}(X_i)$, thus representing one-step influences. This procedure can be extended, adding more basis functions to represent more influences as required. Thus, the structure of the DBN gives us indications of how to choose the basis functions. Other sources of prior knowledge can also be included for further specifying the basis.

Nonetheless, a general algorithm for choosing good factored basis functions still does not exist. However, there are some potential approaches: First, in problems with CSI, one could apply the algorithms of Boutilier *et al.* for a few iterations to generate partial tree-structured solutions. Indicators defined over the variables in backprojection of the leaves could, in turn, be used to generate a basis set for such problems. Second, the Bellman error computation, which we perform efficiently as shown in Section 5.3, not only provide a bound on the quality of the policy, but also the actual state where the error is largest. This knowledge can be used to create a mechanism to incrementally build the basis set, adding new basis functions to tackle states with high Bellman error. Finally, the recent work of Poupart *et al.* [2002] and Patrascu *et al.* [2002], building on our factored algorithms, attempts to greedily construct a set of basis functions in order to improve the quality of the approximation. Such approaches provide some basic groundwork for the design of automated basis function selection mechanisms. Such mechanisms could significantly extend the applicability of our methods.

14.2.2 Structured error analysis

If a factored MDP can be divided into completely independent parts, then it is clear that a factored value function spanning each part separately will be able to represent the optimal value function. Intuitively, if a system is formed by weakly interacting parts, then we expect that it may be possible to approximate its true value function with a factored representation. It would thus be interesting to prove a theoretical bound on the quality of the solutions obtained by our factored algorithms that depends explicitly on the structure of the underlying factored MDP. A potential avenue for proving such bounds could use the error analysis of de Farias and Van Roy [2001a], which allows us to introduce a Lyapunov function, weighing the approximation differently in different parts of the state space. We could thus select a Lyapunov function that is compatible with the structure of the factored MDP, weighing weakly interacting parts appropriately. The structure analysis developed by Boyen and Koller [1999] for approximate inference in DBNs could provide intuitions on how to obtain such a Lyapunov function. Such a bound could be useful both in understanding when a factored MDP can be effectively approximated, and how to select appropriate basis functions to obtain good approximations.

Another interesting, related, theoretical direction is to analyze when our class-level value functions will provide a good (or even exact) solution to new environments. We can again view this problem as one of bounding the quality of the solutions obtained by the LP-based approximation algorithm, though we are now solving the meta MDP $\Pi_{\texttt{meta}}$. However, there are other situations where different types of generalization bounds could be obtained. For example, in some cases that contain significant amounts of symmetry, it may be possible to prove that a class-based value function is equal to the object-based value function we would obtain had we not imposed class restrictions. In the tactical Freecraft problem in Example 12.2.2, if we only had class-based basis functions between an enemy and its related footman, then, by symmetry, this class-based solution will be equal to an object-based solution that allows each enemy to use different parameter values. This type of analysis could also provide automated methods for designing the class structure of the relational MDP.

14.2.3 Models with large induced width

A central element governing the efficiency of our planning algorithms is the induced width of the underlying cost network formed by the backprojection of our basis functions. A very important open direction is the design of algorithms that can tackle problems with large induced width. We have proposed two approaches in this thesis: In problems with large induced width, but with significant amounts of context-specific structure in the model, we can apply our rule-based factored LP decomposition to obtain approximate solutions efficiently. Alternatively, in problems where the backprojection of each basis function depends on a small set of variables, but where the induced width of the underlying cost network is still very large, we can apply our approximately factored dual algorithm described in Chapter 6.

However, there are practical problems that do not fall into one of these two cases. For example, when a variable X_i in the DBN has a noisy-or CPD depending on many other variables in the previous time step, neither our rule-based algorithms, nor our approximate factorization, will be able to tackle this problem. An example of such a variable is the *Gold* variable in our Freecraft model, whose CPD depends on the state of every peasant in the model. We could address the Freecraft problem by generalizing solutions from smaller game scenarios. Nonetheless, it is still important to design algorithms for handling high induced width models in problems where generalization is not effective.

There are many possible solution paths for tackling problems with large induced width. Many of these directions build on successful algorithms for approximate or exact inference in graphical models with large induced width. For example, as discussed in Section 6.3, we can relate our approximately factored dual algorithm to the belief propagation algorithm [Pearl, 1988; Yedidia *et al.*, 2001]. Similarly, combining sampling or conditioning techniques, where a subset of the variables are instantiated, with exact inference has lead to successful algorithms for inference in graphical models [Horvitz *et al.*, 1989; Casella & Robert, 1996; Doucet *et al.*, 2000; Bidyuk & Dechter, 2003; Allen & Darwiche, 2003]. We could follow a similar path by combining the sampling approach of de Farias and Van Roy [2001b] with our LP decomposition technique. Interestingly, we can view our class-level LP formulation as a special case of such a procedure, where a set of worlds is sampled, and our LP decomposition technique is applied exactly for each of these worlds. Finally, Pfeffer [2000] and Poole [2003] have recently developed methods that can handle sets of similar objects simultaneously, without generating a propositional representation of the world. These methods could perhaps also be incorporated into our factored LP decomposition technique, thus handling worlds with a very large number of objects.

14.2.4 Complex state and action variables

We have assumed that each state, or action variable takes on one of a few discrete values. Although our coordinated RL approach can also handle continuous state variables, we feel that this is an important issue, which requires further investigation. We can attempt to handle continuous state variables by using discretization algorithms [Chow & Tsitsiklis, 1991; Rust, 1997], reducing the continuous problem to a discrete one. More interestingly, we could attempt to design discretization methods that are compatible with our factored LP decomposition technique, discretizing the intermediate factors generated by our maximization algorithm, rather than the state variables themselves. Such approach could introduce an explicit link between discretization complexity and the structure of a factored MDP. Similar types of discretization have been applied in DBNs by Kozlov and Koller [1997].

Models containing continuous action variables transform the action selection step into a general nonlinear optimization problem [Isidori, 1989]. Such problems are often very difficult to solve. An obvious approach to address this issue is to apply standard local search or gradient ascent algorithms to perform the optimization. Unfortunately, these approaches are usually prone to local maxima problems. A more interesting direction could be to attempt to exploit structure in our coordination graph. Specifically, we can use the same type of discretization of intermediate factors described above. This would allow us to perform the action selection step using a discretized dynamic programming algorithm, thus minimizing the influence of local optima.

Additionally, some problems may include discrete state or action variables that have very large domain sizes, for example, the action variable of a footman in Freecraft that selects among many enemies, or variables obtained when a continuous variable is discretized, such as the amount of gold in a Freecraft scenario. In such cases, even problems with relatively small induced width may be difficult to solve. We could address this problem by exploiting a structured decomposition of domain values using a similar representation to the rule-based one used for CSI [Geiger & Heckerman, 1996; Friedman & Singer, 2000; Sharma & Poole, 2003]. Alternatively, we could use sampling methods, such as the ones analyzed by de Farias and Van Roy [2001b], to consider only a subset of the assignments in the domain of such variables.

In our implementation, we have used indicator basis functions over assignments of the domain of small sets of variables. Such a representation will be infeasible in problems with large domain sizes. In such problems, we may need to use basis functions that generalize over possible domain values. For example, a basis function over a discretized variable may have values that depend polynomially on the assignment of the original continuous variable.

14.2.5 Model-based reinforcement learning

Coordinated RL is a model-free approach, that is, it attempts to obtain successful policies without explicitly building a model of the environment. Model-free algorithms do not need to make strong assumptions about the underlying structure of the world. Unfortunately, as no model of the world is maintained, it is often difficult to bound the quality of the current solution, or design effective exploration strategies. Model-based approaches, on the other hand, build a parametric model of the world and use this model to explore the environment effectively [Moore & Atkeson, 1993; Kearns & Singh, 1998; Brafman & Tennenholtz, 2001]. Furthermore, if the model parameterization is a good approximation of the underlying world, then model-based methods can be very effective. An interesting future direction is to design algorithms that effectively explore the environment, assuming that the underlying system can be modelled by a factored MDP. Kearns and Koller [1999] and Guestrin *et al.* [2002c] propose algorithms for exploring the environment in order to learn effective policies, assuming that the structure of the underlying factored MDP is known, but that the model parameters are unknown. Although these algorithms provide initial methods to address the factored model-based RL problem, a general solution that effectively learns both the structure and the parameters of a factored model is still an open problem.

14.2.6 Partial observability

We have assumed that the underlying planning problem is fully observable, that is, each agent can observe the state variables relevant to their local Q-function. In more general formulations, the agents may be only able to make noisy observations about the world, for example, using sensors. Such problems can be formulated as a partially observable Markov decision process (POMDP) [Sondik, 1971]. Exact solutions for POMDPs are intractable, even when the number of states is polynomial [Madani *et al.*, 1999; Bernstein *et al.*, 2000]. Typically, exact algorithms can only solve problems with tens of states [Cassandra *et al.*, 1997; Hansen, 1998]. Recent approximate methods have scaled to POMDPs with many hundreds of states Pineau *et al.* [2003].

Designing efficient POMDP solution algorithms that exploit problem structure is an exciting area of future research. One possible direction to tackle this problem is to exploit a factored representation of the POMDP [Boutilier & Poole, 1996], perhaps by using factored value function approximation methods [Guestrin *et al.*, 2001c]. Another option relies on projecting the space of possible beliefs over the state of the system into a lower dimensional space [Roy & Thrun, 2000; Poupart & Boutilier, 2002; Roy & Gordon, 2002]. We believe that an effective method for solving structured POMDPs could combine these two approaches by using a structured representation of the beliefs that is compatible with the structure of the factored POMDP, in a similar manner that our factored value function is compatible with the structure of the factored MDP. This decomposition would be analogous to the one we used to decompose the dual variables in our factored dual algorithm in Chapter 6. We believe that such approach could provide an effective method for solving large-scale POMDPs.

14.2.7 Competitive multiagent settings

This thesis has focused on long-term planning problems involving multiple collaborating agents that have the same reward function. However, many practical problems involve competitive settings, where the agents have different reward functions. Such stochastic dynamic systems involving multiple competing agents can be modelled using *stochastic games*, a generalization of MDPs, which was first proposed by Shapley [1953], and later studied by, among others, Littman [1994] and Brafman and Tennenholtz [2001]. As with standard MDPs, stochastic games suffer from the curse of dimensionality, as the number of possible strategies grows exponentially in the number of agents.

Many existing algorithms tackle stochastic games by using model-free reinforcement learning algorithms in two-player zero-sum settings. Specifically, Littman [1994] focused on exact solutions, while Van Roy [1998] and Lagoudakis and Parr [2002] present approximate solutions for such problems, by using linear approximations of the value function.

In recent years, there has been increasing interest in designing algorithms that exploit structure in *graphical games*, structured representations of competitive multiagent settings that do not evolve over time [Littman *et al.*, 2002; Leyton-Brown & Tennenholtz, 2003; Blum *et al.*, 2003]. This formulation can also be generalized to finite horizon problems represented by competitive extensions of influence diagrams [La Mura, 1999; Koller & Milch, 2001].

We believe that, by using factored value functions, we could exploit structure in factored models to solve two-player zero-sum problems efficiently, using extensions of the techniques developed in this thesis. Furthermore, by combining factored MDPs with graphical games, one could attempt to address infinite horizon problems involving multiple agents.

We can view our collaborative multiagent planning algorithm as an approximate method for obtaining best-response policies when the opponent is "nature". Stochastic games provide equilibrium strategies, where each agent plays a best-response policy, assuming the other agents are perfectly rational. In many settings, such as exponentially-large factored problems, agents can only perform approximate optimizations, and may thus not be perfectly optimal. We believe that often, in such settings, rather than defining the problem as one of attempting to respond optimally to rational agents, one should attempt to respond effectively to opponents that can be classified as belonging to certain classes of opponents. In such settings, one could use our methods, or extension to POMDPs, to obtain good strategies that attempt to respond well to opposing agents sampled from a distribution over the classes of possible opponents.

14.2.8 Hierarchical decompositions

Many researchers have examined the idea of dividing a planning problem into simpler subproblems in order to speed-up the solution process. There are two common ways to split a problem into simpler pieces, which we will call *serial decomposition* and *parallel decomposition*.

In a *serial decomposition*, exactly one subproblem is active at any given time. The overall state consists of an indicator of which subproblem is active along with that subproblem's state. Subproblems interact at their borders, that is, at states where we can enter or leave a subproblem. For example, imagine a robot navigating in a building with multiple rooms connected by doorways: fixing the value of the doorway states decouples the rooms from each other and lets us solve each room separately. In this type of decomposition, the combined state space is the union of the subproblem state spaces, and so the total size of all of the subproblems is approximately equal to the size of the combined problem.

Serial decomposition planners in the literature include the algorithms of Kushner and Chen [1974] and Dean and Lin [1995], as well as a variety of hierarchical planning algorithms. Kushner and Chen were the first to apply Dantzig-Wolfe decomposition to MDPs, while Dean and Lin combined this decomposition with state abstraction. Hierarchical planning algorithms include MAXQ [Dietterich, 2000], hierarchies of abstract machines [Parr & Russell, 1998], and planning with macro-operators [Sutton *et al.*, 1999; Hauskrecht *et al.*, 1998].

By contrast, in a *parallel decomposition,* multiple subproblems can be active at the same time, and the combined state space is the cross product of the subproblem state spaces. The size of the combined problem is therefore exponential rather than linear in the number of subproblems. Thus, a parallel decomposition can potentially save significantly more computation than a serial one. For an example of a parallel decomposition, suppose there are multiple robots in our building, interacting only through a common resource constraint such as limited fuel or through a common goal such as lifting a box which is too heavy for one robot to lift alone. A subproblem of this task might be to plan a path for one robot using only a compact summary of the plans for the other robots.

Parallel decomposition planners in the literature include the algorithms of Singh and

Cohn [1998], Meuleau *et al.* [1998] and Yost [1998]. Singh and Cohn's planner builds the combined state space explicitly, using subproblem solutions to initialize the global search. So, while it may require fewer planning iterations than naive global planning, it is limited by having to enumerate an exponentially-large set. Meuleau *et al.*'s planner, which was further improved by Yost [1998], is designed for parallel decompositions in which the only coupling is through global resource constraints. More complicated interactions such as conjunctive goals or shared state variables are beyond its scope.

Recently, Guestrin and Gordon [2002] propose a planning algorithm that handles both serial and parallel decompositions, providing more opportunities for abstraction than other parallel-decomposition planners. The approach of Guestrin and Gordon builds a hierarchical representation of a factored MDP that is analogous to the hierarchical decomposition of Koller and Pfeffer [1997] for Bayesian networks. In addition, Guestrin and Gordon [2002] propose a fully distributed planning algorithm: at no time is there a global combination step requiring knowledge of all subproblems simultaneously, contrasting with the factored planning algorithms presented in this thesis, which require the offline solution of a global linear program. This approach also allows for the reuse of solutions obtained in one subsystem in other similar subsystems. We can view this property as generalization within a planning problem, while our relational models provide generalizations between planning problems.

Unfortunately, the approach of Guestrin and Gordon [2002] requires a tree decomposition of the environment into subsystems. This tree structure is analogous to the triangulated clusters required in our factored dual algorithm. Thus, this decomposition will be infeasible in problems with large induced width. We believe that the approximate factorization described in Chapter 6, or one of the methods for tackling problems with large induced width described above, could be used to obtain approximate versions of the decomposition of Guestrin and Gordon [2002].

Such approximate decompositions could then be combined with other existing decomposition methods. For example, the algorithms of Meuleau *et al.* [1998] and Yost [1998] allow us to introduce more global resource constraints than our local decomposition technique. These methods could potentially be combined with the decompositions of Guestrin and Gordon [2002] to approximately represent systems involving both global constraints and local structure.

It would also be interesting to explore the combination of our parallel decomposition with the serial decomposition algorithms of Dietterich [2000], Parr and Russell [1998], Sutton *et al.* [1999], Hauskrecht *et al.* [1998], and Andre and Russell [2002]. The algorithm of Andre and Russell [2002], for example, would potentially allow us to introduce temporal abstractions into our factored model. When combined with our relational representation, we could obtain a hierarchical decomposition that allows us to generalize temporally-extended value functions. These two types of generalization could yield effective approximation methods for handling complex systems, using hierarchical, serial and parallel decompositions.

14.2.9 Dynamic uncertain relational structures

Our relational MDP assumed that, in a particular world, relations are either fixed, or change deterministically with the actions of different agents. In general domains, relations may change stochastically over time, though, as we are tackling fully observable problems, the values of the relations will be observed by the agents at every time step. Extending the relational MDP model to allow for changing relational structures is straightforward. The PRM framework of Koller and Pfeffer [1998] allows for relational uncertainty, the same framework could be applied to relational MDPs.

Note, however, that if the relational structure changes, then our definition of the objects in the scope of an instantiated class basis function may also change. In our SysAdmin problem, we had basis functions between pairs of neighboring objects in the network. If the structure of the network changes, the neighbor of a particular machine may change, and its contribution to the global value function will now depend on the state of a different machine. In such cases, we may need more elaborate methods for computing the backprojection of our basis functions. Specifically, the state in the current time step specifies a distribution over assignments to the relations in the next time step. For each one of these relational assignments the scope of our class basis function is well-defined. Thus, the backprojection of a class basis function will be a weighted linear combination of the backprojections obtained for each possible assignment to the relations in the next time step.

More importantly, we must adapt our planning algorithm to tackle such varying relational structures. Such problems will often have very high induced width. For example, consider a model of multiple robots exploring a building after an earthquake. The state of one robot could potentially be influenced every other robot. However, at every time step, a robot's state only depends on robots that are within a certain radius. Clearly, the induced width of such a problem will be very large, involving the state of all robots. However, there is a significant amount of context-specific structure in this problem. Generally, we could address relations that change over time by exploiting context-specific independence. However, CSI may not be sufficient to tackle such problems. In these cases, the other approaches for tackling problems with large induced width suggested above, such as sampling, conditioning, or approximate factorizations, could be used to address problems with dynamically changing relational structure.

14.3 Closing remarks

We believe that the framework described in this thesis significantly extends the efficiency, applicability, and general usability of automated methods in the control of large-scale dynamic systems. However, many issues remain to be studied before automated methods can be deployed in practical settings. In this chapter, we outline a few open directions that particularly relate to our approach. There are, of course, many other more general open questions that must be addressed before effective general-purpose methods can be designed for tackling large-scale complex systems. Ultimately, we hope that such automated methods will aid users in the solution of many real-world long-term planning tasks.

Appendix A

Main proofs

A.1 Proofs for results in Chapter 2

A.1.1 Proof of Lemma 2.3.4

There exists at least a setting to the weights — the all zero setting — that yields a bounded max-norm projection error β_P for any policy ($\beta_P \leq R_{max}$). Our max-norm projection operator chooses the set of weights that minimizes the projection error $\beta^{(t)}$ for each policy $\pi^{(t)}$. Thus, the projection error $\beta^{(t)}$ must be at least as low as the one given by the zero weights β_P (which is bounded). Thus, the error remains bounded for all iterations. ∎

A.1.2 Proof of Theorem 2.3.6

First, we need to bound our approximation of $\mathcal{V}_{\pi^{(t)}}$:

$$\begin{aligned}
\left\|\mathcal{V}_{\pi^{(t)}} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty &\leq \left\|\mathcal{T}_{\pi^{(t)}}\mathbf{H}\mathbf{w}^{(t)} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty + \left\|\mathcal{V}_{\pi^{(t)}} - \mathcal{T}_{\pi^{(t)}}\mathbf{H}\mathbf{w}^{(t)}\right\|_\infty \quad \text{; (triangle inequality;)}\\
&\leq \left\|\mathcal{T}_{\pi^{(t)}}\mathbf{H}\mathbf{w}^{(t)} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty + \gamma\left\|\mathcal{V}_{\pi^{(t)}} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty \quad \text{; (}\mathcal{T}_{\pi^{(t)}}\text{ is a contraction.)}
\end{aligned}$$

Moving the second term to the right hand side and dividing through by $1-\gamma$, we obtain:

$$\left\|\mathcal{V}_{\pi^{(t)}} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty \leq \frac{1}{1-\gamma}\left\|\mathcal{T}_{\pi^{(t)}}\mathbf{H}\mathbf{w}^{(t)} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty = \frac{\beta^{(t)}}{1-\gamma}. \tag{A.1}$$

For the next part of the proof, we adapt a lemma of Bertsekas and Tsitsiklis, [1996, Lemma 6.2, p.277] to fit into our framework. After some manipulation, this lemma can be reformulated as:

$$\left\|\mathcal{V}^* - \mathcal{V}_{\pi^{(t+1)}}\right\|_\infty \leq \gamma \left\|\mathcal{V}^* - \mathcal{V}_{\pi^{(t)}}\right\|_\infty + \frac{2\gamma}{1-\gamma} \left\|\mathcal{V}_{\pi^{(t)}} - \mathbf{H}\mathbf{w}^{(t)}\right\|_\infty . \tag{A.2}$$

The proof is concluded by substituting Equation (A.1) into Equation (A.2) and, finally, induction on t. ∎

A.2 Proof of Theorem 4.3.2

First, note that the equality constraints represent a simple change of variable. Thus, we can rewrite Equation (4.2) in terms of these new LP variables $u^{f_i}_{\mathbf{z}_i}$ as:

$$\phi \geq \max_{\mathbf{x}} \sum_i u^{f_i}_{\mathbf{z}_i}, \tag{A.3}$$

where any assignment to the weights $\mathbf{w}$ implies an assignment for each $u^{f_i}_{\mathbf{z}_i}$. After this stage, we only have LP variables.

It remains to show that the factored LP construction is equivalent to the constraint in Equation (A.3). For a system with n variables $\{X_1, \ldots, X_n\}$, we assume, without loss of generality, that variables are eliminated starting from X_n down to X_1. We now prove the equivalence by induction on the number of variables.

The base case is $n = 0$, so that the functions $c_i(\mathbf{x})$ and $b(\mathbf{x})$ in Equation (4.2) all have empty scope. In this case, Equation (A.3) can be written as:

$$\phi \geq \sum_i u^{e_i}. \tag{A.4}$$

In this case, no transformation is done on the constraint, and equivalence is immediate.

Now, we assume the result holds for systems with $i-1$ variables and prove the equivalence for a system with i variables. In such a system, the maximization can be decomposed into two terms: one with the factors that *do not* depend on X_i, which are irrelevant to the maximization over X_i, and another term with all the factors that depend on X_i. Using this decomposition, we can write Equation (A.3) as:

$$\begin{aligned}\phi &\geq \max_{x_1,\ldots,x_i} \sum_j u^{e_j}_{\mathbf{z}_j};\\ &\geq \max_{x_1,\ldots,x_{i-1}} \left[\sum_{l\,:\,X_i \notin \mathbf{Z}_l} u^{e_l}_{\mathbf{z}_l} + \max_{x_i} \sum_{j\,:\,X_i \in \mathbf{Z}_j} u^{e_j}_{\mathbf{z}_j} \right]. \qquad \text{(A.5)}\end{aligned}$$

At this point we can define new LP variables $u^e_{\mathbf{z}}$ corresponding to the second term on the right hand side of the constraint. These new LP variables must satisfy the following constraint:

$$u^e_{\mathbf{z}} \geq \max_{x_i} \sum_{j=1}^{\ell} u^{e_j}_{(\mathbf{z},x_i)[\mathbf{Z}_j]}. \qquad \text{(A.6)}$$

This new non-linear constraint is again represented in the factored LP construction by a set of equivalent linear constraints:

$$u^e_{\mathbf{z}} \geq \sum_{j=1}^{\ell} u^{e_j}_{(\mathbf{z},x_i)[\mathbf{Z}_j]}, \forall \mathbf{z}, x_i. \qquad \text{(A.7)}$$

The equivalence between the non-linear constraint Equation (A.6) and the set of linear constraints in Equation (A.7) can be shown by considering binding constraints. For each new LP variable created $u^e_{\mathbf{z}}$, there are $|X_i|$ new constraints created, one for each value x_i of X_i. For any assignment to the LP variables in the righthand side of the constraint in Equation (A.7), only one of these $|X_i|$ constraints is relevant. That is, one where $\sum_{j=1}^{\ell} u^{e_j}_{(\mathbf{z},x_i)[\mathbf{Z}_j]}$ is maximal, which corresponds to the maximum over X_i. Again, if for each value of **z** more than one assignment to X_i achieves the maximum, then any of (and only) the constraints corresponding to those maximizing assignments could be binding. Thus, Equation (A.6) and Equation (A.7) are equivalent.

Substituting the new LP variables $u^e_{\mathbf{z}}$ into Equation (A.5), we get:

$$\phi \geq \max_{x_1,\ldots,x_{i-1}} \sum_{l\,:\,x_i \notin \mathbf{z}_l} u^{e_l}_{\mathbf{z}_l} + u^e_{\mathbf{z}},$$

which does not depend on X_i anymore. Thus, it is equivalent to a system with $i - 1$ variables, concluding the induction step and the proof. ∎

A.3 Proof of Lemma 5.3.1

First note that at iteration $t+1$ the objective function $\phi^{(t+1)}$ of the max-norm projection LP is given by:

$$\phi^{(t+1)} = \left\| \mathbf{H}\mathbf{w}^{(t+1)} - \left(R_{\pi^{(t+1)}} + \gamma P_{\pi^{(t+1)}} \mathbf{H}\mathbf{w}^{(t+1)} \right) \right\|_\infty .$$

However, by convergence the value function estimates are equal for both iterations:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}.$$

So we have that:

$$\phi^{(t+1)} = \left\| \mathbf{H}\mathbf{w}^{(t)} - \left(R_{\pi^{(t+1)}} + \gamma P_{\pi^{(t+1)}} \mathbf{H}\mathbf{w}^{(t)} \right) \right\|_\infty .$$

In operator notation, this term is equivalent to:

$$\phi^{(t+1)} = \left\| \mathbf{H}\mathbf{w}^{(t)} - \mathcal{T}_{\pi^{(t+1)}} \mathbf{H}\mathbf{w}^{(t)} \right\|_\infty .$$

Note that, $\pi^{(t+1)} = \text{Greedy}[\mathbf{H}\mathbf{w}^{(t)}]$ by definition. Thus, we have that:

$$\mathcal{T}_{\pi^{(t+1)}} \mathbf{H}\mathbf{w}^{(t)} = \mathcal{T}^* \mathbf{H}\mathbf{w}^{(t)}.$$

Finally, substituting into the previous expression, we obtain the result:

$$\phi^{(t+1)} = \left\| \mathbf{H}\mathbf{w}^{(t)} - \mathcal{T}^* \mathbf{H}\mathbf{w}^{(t)} \right\|_\infty . \quad ∎$$

A.4 Proofs for results in Chapter 6

A.4.1 Proof of Lemma 6.1.1

The non-negativity condition is stated directly in the dual LP in (6.2).

To prove the condition in Equation (6.5), consider the constraint induced by the constant basis function h_0:

$$\sum_{\mathbf{x},a} \phi_a(\mathbf{x}) h_0(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_0(\mathbf{x}) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_0(\mathbf{x}) \ ;$$

yielding:

$$\sum_{\mathbf{x},a} \phi_a(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') \ .$$

Using the facts that $\sum_{\mathbf{x}} \alpha(\mathbf{x}) = 1$, and $\sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') = 1$, we obtain the result. ∎

A.4.2 Proof of Theorem 6.1.2

Item 1: Clearly $\phi_a^\rho(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ and a. We must now show that for an arbitrary basis function h_i:

$$\sum_{\mathbf{x},a} \phi_a^\rho(\mathbf{x}) h_i(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_i(\mathbf{x}) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}^\rho(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_i(\mathbf{x}) \ .$$

Substituting the definition of ϕ_a^ρ in Equation (6.6) into the second term on the righthand side of this constraint:

$$\begin{aligned}
&\gamma \sum_{\mathbf{x}',a'} \phi_{a'}^\rho(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_i(\mathbf{x}) \\
&= \gamma \sum_{\mathbf{x}',a'} \sum_{t=0}^{\infty} \sum_{\mathbf{x}''} \gamma^t \rho(a' \mid \mathbf{x}') P_\rho(\mathbf{x}^{(t)} = \mathbf{x}' \mid \mathbf{x}^{(0)} = \mathbf{x}'') \alpha(\mathbf{x}'') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') h_i(\mathbf{x}) \ ; \\
&= \sum_{\mathbf{x}''} \sum_{\mathbf{x}} \sum_{t=0}^{\infty} \sum_{\mathbf{x}',a'} \alpha(\mathbf{x}'') h_i(\mathbf{x}) \, \gamma^{t+1} P_\rho(\mathbf{x}^{(t)} = \mathbf{x}' \mid \mathbf{x}^{(0)} = \mathbf{x}'') \rho(a' \mid \mathbf{x}') P(\mathbf{x} \mid \mathbf{x}', a') \ .
\end{aligned}$$

As the transition probabilities of our randomized policy are defined by $P_\rho(\mathbf{x} \mid \mathbf{x}') = \sum_{a'} \rho(a' \mid \mathbf{x}')P(\mathbf{x} \mid \mathbf{x}', a')$, we obtain:

$$\begin{aligned}
&\gamma \sum_{\mathbf{x}',a'} \phi^\rho_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a')h_i(\mathbf{x}) \\
&= \sum_{\mathbf{x}''} \alpha(\mathbf{x}'') \sum_{\mathbf{x}} h_i(\mathbf{x}) \sum_{t=0}^{\infty} \gamma^{t+1} P_\rho(\mathbf{x}^{(t+1)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}'') \;; \\
&= \sum_{\mathbf{x}''} \alpha(\mathbf{x}'') \sum_{\mathbf{x}} h_i(\mathbf{x}) \left[\left(\sum_{t=0}^{\infty} \gamma^{t} P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}'') \right) - P_\rho(\mathbf{x}^{(0)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}'') \right] \;; \\
&= \sum_{\mathbf{x}''} \alpha(\mathbf{x}'') \sum_{\mathbf{x}} h_i(\mathbf{x}) \left[\left(\sum_{t=0}^{\infty} \gamma^{t} P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}'') \right) - \mathbb{1}(\mathbf{x}'' = \mathbf{x}) \right] \;; \\
&= \sum_{\mathbf{x},a} \phi^\rho_a(\mathbf{x})h_i(\mathbf{x}) - \sum_{\mathbf{x}} \alpha(\mathbf{x})h_i(\mathbf{x}) \;;
\end{aligned}$$

concluding the proof of Item 1.

Item 2: For k basis functions, there are k constraints in the dual formulation to the linear programming-based approximation formulation (not including positivity constraints). Thus, any non-singular basic feasible solution to the dual will have at most k non-zero variables, *i.e.*, k state-action pairs such that $\phi_a(\mathbf{x}) > 0$. Item 2 holds if k is smaller than the number of states.

Item 3: Consider a simple MDP where every state $\mathbf{x}$ transitions to an initial state $\mathbf{x}_0$ with probability 1, *i.e.*, the transition probabilities are defined by: $P(\mathbf{x}_0 \mid \mathbf{x}', a) = 1$ for all $\mathbf{x}'$ and a.

Now consider the approximate dual LP induced by an approximation architecture with only one basis function, the constant function h_0. Lemma 6.1.1 specifies the only feasibility constraints on the dual variables. Let us select $\phi_a(\mathbf{x}) = \frac{1}{|\mathbf{X}||A|(1-\gamma)}$, clearly a feasible solution. The randomized policy ρ defined in Equation (6.7) becomes the uniform policy: $\rho(a \mid \mathbf{x}) = \frac{1}{|A|}$ for all $\mathbf{x}$.

We now compute the visitation frequencies for ρ according to Equation (6.6):

For $\mathbf{x} \neq \mathbf{x}_0$, we have that:

$$\begin{aligned}
\phi_a^\rho(\mathbf{x}) &= \sum_{t=0}^{\infty} \sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}) P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}') \; ; \\
&= \sum_{\mathbf{x}'} \rho(a \mid \mathbf{x}) P_\rho(\mathbf{x}^{(0)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}') \\
&\qquad + \sum_{t=1}^{\infty} \sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}) P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}') \; ; \\
&= \rho(a \mid \mathbf{x}) \alpha(\mathbf{x}) \; ;
\end{aligned}$$

as $P_\rho(\mathbf{x}^{(t)} = \mathbf{x} \mid \mathbf{x}^{(0)} = \mathbf{x}') = 0$, for all $\mathbf{x} \neq \mathbf{x}_0$, for all $t > 0$.

The visitation frequency for $\mathbf{x}_0$ is given by:

$$\begin{aligned}
\phi_a^\rho(\mathbf{x}_0) &= \sum_{t=0}^{\infty} \sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}_0) P_\rho(\mathbf{x}^{(t)} = \mathbf{x}_0 \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}') \; ; \\
&= \rho(a \mid \mathbf{x}_0) \alpha(\mathbf{x}_0) + \sum_{t=1}^{\infty} \sum_{\mathbf{x}'} \gamma^t \rho(a \mid \mathbf{x}_0) P_\rho(\mathbf{x}^{(t)} = \mathbf{x}_0 \mid \mathbf{x}^{(0)} = \mathbf{x}') \alpha(\mathbf{x}') \; ; \\
&= \rho(a \mid \mathbf{x}_0) \alpha(\mathbf{x}_0) + \rho(a \mid \mathbf{x}_0) \sum_{t=1}^{\infty} \gamma^t \sum_{\mathbf{x}'} \alpha(\mathbf{x}') \; ; \\
&= \rho(a \mid \mathbf{x}_0) \alpha(\mathbf{x}_0) + \frac{\gamma \rho(a \mid \mathbf{x}_0)}{1-\gamma} \; ;
\end{aligned}$$

as $P_\rho(\mathbf{x}^{(t)} = \mathbf{x}_0 \mid \mathbf{x}^{(0)} = \mathbf{x}') = 1$, for all $t > 0$.

Thus, $\phi_a^\rho(\mathbf{x}) \neq \phi_a(\mathbf{x})$ for all $\mathbf{x}$ and a, concluding the proof of Item 3. ∎

A.4.3 Proof of Lemma 6.1.4

First note that, by standard primal-dual results (*e.g.*, [Bertsimas & Tsitsiklis, 1997]), a dual variable is positive, $\phi_a(\mathbf{x}) > 0$, if and only if the primal constraint corresponding to the state $\mathbf{x}$ and the action a is tight:

$$\sum_i w_i h_i(\mathbf{x}) = R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i h_i(\mathbf{x}').$$

Now consider the optimal solution $\widehat{\mathbf{w}}$ to the primal LP in (2.8). The greedy policy with respect to this solution is given by:

$$\mathsf{Greedy}[\mathcal{V}^{\widehat{\mathbf{w}}}](\mathbf{x}) = \arg\max_a \left[R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i \widehat{w}_i h_i(\mathbf{x}') \right]. \tag{A.8}$$

If the constraints for some state $\mathbf{x}$ and for all actions a are loose, then the corresponding dual variable $\phi_a(\mathbf{x})$ is equal to 0, for all actions in all optimal dual solutions corresponding to the primal solution $\widehat{\mathbf{w}}$. Thus, according to Definition 6.1.3 our policies in can select any (randomized) action for this state, including $\mathsf{Greedy}[\mathcal{V}^{\widehat{\mathbf{w}}}](\mathbf{x})$.

We must now consider states $\mathbf{x}$ where our primal constraints are tight for at least some action a. If the constraints are tight for exactly one action, then this is exactly the greedy action in Equation (A.8). Moreover, the corresponding dual variable for this action $\phi_a(\mathbf{x})$ is strictly positive, in all optimal dual solutions corresponding to the primal solution $\widehat{\mathbf{w}}$. Thus, according to Definition 6.1.3 all of our policies must select the action $\mathsf{Greedy}[\mathcal{V}^{\widehat{\mathbf{w}}}](\mathbf{x})$ at state $\mathbf{x}$. In cases where, for some state $\mathbf{x}$, the primal constraints are tight for more than one action, then the $\arg\max_a$ in Equation (A.8) is not unique, and there is a basic feasible dual solution for each possible maximizing action. ∎

A.4.4 Proof of Theorem 6.1.6

Let $\phi_a^{\widehat{\rho}}$ be the true state-action visitation frequencies of policy $\widehat{\rho}$. By Theorem 2.2.1, we can decompose these frequencies into:

$$\phi_a^{\widehat{\rho}}(\mathbf{x}) = \widehat{\rho}(a \mid \mathbf{x}) \phi^{\widehat{\rho}}(\mathbf{x}),$$

where $\phi^{\widehat{\rho}}(\mathbf{x}) = \frac{\phi_a^{\widehat{\rho}}(\mathbf{x})}{\sum_{a'} \phi_{a'}^{\widehat{\rho}}(\mathbf{x})}$.

Now note that we can decompose our optimal solution $\widehat{\phi}_a$ to the approximate dual in a similar manner:

$$\widehat{\phi}_a(\mathbf{x}) = \widehat{\rho}(a \mid \mathbf{x}) \widehat{\phi}(\mathbf{x}),$$

for any policy in $\mathsf{PoliciesOf}[\widehat{\phi}_a]$, as $\widehat{\phi}(\mathbf{x}) = \frac{\widehat{\phi}_a(\mathbf{x})}{\sum_{a'} \widehat{\phi}_{a'}(\mathbf{x})}$ if $\sum_{a'} \widehat{\phi}_{a'}(\mathbf{x}) > 0$, and zero otherwise.

We can now define the difference between these two sets of visitation frequencies:

$$\begin{aligned}
\epsilon_a^{\widehat{\rho}}(\mathbf{x}) &= \widehat{\phi}_a(\mathbf{x}) - \phi_a^{\widehat{\rho}}(\mathbf{x}); \\
&= \widehat{\rho}(a \mid \mathbf{x}) \left(\widehat{\phi}(\mathbf{x}) - \phi^{\widehat{\rho}}(\mathbf{x})\right); \\
&= \widehat{\rho}(a \mid \mathbf{x}) \epsilon^{\widehat{\rho}}(\mathbf{x});
\end{aligned}$$

where we define $\epsilon^{\widehat{\rho}}(\mathbf{x}) = \widehat{\phi}(\mathbf{x}) - \phi^{\widehat{\rho}}(\mathbf{x})$.

As the $\phi_a^{\widehat{\rho}}$ are the true visitation frequencies of policy $\widehat{\rho}$, by Theorem 2.2.1 we know that this is a feasible solution to the exact dual LP. Thus, we have that:

$$\phi^{\widehat{\rho}}(\mathbf{x}) = \alpha(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} \phi^{\widehat{\rho}}(\mathbf{x}') P_{\widehat{\rho}}(\mathbf{x} \mid \mathbf{x}'),$$

where $P_{\widehat{\rho}}(\mathbf{x} \mid \mathbf{x}') = \sum_a \widehat{\rho}(a \mid \mathbf{x}') P(\mathbf{x} \mid \mathbf{x}', a)$. In matrix notation, we have that:

$$\phi^{\widehat{\rho}} = \alpha + \gamma \phi^{\widehat{\rho}} P_{\widehat{\rho}}.$$

As $\phi^{\widehat{\rho}} = \widehat{\phi} - \epsilon^{\widehat{\rho}}$, we have that:

$$\widehat{\phi} - \epsilon^{\widehat{\rho}} = \alpha + \gamma \left(\widehat{\phi} - \epsilon^{\widehat{\rho}}\right) P_{\widehat{\rho}}.$$

Rearranging, we finally get:

$$\begin{aligned}
\epsilon^{\widehat{\rho}} &= \left(\widehat{\phi} - \alpha - \gamma \widehat{\phi} P_{\widehat{\rho}}\right) (I - \gamma P_{\widehat{\rho}})^{-1}; \\
&= \left(\Delta[\widehat{\phi}_a]\right)^{\mathsf{T}} (I - \gamma P_{\widehat{\rho}})^{-1}. \qquad \text{(A.9)}
\end{aligned}$$

Let ϕ_a^* be an optimal solution to the exact dual LP. As ϕ_a^* is feasible in the approximate dual LP in (6.2), we have that:

$$\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x}, a) \geq \sum_{\mathbf{x},a} \phi_a^*(\mathbf{x}) R(\mathbf{x}, a).$$

Similarly, $\phi_a^{\widehat{\rho}}$ is a feasible solution to the exact dual LP in (6.1), thus:

$$\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x}, a) \geq \sum_{\mathbf{x},a} \phi_a^*(\mathbf{x}) R(\mathbf{x}, a) \geq \sum_{\mathbf{x},a} \phi_a^{\widehat{\rho}}(\mathbf{x}) R(\mathbf{x}, a). \qquad \text{(A.10)}$$

From the definition of $\epsilon^{\widehat{\rho}}$, we have that:

$$\begin{aligned}\sum_{\mathbf{x},a} \phi_a^{\widehat{\rho}}(\mathbf{x}) R(\mathbf{x},a) &= \sum_{\mathbf{x}} \phi^{\widehat{\rho}}(\mathbf{x}) R_{\widehat{\rho}}(\mathbf{x}); \\ &= \sum_{\mathbf{x}} \widehat{\phi}(\mathbf{x}) R_{\widehat{\rho}}(\mathbf{x}) - \sum_{\mathbf{x}} \epsilon^{\widehat{\rho}}(\mathbf{x}) R_{\widehat{\rho}}(\mathbf{x});\end{aligned}$$

where $R_{\widehat{\rho}}(\mathbf{x}) = \sum_a \widehat{\rho}(a \mid \mathbf{x}) R(\mathbf{x},a)$. In matrix notation, we have that:

$$(\phi^{\widehat{\rho}})^{\mathsf{T}} R_{\widehat{\rho}} = (\widehat{\phi})^{\mathsf{T}} R_{\widehat{\rho}} \; - \; (\epsilon^{\widehat{\rho}})^{\mathsf{T}} R_{\widehat{\rho}}.$$

Substituting $\epsilon^{\widehat{\rho}}$ from Equation (A.9), we have that:

$$(\phi^{\widehat{\rho}})^{\mathsf{T}} R_{\widehat{\rho}} = (\widehat{\phi})^{\mathsf{T}} R_{\widehat{\rho}} \; - \; \left(\Delta[\widehat{\phi}_a]\right)^{\mathsf{T}} \; (I - \gamma P_{\widehat{\rho}})^{-1} \, R_{\widehat{\rho}}.$$

Note that $\mathcal{V}_{\widehat{\rho}} = (I - \gamma P_{\widehat{\rho}})^{-1} \, R_{\widehat{\rho}}$. Thus:

$$(\phi^{\widehat{\rho}})^{\mathsf{T}} R_{\widehat{\rho}} = (\widehat{\phi})^{\mathsf{T}} R_{\widehat{\rho}} \; - \; \left(\Delta[\widehat{\phi}_a]\right)^{\mathsf{T}} \; \mathcal{V}_{\widehat{\rho}}.$$

Rearranging, we obtain that:

$$\begin{aligned}\sum_{\mathbf{x},a} \phi_a^{\widehat{\rho}}(\mathbf{x}) R(\mathbf{x},a) + \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x}) \; V_{\widehat{\rho}}(\mathbf{x});& \\ &= \sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x},a); \\ &\geq \sum_{\mathbf{x},a} \phi_a^{*}(\mathbf{x}) R(\mathbf{x},a); \\ &\geq \sum_{\mathbf{x},a} \phi_a^{\widehat{\rho}}(\mathbf{x}) R(\mathbf{x},a) \\ &= \sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x},a) - \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x}) \; V_{\widehat{\rho}}(\mathbf{x}); \qquad \text{(A.11)}\end{aligned}$$

where the inequalities are substitutions from Equation (A.10).

By the strong duality theorem for LPs, we have that:

$$\begin{aligned}
\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x},a) &= \sum_{\mathbf{x}} \alpha(\mathbf{x}) V^{\widehat{\mathbf{w}}}(\mathbf{x}); \\
\sum_{\mathbf{x},a} \phi^*_a(\mathbf{x}) R(\mathbf{x},a) &= \sum_{\mathbf{x}} \alpha(\mathbf{x}) V^{*}(\mathbf{x}); \\
\sum_{\mathbf{x},a} \phi^{\widehat{\rho}}_a(\mathbf{x}) R(\mathbf{x},a) &= \sum_{\mathbf{x}} \alpha(\mathbf{x}) V_{\widehat{\rho}}(\mathbf{x}).
\end{aligned}$$

Substituting these results into Equation (A.11), we first obtain:

$$\begin{aligned}
&\sum_{\mathbf{x},a} \phi^{\widehat{\rho}}_a(\mathbf{x}) R(\mathbf{x},a) + \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x})\ V_{\widehat{\rho}}(\mathbf{x}); \\
&\qquad = \sum_{\mathbf{x}} \alpha(\mathbf{x}) V_{\widehat{\rho}}(\mathbf{x}) + \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x})\ V_{\widehat{\rho}}(\mathbf{x}); \\
&\qquad \geq \sum_{\mathbf{x},a} \phi^*_a(\mathbf{x}) R(\mathbf{x},a); \\
&\qquad = \sum_{\mathbf{x}} \alpha(\mathbf{x}) V^{*}(\mathbf{x}); \qquad (A.12)
\end{aligned}$$

yielding Equation (6.9) when we note that, for each state $\mathbf{x}$, $V^*(\mathbf{x}) \geq \mathcal{V}_{\widehat{\rho}}(\mathbf{x})$ by the optimality of $\mathcal{V}^*$.

Substituting the strong duality results into Equation (A.11) again, we also obtain:

$$\begin{aligned}
&\sum_{\mathbf{x},a} \phi^*_a(\mathbf{x}) R(\mathbf{x},a) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) V^{*}(\mathbf{x}); \\
&\qquad \geq \sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x},a) - \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x})\ V_{\widehat{\rho}}(\mathbf{x}); \\
&\qquad = \sum_{\mathbf{x}} \alpha(\mathbf{x}) V^{\widehat{\mathbf{w}}}(\mathbf{x}) - \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x})\ V_{\widehat{\rho}}(\mathbf{x}). \qquad (A.13)
\end{aligned}$$

Equation (6.10) now follows by noting that Equation (A.13) holds for any $\widehat{\rho} \in \mathsf{PoliciesOf}[\widehat{\phi}_a]$, and that $V^{\widehat{\mathbf{w}}}(\mathbf{x}) \geq \mathcal{V}^*(\mathbf{x})$ for every state $\mathbf{x}$ (as shown by de Farias and Van Roy [2001a]). ▮

A.4.5 Proof of Theorem 6.1.7

First, note that, by the feasibility of $\widehat{\phi}_a$ in the approximate dual formulation, we have that, for any set of weights $\mathbf{w}$:

$$\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) \sum_i w_i h_i(\mathbf{x}) = \sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i h_i(\mathbf{x}) + \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') \sum_i w_i h_i(\mathbf{x});$$

where this equation is just a weighted combination of the flow constraints in Equation (6.3). Rearranging, we obtain:

$$\begin{aligned}&\textstyle\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) \sum_i w_i h_i(\mathbf{x}) \\ &\quad - \textstyle\sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i h_i(\mathbf{x}) - \gamma \sum_{\mathbf{x}',a'} \phi_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') \sum_i w_i h_i(\mathbf{x}) = 0.\end{aligned} \tag{A.14}$$

Theorem 6.1.6 says that we must bound:

$$(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} = \sum_{\mathbf{x}} \Delta[\widehat{\phi}_a](\mathbf{x}) \; \mathcal{V}_{\widehat{\rho}}(\mathbf{x}),$$

or equivalently:

$$(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} = \sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) \mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_{\mathbf{x}} \alpha(\mathbf{x}) \mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_{\mathbf{x}',a'} \widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') \mathcal{V}_{\widehat{\rho}}(\mathbf{x}).$$

Subtracting Equation (A.14), we obtain:

$$\begin{aligned}(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} = &\textstyle\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) \left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right] - \sum_{\mathbf{x}} \alpha(\mathbf{x}) \left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right] \\ &- \gamma \textstyle\sum_{\mathbf{x}',a'} \widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') \left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right];\end{aligned} \tag{A.15}$$

for any set of weights $\mathbf{w}$.

We can now prove the first part of our theorem by choosing the set of weights that define the minimum in Equation (6.11), and thus noting that:

$$\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \leq \varepsilon^{\infty}_{\widehat{\rho}}, \; \forall \mathbf{x}.$$

Substituting into Equation (A.15), we obtain:

$$\begin{aligned}(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} &\leq \varepsilon^{\infty}_{\widehat{\rho}} \left[\sum_{\mathbf{x},a} \left|\widehat{\phi}_a(\mathbf{x})\right| + \sum_{\mathbf{x}} |\alpha(\mathbf{x})| + \gamma \left|\sum_{\mathbf{x}',a'} \widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a')\right|\right];\\
&= \varepsilon^{\infty}_{\widehat{\rho}} \left[\frac{1}{1-\gamma} + 1 + \frac{\gamma}{1-\gamma}\right];\\
&= \frac{2\varepsilon^{\infty}_{\widehat{\rho}}}{1-\gamma};\end{aligned}$$

concluding the proof of the first part of the theorem.

For the proof of the second part, we multiply each term in Equation (A.15) by $\frac{L(\mathbf{x})}{L(\mathbf{x})}$:

$$\begin{aligned}(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} = \textstyle\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) \frac{L(\mathbf{x})}{L(\mathbf{x})} \left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right] - \sum_{\mathbf{x}} \alpha(\mathbf{x}) \frac{L(\mathbf{x})}{L(\mathbf{x})} \left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right]\\
-\gamma \textstyle\sum_{\mathbf{x}',a'} \widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') \frac{L(\mathbf{x})}{L(\mathbf{x})} \left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right].\end{aligned}$$

Substituting the weighted max-norm error $\varepsilon^{\infty,1/L}_{\widehat{\rho}}$ in place of each $\frac{1}{L(\mathbf{x})}\left[\mathcal{V}_{\widehat{\rho}}(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right]$, we obtain:

$$\begin{aligned}(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} &\leq \varepsilon^{\infty,1/L}_{\widehat{\rho}} \left[\sum_{\mathbf{x},a} \left|\widehat{\phi}_a(\mathbf{x}) L(\mathbf{x})\right| + \sum_{\mathbf{x}} |\alpha(\mathbf{x}) L(\mathbf{x})|\right.\\
&\qquad\qquad \left. + \gamma \sum_{\mathbf{x}',a'} \left|\widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') L(\mathbf{x})\right|\right];\\
&= \varepsilon^{\infty,1/L}_{\widehat{\rho}} \left[\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) L(\mathbf{x}) + \sum_{\mathbf{x}} \alpha(\mathbf{x}) L(\mathbf{x})\right.\\
&\qquad\qquad \left. + \gamma \sum_{\mathbf{x}',a'} \widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') L(\mathbf{x})\right];\\
&= \varepsilon^{\infty,1/L}_{\widehat{\rho}} \left[\sum_{\mathbf{x},a} \widehat{\phi}_a(\mathbf{x}) L(\mathbf{x}) + \sum_{\mathbf{x}} \alpha(\mathbf{x}) L(\mathbf{x})\right.\\
&\qquad\qquad \left. + \left(1 - \frac{2}{1-\kappa} + \frac{2}{1-\kappa}\right) \gamma \sum_{\mathbf{x}',a'} \widehat{\phi}_{a'}(\mathbf{x}') \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{x}', a') L(\mathbf{x})\right];\end{aligned}$$

where we remove the absolute values in the second equality because all terms are non-negative. Using the Lyapunov condition in Equation (6.14), we can change

$$\left(\frac{2}{1-\kappa}\right)\gamma\sum_{\mathbf{x}',a'}\widehat{\phi}_{a'}(\mathbf{x}')\sum_{\mathbf{x}}P(\mathbf{x}\mid\mathbf{x}',a')L(\mathbf{x}),$$

which is equal to

$$\left(\frac{2}{1-\kappa}\right)\gamma\sum_{\mathbf{x}'}\widehat{\phi}(\mathbf{x}')\sum_{\mathbf{x}}P_{\widehat{\rho}}(\mathbf{x}\mid\mathbf{x}')L(\mathbf{x}),$$

into the larger

$$\left(\frac{2\kappa}{1-\kappa}\right)\sum_{\mathbf{x}}\widehat{\phi}(\mathbf{x})L(\mathbf{x}),$$

which is equal to

$$\left(\frac{2\kappa}{1-\kappa}\right)\sum_{\mathbf{x},a}\widehat{\phi}_a(\mathbf{x})L(\mathbf{x}),$$

obtaining:

$$\begin{aligned}(\Delta[\widehat{\phi}_a])^\intercal\mathcal{V}_{\widehat{\rho}} \;\leq\; & \varepsilon_{\widehat{\rho}}^{\infty,1/L}\Bigg[\left(1+\frac{2\kappa}{1-\kappa}\right)\sum_{\mathbf{x},a}\widehat{\phi}_a(\mathbf{x})L(\mathbf{x})+\sum_{\mathbf{x}}\alpha(\mathbf{x})L(\mathbf{x})\\ &\qquad+\left(1-\frac{2}{1-\kappa}\right)\gamma\sum_{\mathbf{x}',a'}\widehat{\phi}_{a'}(\mathbf{x}')\sum_{\mathbf{x}}P(\mathbf{x}\mid\mathbf{x}',a')L(\mathbf{x})\Bigg];\\ =\; & \varepsilon_{\widehat{\rho}}^{\infty,1/L}\Bigg[\left(\frac{1+\kappa}{1-\kappa}\right)\sum_{\mathbf{x},a}\widehat{\phi}_a(\mathbf{x})L(\mathbf{x})+\sum_{\mathbf{x}}\alpha(\mathbf{x})L(\mathbf{x})\\ &\qquad+\left(-\frac{1+\kappa}{1-\kappa}\right)\gamma\sum_{\mathbf{x}',a'}\widehat{\phi}_{a'}(\mathbf{x}')\sum_{\mathbf{x}}P(\mathbf{x}\mid\mathbf{x}',a')L(\mathbf{x})\Bigg].\end{aligned}$$

As the Lyapunov function is in the space of our basis functions, we can use Equation (A.14) with weights $\mathbf{w}^L$ to substitute the term:

$$\left(\frac{1+\kappa}{1-\kappa}\right)\sum_{\mathbf{x},a}\widehat{\phi}_a(\mathbf{x})L(\mathbf{x})+\left(-\frac{1+\kappa}{1-\kappa}\right)\gamma\sum_{\mathbf{x}',a'}\widehat{\phi}_{a'}(\mathbf{x}')\sum_{\mathbf{x}}P(\mathbf{x}\mid\mathbf{x}',a')L(\mathbf{x})$$

with $\frac{1+\kappa}{1-\kappa}\sum_{\mathbf{x}}\alpha(\mathbf{x})L(\mathbf{x})$, obtaining:

$$\begin{aligned}(\Delta[\widehat{\phi}_a])^\intercal \mathcal{V}_{\widehat{\rho}} &\leq \varepsilon_{\widehat{\rho}}^{\infty,1/L} \left(\frac{1+\kappa}{1-\kappa}+1\right) \sum_{\mathbf{x}} \alpha(\mathbf{x}) L(\mathbf{x}); \\ &= \varepsilon_{\widehat{\rho}}^{\infty,1/L} \frac{2}{1-\kappa} \sum_{\mathbf{x}} \alpha(\mathbf{x}) L(\mathbf{x});\end{aligned}$$

thus concluding our proof. ▮

A.4.6 Proof of Lemma 6.2.4

By contradiction: assume that there exists a set of global visitation frequencies $\widehat{\phi}_a(\mathbf{x})$ satisfying the flow constraints in Equation (6.3), such that $\widehat{\phi}_a(\mathbf{x})$ and μ_a^* are not consistent flows, and:

$$\sum_a \sum_{\mathbf{x}} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x}, a) > \sum_a \sum_{\mathbf{x}} \phi_a^*(\mathbf{x}) R(\mathbf{x}, a). \qquad \text{(A.16)}$$

Let $\widehat{\mu}_a$ be the marginal visitation frequencies associated with $\widehat{\phi}_a$ as defined in Equations (6.23) and (6.24).

Each $\widehat{\mu}_a$ is guaranteed to be non-negative, by the non-negativity of $\widehat{\phi}_a$. The derivation in Section 6.2.2 shows that $\widehat{\mu}_a$ must satisfy the factored flow constraints.

As ϕ_a^* and μ_a^* are consistent flows, Equation (6.25) implies that:

$$\sum_a \sum_{\mathbf{x}} \phi_a^*(\mathbf{x}) R(\mathbf{x}, a) = \sum_{j=1}^r \sum_a \sum_{\mathbf{w}_j^a \in \mathrm{Dom}[\mathbf{W}_j^a]} \mu_a^*(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a) \ .$$

Similarly, $\widehat{\mu}_a$ and $\widehat{\phi}_a$ are consistent flows by definition, yielding:

$$\sum_a \sum_{\mathbf{x}} \widehat{\phi}_a(\mathbf{x}) R(\mathbf{x}, a) = \sum_{j=1}^r \sum_a \sum_{\mathbf{w}_j^a \in \mathrm{Dom}[\mathbf{W}_j^a]} \widehat{\mu}_a(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a) \ .$$

Substituting these two equations into Equation (A.16), we obtain:

$$\sum_{j=1}^r \sum_a \sum_{\mathbf{w}_j^a \in \mathrm{Dom}[\mathbf{W}_j^a]} \mu_a^*(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a) < \sum_{j=1}^r \sum_a \sum_{\mathbf{w}_j^a \in \mathrm{Dom}[\mathbf{W}_j^a]} \widehat{\mu}_a(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a) \ ;$$

contradicting the optimality of μ_a^*. ▮

A.5 Proof of Theorem 13.3.2

We start the proof of our Theorem 13.3.2 with a lemma that measures the effect of sampling on the objective function:

Lemma A.5.1 *Consider the following class-based value functions (each with k parameters): $\widehat{\mathcal{V}}$ obtained from the LP over all possible worlds Ω by minimizing Equation (13.8) subject to the constraints in Equation (13.5); and $\widetilde{\mathcal{V}}$ obtained by solving the class-level LP in (13.11) with constraints only for a set $\mathcal{D}_{\leq n}$ of m worlds sampled from $P_{\leq n}(\omega)$, i.e., only sampled from the set of worlds $\Omega_{\leq n}$ with at most n objects, for any $n \geq 1$. For any $\delta > 0$ and $\varepsilon > 0$, if the number of sampled worlds m is:*

$$m \geq 2\left[\left\lceil\left(\frac{16k}{\varepsilon}\right)^2\right\rceil \ln(2k+1) + \ln\frac{8}{\delta}\right]\left(\frac{8}{\varepsilon}\right)^2,$$

then:

$$\mathbf{E}_\Omega\left[\widetilde{\mathcal{V}}\right] - \mathbf{E}_\Omega\left[\widehat{\mathcal{V}}\right] \leq \frac{2R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[\varepsilon n + \left(n(1-\varepsilon) + \frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n}\right], \tag{A.17}$$

with probability at least $1 - \frac{\delta}{2}$; where $\mathbf{E}_\Omega\left[\mathcal{V}\right] = \sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)P^0_\omega(\mathbf{x})\mathcal{V}_\omega(\mathbf{x})$, and R^o_{max} is the maximum per-object reward.

Proof:

As described in Section 13.2.2, we can decompose the probability of a world into:

$$P(\omega) = P(\sharp)P(\omega_\sharp \mid \sharp).$$

Substituting this formulation into the left side of Equation (A.17), we obtain:

$$\mathbf{E}_\Omega\left[\widetilde{\mathcal{V}}\right] - \mathbf{E}_\Omega\left[\widehat{\mathcal{V}}\right] = \sum_{i=1}^{\infty}\sum_{\omega\in\Omega_i}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P(\sharp = i)P(\omega \mid \sharp = i)P^0_\omega(\mathbf{x})\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) - \widehat{\mathcal{V}}_\omega(\mathbf{x})\right),$$

or equivalently:

$$\begin{aligned}\mathbf{E}_{\Omega}\left[\widetilde{\mathcal{V}}\right]-\mathbf{E}_{\Omega}\left[\widehat{\mathcal{V}}\right] &= \sum_{i=1}^{n}\sum_{\omega\in\Omega_i}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P(\sharp=i)P(\omega\mid\sharp=i)P^0_\omega(\mathbf{x})\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})-\widehat{\mathcal{V}}_\omega(\mathbf{x})\right)+\\ &\quad \sum_{j=n+1}^{\infty}\sum_{\omega\in\Omega_j}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P(\sharp=j)P(\omega\mid\sharp=j)P^0_\omega(\mathbf{x})\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})-\widehat{\mathcal{V}}_\omega(\mathbf{x})\right).\end{aligned} \tag{A.18}$$

We will bound each term in Equation (A.18) in turn.

Let us start by considering the first term on the righthand side of Equation (A.18), which we can rewrite as:

$$\sum_{i=1}^{n}\sum_{\omega\in\Omega_i}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P(\sharp=i)P(\omega\mid\sharp=i)P^0_\omega(\mathbf{x})\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})-\widehat{\mathcal{V}}_\omega(\mathbf{x})\right)=\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}}-\widehat{\mathcal{V}}\right]\sum_{\omega'\in\Omega_{\leq n}}P(\omega'). \tag{A.19}$$

In addition, recall that our class-based LP minimizes:

$$\mathbf{E}_{\mathcal{D}_{\leq n}}\left[\mathcal{V}\right]=\sum_{\omega\in\mathcal{D}_{\leq n},\mathbf{x}\in\mathbf{X}_\omega}P(\omega)P^0_\omega(\mathbf{x})\mathcal{V}_\omega(\mathbf{x}),$$

which is a sample-based approximation to the expectation $\mathbf{E}_{\Omega_{\leq n}}[\mathcal{V}]$, and that $\widetilde{\mathcal{V}}$ is an optimal solution to this linear program. $\widehat{\mathcal{V}}$, on the other hand, satisfies the constraints for all worlds, including of course the constraints for worlds in $\mathcal{D}_{\leq n}$. Thus, $\widehat{\mathcal{V}}$ is clearly a feasible solution for our class-level LP, consequently:

$$\mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}}\right]\leq\mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widehat{\mathcal{V}}\right].$$

As we want to bound $\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}}-\widehat{\mathcal{V}}\right]$, and we now know that $\mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}}-\widehat{\mathcal{V}}\right]\leq 0$, it is sufficient to bound:

$$\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}}-\widehat{\mathcal{V}}\right]-\mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}}-\widehat{\mathcal{V}}\right].$$

If the weights of our approximations $\widetilde{\mathcal{V}}$ and $\widehat{\mathcal{V}}$ were fixed, we could use Hoeffding's inequality to bound these terms, as they are a difference between an expectation and the sample mean. However, our LP picks the basis function weights after the worlds are sampled, and thus Hoeffding's no longer holds, as an adversarial could pick weights that maximize the error. Fortunately, we can compute such a bound for the worst possible (most adversarial) choice of weights, with high probability, using the framework of Pollard [1984]. This framework bounds the number of ways that the weights can be picked by using a covering number. A union bound is the used to combine the probability of a large deviation for all possible choices of weights. Pollard [1984] then proves a Hoeffding-style inequality using this covering number:

$$\begin{aligned} P\left(\exists\, \widetilde{\mathbf{w}}, \widehat{\mathbf{w}} :\ \mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] - \mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] > \varepsilon\right) \\ \leq\ 2\,\mathbf{E}\left[\mathcal{N}\left(\varepsilon/16, L^{(k)}, \mathcal{D}_{\leq n}\right)\right] e^{\frac{-\varepsilon^2 m}{128\|\widetilde{\mathcal{V}}-\widehat{\mathcal{V}}\|_S^2}}, \end{aligned} \tag{A.20}$$

where $\widetilde{\mathbf{w}}$ and $\widehat{\mathbf{w}}$ are the parameters of $\widetilde{\mathcal{V}}$ and $\widehat{\mathcal{V}}$, respectively; $\mathcal{N}\left(\varepsilon/16, L^{(k)}, \mathcal{D}_{\leq n}\right)$ is the *covering number* of a linear function with k parameters [Pollard, 1984]; and the span norm $\|\cdot\|_S$ is defined to be $\|\mathcal{V}\|_S = \max_{\mathbf{x}} \mathcal{V}(\mathbf{x}) - \min_{\mathbf{x}} \mathcal{V}(\mathbf{x})$.

The bound of Pollard [1984] thus depends on the covering number of our linear function parameterized by $\mathbf{w}$. We can bound this covering number as a function of the number of basis functions, the maximum value of each basis, and the magnitude of the weights, using the result of Zhang [2002] Theorem 3:

$$\ln \mathcal{N}\left(\varepsilon/16, L^{(k)}, \mathcal{D}_{\leq n}\right) \leq \left\lceil \left(\frac{16\, a_{\leq n}\, \|\widetilde{\mathbf{w}} - \widehat{\mathbf{w}}\|_1}{\varepsilon}\right)^2 \right\rceil \ln(2k+1), \tag{A.21}$$

where

$$a_{\leq n} = \max_{C} \max_{h_i^C \in \mathsf{Basis}[C]} \max_{\omega \in \Omega_{\leq n}} |\mathcal{O}[\omega][C]| \left\|h_i^C\right\|_\infty .$$

Using Assumption 13.3.1, we obtain the following bounds:

$$\begin{aligned} a_{\leq n} &\leq n; \\ \|\widetilde{\mathbf{w}} - \widehat{\mathbf{w}}\|_1 &\leq \frac{2kR^o_{max}}{1-\gamma}; \\ \left\|\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right\|_S &\leq \frac{2nR^o_{max}}{1-\gamma}. \end{aligned}$$

By substituting these bounds into Equation (A.20), we obtain the bound:

$$\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] - \mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] \leq \varepsilon,$$

for a number of sampled worlds:

$$m \geq 2\left[\left\lceil\left(\frac{32nkR^o_{max}}{\varepsilon(1-\gamma)}\right)^2\right\rceil \ln(2k+1) + \ln\frac{8}{\delta}\right]\left(\frac{16nR^o_{max}}{\varepsilon(1-\gamma)}\right)^2,$$

with probability at least $1 - \frac{\delta}{2}$. Recasting ε as

$$\varepsilon\frac{2nR^o_{max}}{(1-\gamma)},$$

we obtain:

$$\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] - \mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] \leq \varepsilon\frac{2nR^o_{max}}{(1-\gamma)}, \tag{A.22}$$

for a number of sampled worlds:

$$m \geq 2\left[\left\lceil\left(\frac{16k}{\varepsilon}\right)^2\right\rceil \ln(2k+1) + \ln\frac{8}{\delta}\right]\left(\frac{8}{\varepsilon}\right)^2, \tag{A.23}$$

with probability at least $1-\frac{\delta}{2}$, which is the number of samples that appears on the statement of this lemma.

Substituting Equation (A.22) into Equation (A.19) we obtain:

$$\begin{aligned}
\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] \sum_{\omega' \in \Omega_{\leq n}} P(\omega') &\leq \left(\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] - \mathbf{E}_{\mathcal{D}_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right]\right) \sum_{\omega' \in \Omega_{\leq n}} P(\omega'); \\
&\leq \varepsilon \frac{2nR^o_{max}}{(1-\gamma)} \sum_{\omega' \in \Omega_{\leq n}} P(\omega').
\end{aligned}$$

We can bound the term $\sum_{\omega' \in \Omega_{\leq n}} P(\omega')$ by using Assumption 13.2.1:

$$\begin{aligned}
\sum_{\omega' \in \Omega_{\leq n}} P(\omega') &= \sum_{i=1}^{n} \sum_{\omega \in \Omega_i} P(\sharp = i) P(\omega \mid \sharp = i); \\
&= \sum_{i=1}^{n} P(\sharp = i) \\
&\leq \sum_{i=1}^{n} \kappa_\sharp e^{-\lambda_\sharp i}; \\
&\leq \int_0^n \kappa_\sharp e^{-\lambda_\sharp x} dx; \\
&= \frac{\kappa_\sharp}{\lambda_\sharp}\left[1 - e^{-\lambda_\sharp n}\right]. \qquad \text{(A.24)}
\end{aligned}$$

Concluding the bound of the first term on the righthand side of Equation (A.18) as:

$$\mathbf{E}_{\Omega_{\leq n}}\left[\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right] \sum_{\omega' \in \Omega_{\leq n}} P(\omega') \leq \varepsilon \frac{2nR^o_{max}}{(1-\gamma)} \frac{\kappa_\sharp}{\lambda_\sharp}\left[1 - e^{-\lambda_\sharp n}\right]. \qquad \text{(A.25)}$$

We can now focus on the second term on the righthand side of Equation (A.18):

$$\begin{aligned}
\sum_{j=n+1}^{\infty}\sum_{\omega\in\Omega_j}\sum_{\mathbf{x}\in\mathbf{X}_\omega} & P(\sharp=j)P(\omega\mid\sharp=j)P^0_\omega(\mathbf{x})\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})-\widehat{\mathcal{V}}_\omega(\mathbf{x})\right) \\
&\leq \sum_{j=n+1}^{\infty}\sum_{\omega\in\Omega_j}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P(\sharp=j)P(\omega\mid\sharp=j)P^0_\omega(\mathbf{x})\left\|\widetilde{\mathcal{V}}_\omega-\widehat{\mathcal{V}}_\omega\right\|_\infty; \\
&\leq \sum_{j=n+1}^{\infty}\sum_{\omega\in\Omega_j}\sum_{\mathbf{x}\in\mathbf{X}_\omega} P(\sharp=j)P(\omega\mid\sharp=j)P^0_\omega(\mathbf{x})\frac{2jR^o_{max}}{1-\gamma}; \\
&= \frac{2R^o_{max}}{1-\gamma}\sum_{j=n+1}^{\infty} j\ P(\sharp=j),
\end{aligned}$$

where the first inequality bounds the difference $\widetilde{\mathcal{V}}_\omega(\mathbf{x})-\widehat{\mathcal{V}}_\omega(\mathbf{x})$ by the max-norm term $\left\|\widetilde{\mathcal{V}}_\omega-\widehat{\mathcal{V}}_\omega\right\|_\infty$. This max-norm term is bounded in the second inequality by $\frac{2jR^o_{max}}{1-\gamma}$, as every world in Ω_j has at most j objects, and thus each value function is bounded by $\frac{jR^o_{max}}{1-\gamma}$, and the difference between two value functions is no larger than 2 times this bound.

We can now focus on $\sum_{j=n+1}^{\infty} j\ P(\sharp=j)$. Using Assumption 13.3.1, we obtain the following bound:

$$\begin{aligned}
\sum_{j=n+1}^{\infty} j\ P(\sharp=j) &\leq \sum_{j=n+1}^{\infty} j\ \kappa_\sharp e^{-\lambda_\sharp j}; \\
&\leq \int_n^{\infty} x\kappa_\sharp e^{-\lambda_\sharp x}dx; \\
&= \frac{\kappa_\sharp}{\lambda_\sharp}\left[n+\frac{1}{\lambda_\sharp}\right]e^{-\lambda_\sharp n}. \qquad \text{(A.26)}
\end{aligned}$$

We thus conclude the bound on the second term on the righthand side of Equation (A.18) by:

$$\frac{2R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[n+\frac{1}{\lambda_\sharp}\right]e^{-\lambda_\sharp n}. \qquad \text{(A.27)}$$

Our final result follows from the sum of Equation (A.25) and Equation (A.27), and rearranging the terms. ∎

In order to prove our main result we use a theorem by de Farias and Van Roy [2001b] that considers linear systems with large number of constraints that are represented approximately by a sampled subset of these constraints. Our class-level LP is an example of such a system. If we solve an LP only considering this subset of the constraints, some of the other constraints may be violated. Their theorem bounds the "number" of such constraints that are violated:

Theorem A.5.2 (de Farias and Van Roy [2001b]) *Consider a (satisfiable) set of linear constraints:*

$$a_z^\intercal \mathbf{w} + b_z \geq 0, \ \forall z \in \mathcal{Z},$$

where $\mathbf{w} \in \mathbb{R}^k$ *and* $\mathcal{Z}$ *is a set of constraint indices.*

For any $\delta > 0$ *and* $\varepsilon > 0$*, and*

$$m \geq \frac{4}{\varepsilon}\left(k \ln \frac{12}{\varepsilon} + \ln \frac{4}{\delta}\right),$$

a set $\widehat{\mathcal{Z}}$ *of* m *i.i.d. random variables sampled from* $\mathcal{Z}$ *according to a distribution* ψ *satisfies:*

$$\sup_{\{\mathbf{w} \mid \ a_z^\intercal \mathbf{w} + b_z \geq 0, \ \forall z \in \widehat{\mathcal{Z}}\}} \sum_{z \in \mathcal{Z}} \psi(z) \mathbb{1}\left(a_z^\intercal \mathbf{w} + b_z < 0\right) \ \leq \ \varepsilon,$$

with probability at least $1 - \frac{\delta}{2}$. ∎

We can now prove our main theorem:

Theorem A.5.3 *Consider the following class-based value functions (each with* k *parameters):* $\widehat{\mathcal{V}}$ *obtained from the LP over all possible worlds* Ω *by minimizing Equation (13.8) subject to the constraints in Equation (13.5); and* $\widetilde{\mathcal{V}}$ *obtained by solving the class-level LP in (13.11) with constraints only for a set* $\mathcal{D}_{\leq n}$ *of* m *worlds sampled from* $P_{\leq n}(\omega)$*, i.e., only sampled from the set of worlds* $\Omega_{\leq n}$ *with at most* n *objects, for any* $n \geq 1$*. Let* $\mathcal{V}^*$ *be the optimal value function of the meta-MDP* Π_{meta} *over all possible worlds* Ω*. For any* $\delta > 0$ *and* $\varepsilon > 0$*, if the number of sampled worlds* m *is:*

$$m \geq \frac{4}{\varepsilon(1-\gamma)}\left(k \ln \frac{12}{\varepsilon(1-\gamma)} + \ln \frac{4}{\delta}\right) + 2\left[\left\lceil\left(\frac{16k}{\varepsilon}\right)^2\right\rceil \ln(2k+1) + \ln \frac{8}{\delta}\right]\left(\frac{8}{\varepsilon}\right)^2,$$

the error introduced by sampling worlds is bounded by:

$$\left\|\widetilde{\mathcal{V}}-\mathcal{V}^*\right\|_{1,P_\Omega} \leq \left\|\widehat{\mathcal{V}}-\mathcal{V}^*\right\|_{1,P_\Omega} + \frac{6R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[\varepsilon n + \left(n(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n}\right];$$

with probability at least $1-\delta$*; where* $\|\mathcal{V}\|_{1,P_\Omega} = \sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)P^0_\omega(\mathbf{x})\,|\mathcal{V}_\omega(\mathbf{x})|$*, and* R^o_{max} *is the maximum per-object reward.*

Proof:

For any vector $\mathcal{V}$, we denote its positive and negative parts by:

$$\mathcal{V}^+ = \max(\mathcal{V},0), \text{ and } \mathcal{V}^- = \max(-\mathcal{V},0),$$

where the maximization is computed componentwise.

Let P_{π^*}, R_{π^*}, and $\mathcal{T}_{\pi^*}$ be, respectively, the transition model, reward function, and Bellman operator associated with π^*, the optimal policy of the meta-MDP $\Pi_{\texttt{meta}}$. As noted by de Farias and Van Roy [2001b], Theorem 3.1, we have that:

$$\begin{aligned}
\left\|\widetilde{\mathcal{V}}-\mathcal{V}^*\right\|_{1,P_\Omega} &= (P_\Omega)^\intercal\left|(I-\gamma P_{\pi^*})^{-1}((I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*})\right|;\\
&\leq (P_\Omega)^\intercal\left|(I-\gamma P_{\pi^*})^{-1}\right|\left|(I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*}\right|;\\
&= (P_\Omega)^\intercal(I-\gamma P_{\pi^*})^{-1}\left|(I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*}\right|;\\
&= (P_\Omega)^\intercal(I-\gamma P_{\pi^*})^{-1}\left[((I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*})^+ + ((I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*})^-\right];\\
&= (P_\Omega)^\intercal(I-\gamma P_{\pi^*})^{-1}\Big[((I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*})^+ - ((I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*})^-\\
&\qquad\qquad +2((I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*})^-\Big];\\
&= (P_\Omega)^\intercal(I-\gamma P_{\pi^*})^{-1}\left[(I-\gamma P_{\pi^*})\widetilde{\mathcal{V}}-R_{\pi^*}+2(\widetilde{\mathcal{V}}-\mathcal{T}_{\pi^*}\widetilde{\mathcal{V}})^-\right];\\
&= (P_\Omega)^\intercal(\widetilde{\mathcal{V}}-\mathcal{V}^*)+2\,(P_\Omega)^\intercal(I-\gamma P_{\pi^*})^{-1}(\widetilde{\mathcal{V}}-\mathcal{T}_{\pi^*}\widetilde{\mathcal{V}})^-. \qquad \text{(A.28)}
\end{aligned}$$

The left side of Equation (A.28) is exactly the term we are bounding in this theorem. We will obtain this bound by bounding each term on the righthand side of Equation (A.28) in turn.

Let us first consider the term $(P_\Omega)^\intercal(\widetilde{\mathcal{V}}-\mathcal{V}^*)$, which is equivalent to:

$$(P_\Omega)^\intercal(\widetilde{\mathcal{V}}-\mathcal{V}^*) = \mathbf{E}_\Omega\left[\widetilde{\mathcal{V}}\right]-\mathbf{E}_\Omega\left[\mathcal{V}^*\right].$$

We can bound this term using the result in our Lemma A.5.1:

$$\mathbf{E}_\Omega\left[\widetilde{\mathcal{V}}\right] - \mathbf{E}_\Omega\left[\mathcal{V}^*\right] \leq \mathbf{E}_\Omega\left[\widehat{\mathcal{V}}\right] - \mathbf{E}_\Omega\left[\mathcal{V}^*\right] + \frac{2R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[\varepsilon n + \left(n(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n}\right],$$

with probability at least $1-\frac{\delta}{2}$. As $\widehat{\mathcal{V}}$ satisfies the constraints for all worlds Ω in Equation (13.5), we have that $\widehat{\mathcal{V}} \geq \mathcal{V}^*$, componentwise (as shown by de Farias and Van Roy [2001a] for general MDPs). Thus:

$$\begin{aligned}(P_\Omega)^\intercal(\widetilde{\mathcal{V}}-\mathcal{V}^*) &= \mathbf{E}_\Omega\left[\widetilde{\mathcal{V}}\right] - \mathbf{E}_\Omega\left[\mathcal{V}^*\right]; \\ &\leq \mathbf{E}_\Omega\left[\widehat{\mathcal{V}}\right] - \mathbf{E}_\Omega\left[\mathcal{V}^*\right] + \frac{2R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[\varepsilon n + \left(n(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n}\right]; \\ &= \left\|\widehat{\mathcal{V}}-\mathcal{V}^*\right\|_{1,P_\Omega} + \frac{2R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[\varepsilon n + \left(n(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n}\right].\end{aligned} \tag{A.29}$$

Equation (A.29) gives us a bound on the first term on the righthand side of Equation (A.28). We must now bound the second term in this equation. Let ϕ^* be the visitation frequencies of π^*, the optimal policy of the meta-MDP for the worlds in Ω. For the starting-state distribution P_Ω, we have that:

$$\phi^*(\omega,\mathbf{x}) \quad = \quad P(\omega)\phi^*_\omega(\mathbf{x});$$

where ϕ^*_ω are the state visitation frequencies of the optimal policy in the MDP Π_ω for world ω. That is, the optimal visitation frequency for a particular world and state in this world factorizes into the probability of selecting this world defined by P_Ω times the optimal visitation frequencies for this world. In matrix notation, we have that

$$(\phi^*)^\intercal = (P_\Omega)^\intercal (I-\gamma P_{\pi^*})^{-1};$$

and thus, returning to the second term on the righthand side of Equation (A.28), we have that:

$$\begin{aligned}
&2\,(P_\Omega)^\intercal\,(I-\gamma P_{\pi^*})^{-1}(\widetilde{\mathcal{V}}-\mathcal{T}_{\pi^*}\widetilde{\mathcal{V}})^- \\
&= 2(\phi^*)^\intercal(\widetilde{\mathcal{V}}-\mathcal{T}_{\pi^*}\widetilde{\mathcal{V}})^-; \\
&= 2\sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)\phi^*_\omega(\mathbf{x})\left[(\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})-\widetilde{\mathcal{V}}_\omega(\mathbf{x})\right]\mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})<(\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right); \\
&\leq 2\sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)\phi^*_\omega(\mathbf{x})\left\|\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega-\widetilde{\mathcal{V}}_\omega\right\|_\infty\mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})<(\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right). \\
&\leq 2\sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)\phi^*_\omega(\mathbf{x})\left[\|R^\omega_{\pi^*}\|_\infty+\left\|\gamma P^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega\right\|_\infty+\left\|\widetilde{\mathcal{V}}_\omega\right\|_\infty\right]\mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})<(\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right). \\
&\leq 2\sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega} P(\omega)\phi^*_\omega(\mathbf{x})\left[\sharp[\omega]R^o_{max}+\frac{\gamma\sharp[\omega]R^o_{max}}{1-\gamma}+\frac{\sharp[\omega]R^o_{max}}{1-\gamma}\right]\mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})<(\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right). \\
&= 2\sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega}\frac{2\sharp[\omega]R^o_{max}}{1-\gamma}\,P(\omega)\phi^*_\omega(\mathbf{x})\,\mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x})<(\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right);
\end{aligned} \tag{A.30}$$

where the first inequality replaces the difference between the one-step lookahead value $\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega$ and the value function $\widetilde{\mathcal{V}}_\omega$ with the maximum difference; the second inequality simply relies on the triangle inequality for each term in $\left\|\mathcal{T}^\omega_{\pi^*}\widetilde{\mathcal{V}}_\omega-\widetilde{\mathcal{V}}_\omega\right\|_\infty$; and the bound on each max-norm term in the third inequality uses Assumption 13.3.1, where $\sharp[\omega]$ is the number of objects in world ω.

By moving out the constant term $\frac{4R^o_{max}}{1-\gamma}$, the summation term in Equation (A.30) becomes:

$$\sum_{\omega\in\Omega,\mathbf{x}\in\mathbf{X}_\omega}\sharp[\omega]P(\omega)\phi^*_\omega(\mathbf{x})\mathbb{1}\left(\widetilde{\mathcal{V}}(\mathbf{x})<(\mathcal{T}_{\pi^*}\widetilde{\mathcal{V}})(\mathbf{x})\right) \tag{A.31}$$

As described in Section 13.2.2, we can decompose the probability of a world into:

$$P(\omega)=P(\sharp)P(\omega_\sharp\mid\sharp).$$

Substituting this formulation into Equation (A.31), we obtain:

$$\sum_{i=1}^{\infty} \sum_{\omega \in \Omega_i} \sum_{\mathbf{x} \in \mathbf{X}_\omega} i\, P(\sharp = i) P(\omega \mid \sharp = i) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right),$$

or equivalently, by separating the summation into one term for worlds with up to n objects, and another term for larger worlds, we obtain:

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} \sum_{\omega \in \Omega_i} \sum_{\mathbf{x} \in \mathbf{X}_\omega} i\, P(\sharp = i) P(\omega \mid \sharp = i) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right) + \\ &\qquad \textstyle\sum_{j=n+1}^{\infty} \sum_{\omega \in \Omega_j} \sum_{\mathbf{x} \in \mathbf{X}_\omega} j\, P(\sharp = j) P(\omega \mid \sharp = j) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right). \end{aligned} \tag{A.32}$$

Let us start by considering the first term in Equation (A.32). We can apply Theorem A.5.2 to bound this term with high probability. In particular, we choose our sampling distribution $\psi_{\leq n}$ to be

$$\begin{aligned} \psi_{\leq n}(\omega, \mathbf{x}) &= (1-\gamma) P_{\leq n}(\omega) \phi^*_\omega(\mathbf{x}); \\ &= \frac{1-\gamma}{\sum_{\omega' \in \Omega_{\leq n}} P(\omega')} P(\omega) \phi^*_\omega(\mathbf{x}). \end{aligned}$$

We thus have that:

$$\begin{aligned} &\sum_{i=1}^{n} \sum_{\omega \in \Omega_i} \sum_{\mathbf{x} \in \mathbf{X}_\omega} i\, P(\sharp = i) P(\omega \mid \sharp = i) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right) \\ &\quad \leq n \sum_{i=1}^{n} \sum_{\omega \in \Omega_i} \sum_{\mathbf{x} \in \mathbf{X}_\omega} P(\sharp = i) P(\omega \mid \sharp = i) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right) \\ &\quad = n \sum_{\omega \in \Omega_{\leq n}} \sum_{\mathbf{x} \in \mathbf{X}_\omega} P(\omega) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right); \\ &\quad = \frac{\sum_{\omega' \in \Omega_{\leq n}} P(\omega')}{1-\gamma}\, n \sum_{\omega \in \Omega_{\leq n}} \sum_{\mathbf{x} \in \mathbf{X}_\omega} \psi_{\leq n}(\omega, \mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^\omega_{\pi^*} \widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right); \\ &\quad \leq \sum_{\omega' \in \Omega_{\leq n}} P(\omega')\, \varepsilon n, \end{aligned} \tag{A.33}$$

where the first inequality simply substitutes i with n; and the second inequality uses Theorem A.5.2, recasting ε as $\varepsilon(1-\gamma)$. By using the bound in Equation (A.24), we obtain the bound:

$$\sum_{\omega' \in \Omega_{\le n}} P(\omega')\, \varepsilon n \;\; \le \;\; \frac{\kappa_\sharp}{\lambda_\sharp}\left[1 - e^{-\lambda_\sharp n}\right] \varepsilon n, \tag{A.34}$$

with probability at least $1 - \frac{\delta}{2}$, if the number of sampled worlds is:

$$m \ge \frac{4}{\varepsilon(1-\gamma)}\left(k \ln \frac{12}{\varepsilon(1-\gamma)} + \ln \frac{4}{\delta}\right).$$

Note that our algorithm does not sample directly from this distribution $\psi_{\le n}$. However, we do sample worlds from $P_{\le n}(\omega)$, and then represent the constraints for all states and actions in this world, in closed form, by using our factored LP decomposition technique. Thus, each one of our sampled worlds corresponds to an i.i.d. sample from $\psi_{\le n}$ in our constraint set (plus many additional constraints), thus guaranteing the condition of the Theorem A.5.2.

Now, let us consider the second term in Equation (A.32).

$$\begin{aligned}
&\sum_{j=n+1}^{\infty} \sum_{\omega \in \Omega_i} \sum_{\mathbf{x} \in \mathbf{X}_\omega} j\, P(\sharp = j) P(\omega \mid \sharp = j) \phi^*_\omega(\mathbf{x}) \mathbb{1}\left(\widetilde{\mathcal{V}}_\omega(\mathbf{x}) < (\mathcal{T}^{\omega}_{\pi^*}\widetilde{\mathcal{V}}_\omega)(\mathbf{x})\right) \\
&\qquad \le \sum_{j=n+1}^{\infty} \sum_{\omega \in \Omega_i} \sum_{\mathbf{x} \in \mathbf{X}_\omega} j\, P(\sharp = j) P(\omega \mid \sharp = j) \phi^*_\omega(\mathbf{x}); \\
&\qquad = \sum_{j=n+1}^{\infty} j\, P(\sharp = j).
\end{aligned} \tag{A.35}$$

By using Equation (A.26), we bound this term by:

$$\sum_{j=n+1}^{\infty} j\, P(\sharp = j) \;\; \le \;\; \frac{\kappa_\sharp}{\lambda_\sharp}\left[n + \frac{1}{\lambda_\sharp}\right] e^{-\lambda_\sharp n}. \tag{A.36}$$

We can finally obtain a bound for the terms in Equation (A.28):

$$\begin{aligned}
\left\|\widetilde{\mathcal{V}}-\mathcal{V}^*\right\|_{1,P_\Omega} &\leq (P_\Omega)^\intercal\,(\widetilde{\mathcal{V}}-\mathcal{V}^*)+2\,(P_\Omega)^\intercal\,(I-\gamma P_{\pi^*})^{-1}(\widetilde{\mathcal{V}}-\mathcal{T}_{\pi^*}\widetilde{\mathcal{V}})^- \\
&\leq \left\|\widehat{\mathcal{V}}-\mathcal{V}^*\right\|_{1,P_\Omega}+\frac{2R^o_{max}}{1-\gamma}\frac{\kappa_\sharp}{\lambda_\sharp}\left[\varepsilon n+\left(n(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n}\right]+ \\
&\quad \frac{4R^o_{max}}{1-\gamma}\left[\frac{\kappa_\sharp}{\lambda_\sharp}\left[1-e^{-\lambda_\sharp n}\right]\varepsilon n+\frac{\kappa_\sharp}{\lambda_\sharp}\left[n+\frac{1}{\lambda_\sharp}\right]e^{-\lambda_\sharp n}\right], \qquad \text{(A.37)}
\end{aligned}$$

where we substituted the intermediate results in Equations (A.29), (A.30), (A.34), and (A.36), into Equation (A.28). The proof is concluded by rearranging the terms in Equation (A.37). ▮

The simplified version presented in Theorem 13.3.2 is obtained by choosing:

$$n=\left\lfloor\frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp}\right\rfloor,$$

yielding:

$$\begin{aligned}
\varepsilon n+\left(n(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp n} &\leq \varepsilon\frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp}+\left(\frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp}(1-\varepsilon)+\frac{1}{\lambda_\sharp}\right)e^{-\lambda_\sharp\frac{\ln\left(\frac{1}{\varepsilon}\right)}{\lambda_\sharp}}; \\
&= \frac{1}{\lambda_\sharp}\left[\varepsilon\ln\left(\frac{1}{\varepsilon}\right)+\varepsilon\left(\ln\left(\frac{1}{\varepsilon}\right)(1-\varepsilon)+1\right)\right]; \\
&= \frac{\varepsilon}{\lambda_\sharp}\left[\ln\left(\frac{1}{\varepsilon}\right)(2-\varepsilon)+1\right]. \\
&\leq \frac{3\varepsilon}{\lambda_\sharp}\ln\left(\frac{1}{\varepsilon}\right).
\end{aligned}$$

▮

Bibliography

Albus, J. (1975). A new approach to manipular control: The cerebellar model articulation controller (CMAC). *Dynamoc Systems, Measurement and Control, 97,* 220–227.

Allen, D., & Darwiche, A. (2003). New advances in inference by recursive conditioning. *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-03)*. Acapulco, Mexico: Morgan Kaufmann.

Allender, E., Arora, S., Kearns, M., Moore, C., & Russell, A. (2002). A note on the representational incompatability of function approximation and factored dynamics. *15th Neural Information Processing Systems (NIPS-15)*. Vancouver, Canada.

Andre, D., & Russell, S. (2002). State abstraction for programmable reinforcement learning agents. *The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*. Edmonton, Canada.

Arnborg, S., Corneil, D. G., & Proskurowski, A. (1987). Complexity of finding embeddings in a K-tree. *SIAM Journal of Algebraic and Discrete Methods, 8*(2), 277 – 284.

Atkeson, C., & Santamaria, J. (1997). A comparison of direct and model-based reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA-97)*.

Baxter, J., & Bartlett, P. (2000). Reinforcement learning in POMDP's via direct gradient ascent. *Proc. 17th International Conf. on Machine Learning* (pp. 41–48). Morgan Kaufmann, San Francisco, CA.

Becker, A., & Geiger, D. (2001). A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence, 125*(1-2), 3–17.

Bellman, R., Kalaba, R., & Kotkin, B. (1963). Polynomial approximation – a new computational technique in dynamic programming. *Math. Comp.*, *17*(8), 155–161.

Bellman, R. E. (1957). *Dynamic programming*. Princeton, New Jersey: Princeton University Press.

Bernstein, D., Zilberstein, S., & Immerman, N. (2000). The complexity of decentralized control of Markov decision processes. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*. Stanford, California: Morgan Kaufmann.

Bertele, U., & Brioschi, F. (1972). *Nonserial dynamic programming*. New York: Academic Press.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont, Massachusetts: Athena Scientific.

Bertsimas, D., & Tsitsiklis, J. (1997). *Introduction to linear optimization*. Belmont, Massachusetts: Athena Scientific.

Bidyuk, B., & Dechter, R. (2003). An empirical study of w-cutset sampling for bayesian networks. *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-03)*. Acapulco, Mexico: Morgan Kaufmann.

Blum, A., & Langford, J. (1999). Probabilistic planning in the graphplan framework. *5th European Conference on Planning (ECP'99)* (pp. 319–332).

Blum, B., Shelton, C., & Koller, D. (2003). A continuation method for nash equilibria in structured games. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (pp. 757 – 764). Acapulco, Mexico: Morgan Kaufmann.

Bolch, G., Greiner, S., de Meer, H., & Trivedi, K. (1998). *Queueing networks and markov chains*. Wiley.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. *Theoretical Aspects of Rationality and Knowledge* (pp. 195–201).

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1 – 94.

Boutilier, C., & Dearden, R. (1996). Approximating value trees in structured dynamic programming. *Proc. ICML* (pp. 54–62).

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. *Proc. IJCAI* (pp. 1104–1111).

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*(1-2), 49–107.

Boutilier, C., & Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)* (pp. 1168–1175). Portland, Oregon: AAAI Press.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 690 – 697). Seattle, Washington: Morgan Kaufmann.

Boyan, J., & Littman, M. (1993). Packet routing in dynamically changing networks: A reinforcement learning approach. *The Sixth Conference on Advances in Neural Information Processing Systems* (pp. 671–678). San Francisco, California: Morgan Kaufmann.

Boyen, X., & Koller, D. (1999). Exploiting the architecture of dynamic systems. *National Conference on Artificial Intelligence AAAI-99*.

Brafman, R., & Tennenholtz, M. (2001). R-max - A general polynomial time algorithm for near-optimal reinforcement learning. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*. Seattle, Washington: Morgan Kaufmann.

Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and regression trees*. Chapman and Hall.

Casella, G., & Robert, C. P. (1996). Rao-Blackwellisation of sampling schemes. *Biometrika*, *83*(1), 81–94.

Cassandra, A. R., Littman, M. L., & Zhang, N. L. (1997). Incremental prunning: A simple, fast, exact method for partially observable markov decision processes. *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference* (pp. 54–61). Providence, Rhode Island: Morgan Kaufmann.

Cheney, E. W. (1982). *Approximation theory*. New York, NY: Chelsea Publishing Co. 2nd edition.

Chow, C., & Tsitsiklis, J. (1991). An optimal one-way multigrid algorithm for discrete time stochastic control. *IEEE Transactions on Automatic Control*, *36*, 898 – 914.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* (pp. 746–752). Wisconsin: AAAI Press.

Cowell, R., Dawid, P., Lauritzen, S., & Spiegelhalter, D. (1999). *Probabilistic networks and expert systems*. New York: Spinger.

Crespo, A., & Garcia-Molina, H. (2002). Routing indices for peer-to-peer systems. *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*.

Crites, R., & Barto, A. (1996). Improving elevator performance using reinforcement learning. *Eigth Conference on the Advances in Neural Information Processing Systems* (pp. 1017–1023). Denver, Colorado: MIT Press.

de Farias, D., & Van Roy, B. (2001a). The linear programming approach to approximate dynamic programming. *Submitted to Operations Research*.

de Farias, D., & Van Roy, B. (2001b). On constraint sampling for the linear programming approach to approximate dynamic programming. *To appear in Mathematics of Operations Research*.

De Raedt (ed.), L. (1995). *Advances in inductive logic programming*. IOS Press.

Dean, T., & Givan, R. (1997). Model minimization in Markov decision processes. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)* (pp. 106–111). Providence, Rhode Island, Oregon: AAAI Press.

Dean, T., Kaelbling, L. P., Kirman, J., & Nicholson, A. (1993). Planning with deadlines in stochastic domains. *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)* (pp. 574–579). Washington, D.C.: AAAI Press.

Dean, T., & Kanazawa, K. (1988). Probabilistic temporal reasoning. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)* (pp. 524–528). St. Paul, Minnesota.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(3), 142–150.

Dean, T., & Lin, S. (1995). Decomposition techniques for planning in stochastic domains. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montreal, Canada: Morgan Kaufmann.

Dearden, R., & Boutilier, C. (1997). Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, *89*(1), 219–283.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1–2), 41–85.

Derman, C. (1970). *Finite state Markovian decision processes*. Academic Press.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Doucet, A., de Freitas, N., Murphy, K., & Russell, S. (2000). Rao-Blackwellised particle filtering for dynamic bayesian networks. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)* (pp. 176 – 183). Stanford, California: Morgan Kaufmann.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*(1/2), 7–52.

Estrin, D., Govindan, R., Heidemann, J., & Kumar, S. (1999). Next century challenges: Scalable coordination in sensor networks. *Mobile Computing and Networking* (pp. 263–270).

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*(4), 251–288.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3–4), 189–208.

Freecraft (2003). Open source strategic computer game. http://www.freecraft.org, http://www.nongnu.org/stratagus/.

Friedman, N., & Singer, Y. (2000). Efficient bayesian parameter estimation in large discrete domains. *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*. MIT Press.

Geiger, D., & Heckerman, D. (1996). Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence*, *82*(1-2), 45–74.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *6*(6), 721–741.

Gordon, G. (1995). Stable function approximation in dynamic programming. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 261–268). Tahoe City, CA: Morgan Kaufmann.

Gordon, G. (1999). *Approximate solutions to Markov decision processes*. Doctoral dissertation, Carnegie Mellon University.

Gordon, G. (2001). Reinforcement learning with function approximation converges to a region. *Advances in Neural Information Processing Systems 13* (pp. 1040–1046). MIT Press.

Guestrin, C. E., & Gordon, G. (2002). Distributed planning in hierarchical factored MDPs. *The Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-2002)* (pp. 197–206). Edmonton, Canada.

Guestrin, C. E., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments with relational MDPs. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (pp. 1003 – 1010). Acapulco, Mexico: Morgan Kaufmann.

Guestrin, C. E., Koller, D., & Parr, R. (2001a). Max-norm projections for factored MDPs. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 673 – 680). Seattle, Washington: Morgan Kaufmann.

Guestrin, C. E., Koller, D., & Parr, R. (2001b). Multiagent planning with factored MDPs. *14th Neural Information Processing Systems (NIPS-14)* (pp. 1523–1530). Vancouver, Canada.

Guestrin, C. E., Koller, D., & Parr, R. (2001c). Solving factored POMDPs with linear value functions. *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01) workshop on Planning under Uncertainty and Incomplete Information* (pp. 67 – 75). Seattle, Washington.

Guestrin, C. E., Koller, D., Parr, R., & Venkataraman, S. (2002a). Efficient solution algorithms for factored MDPs. *Accepted in Journal of Artificial Intelligence Research (JAIR)*.

Guestrin, C. E., Lagoudakis, M., & Parr, R. (2002b). Coordinated reinforcement learning. *The Nineteenth International Conference on Machine Learning (ICML-2002)* (pp. 227–234). Sydney, Australia.

Guestrin, C. E., Patrascu, R., & Schuurmans, D. (2002c). Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. *The Nineteenth International Conference on Machine Learning (ICML-2002)* (pp. 235–242). Sydney, Australia.

Guestrin, C. E., Venkataraman, S., & Koller, D. (2002d). Context specific multiagent coordination and planning with factored MDPs. *The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)* (pp. 253–259). Edmonton, Canada.

Hansen, E. (1998). *Finite-memory control of partially observable systems*. Doctoral dissertation, University of Massachusetts Amherst, Amherst, Massachusetts.

Hauskrecht, M., Meuleau, N., Kaelbling, L., Dean, T., & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)* (pp. 220–229). Madison, Wisconsin: Morgan Kaufmann.

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)* (pp. 279–288). Stockholm, Sweden: Morgan Kaufmann.

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (2002). Stochastic planning using decision diagrams – C implementation. http://www.cs.ubc.ca/spider/staubin/Spudd/.

Horvitz, E., Suermondt, H., & Cooper, G. (1989). Bounded conditioning: Flexible inference for decisions under scarce resources. *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)* (pp. 182–193). Windsor, Ontario: Morgan Kaufmann.

Howard, R. A. (1960). *Dynamic programming and Markov processes*. Cambridge, Massachusetts: MIT Press.

Howard, R. A., & Matheson, J. E. (1984). Influence diagrams. In R. A. Howard and J. E. Matheson (Eds.), *Readings on the principles and applications of decision analysis*, 721–762. Menlo Park, California: Strategic Decisions Group.

Isidori, A. (1989). *Nonlinear control systems: An introduction*. Springer.

Jaakkola, T., Singh, S., & Jordan, M. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. *Advances in Neural Information Processing Systems 7* (pp. 345–352). Cambridge, Massachusetts: MIT Press.

Jensen, F., Jensen, F. V., & Dittmer, S. (1994). From influence diagrams to junction trees. *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference* (pp. 367–373). Seattle, Washington: Morgan Kaufmann.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*. Morgan Kaufmann.

Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. *The Fifteenth International Conference on Machine Learning*. Madison, Wisconsin: Morgan Kaufmann.

Keeney, R. L., & Raiffa, H. (1976). *Decisions with multiple objectives: Preferences and value tradeoffs*. New York: Wiley.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence, 113*, 125–148.

Kim, K.-E., & Dean, T. (2001). Solving factored Mdps using non-homogeneous partitioning. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 683 – 689). Seattle, Washington: Morgan Kaufmann.

Kjaerulff, U. (1990). *Triangulation of graphs – algorithms giving small total state space* (Technical Report TR R 90-09). Department of Mathematics and Computer Science, Strandvejen, Aalborg, Denmark.

Kok, J. R., Spaan, M. T. J., & Vlassis, N. (2003). Multi-robot decision making using coordination graphs. *Proc. 11th Int. Conf. on Advanced Robotics*. Coimbra, Portugal.

Koller, D., & Milch, B. (2001). Multi-agent influence diagrams for representing and solving games. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 1027–1036). Seattle, Washington: Morgan Kaufmann.

Koller, D., & Parr, R. (1999). Computing factored value functions for policies in structured MDPs. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)* (pp. 1332 – 1339). Morgan Kaufmann.

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)* (pp. 326 – 334). Stanford, California: Morgan Kaufmann.

Koller, D., & Pfeffer, A. (1997). Object-oriented Bayesian networks. *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference* (pp. 302–313). Providence, Rhode Island: Morgan Kaufmann.

Koller, D., & Pfeffer, A. (1998). Probabilistic frame-based systems. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. Wisconsin: AAAI Press.

Konda, V., & Tsitsiklis, J. (2000). Actor-critic algorithms. *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*. MIT Press.

Kozlov, A., & Koller, D. (1997). Nonuniform dynamic discretization in hybrid networks. *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference* (pp. 314–325). Providence, Rhode Island: Morgan Kaufmann.

Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, *76*(1-2), 239–286.

Kushner, H. J., & Chen, C. H. (1974). Decomposition of systems governed by Markov chains. *IEEE Transactions on Automatic Control*, *19*(5), 501–507.

La Mura, P. (1999). *Foundations of multi-agent systems*. Doctoral dissertation, Graduate School of Business, Stanford University.

Lagoudakis, M., & Parr, R. (2001). Model free least squares policy iteration. *14th Neural Information Processing Systems (NIPS-14)*. Vancouver, Canada.

Lagoudakis, M., & Parr, R. (2002). Learning in zero-sum team markov games using factored value functions. *15th Neural Information Processing Systems (NIPS-15)*. Vancouver, Canada.

Laird, J., & Van Lent, M. (2001). Human-level AI's killer application: Interactive computer games. *Artificial Intelligence Magazine*, *22*(2), 15 – 26.

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B 50*(2), 157–224.

Leyton-Brown, K., & Tennenholtz, M. (2003). Local-effect games. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (pp. 772 – 777). Acapulco, Mexico: Morgan Kaufmann.

Liberatore, P. (2002). The size of mdp factored policies. *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002)* (pp. 267–272).

Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the 11th International Conference on Machine Learning* (pp. 157–163). New Brunswick, NJ: Morgan Kaufmann.

Littman, M., Kearns, M., & Singh, S. (2002). An efficient, exact algorithm for solving tree-structured graphical games. *Advances in Neural Information Processing Systems 14*. MIT Press.

Madani, O., Condon, A., & Hanks, S. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision process problems. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*. Florida: AAAI Press.

Manne, A. S. (1960). Linear programming and sequential decisions. *Management Science*, *6*(3), 259–267.

Martin, M., & Geffner, H. (2000). Learning generalized policies in planning using concept languages. *Proc. 7th Int. Conf. on Knowledge Representation and Reasoning (KR 2000)*. Colorado: Morgan Kaufmann.

Meuleau, N., Hauskrecht, M., Peshkin, L., Kaelbling, L., Dean, T., & Boutilier, C. (1998). Solving very large weakly-coupled Markov decision processes. *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 165–172). Madison, WI.

Meuleau, N., Peshkin, L., & Kim, K. (2001). *Exploration in gradient-based reinforcement learning*AI Memo 1713). MIT.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping—reinforcement learning with less data and less time. *Machine Learning*, *13*, 103–130.

Mundhenk, M., Goldsmith, J., Lusena, C., & Allender, E. (2000). Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, *47*(4), 681–720.

Ng, A., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*. Stanford, California: Morgan Kaufmann.

Parr, R. (1998). Flexible decomposition algorithms for weakly coupled markov decision problems. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*. Madison, Wisconsin: Morgan Kaufmann.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *10th Neural Information Processing Systems (NIPS-10)*. MIT Press.

Patrascu, R., Schuurmans, D., Poupart, P., Boutilier, C., & Guestrin, C. (2002). Greedy linear value-approximation for factored Markov decision processes. *The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*. Edmonton, Canada.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, *32*, 247–257.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, California: Morgan Kaufmann.

Peshkin, L., Meuleau, N., Kim, K., & Kaelbling, L. (2000). Learning to cooperate via policy search. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)* (pp. 307–314). Stanford, California: Morgan Kaufmann.

Pfeffer, A. J. (2000). *Probabilistic reasoning for complex systems*. Doctoral dissertation, Stanford University.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: an anytime algorithm for pomdps. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*. Acapulco, Mexico: Morgan Kaufmann.

Pollard, D. (1984). *Convergence of stochastic processes*. Springer-Verlag.

Poole, D. (2003). First-order probabilistic inference. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (pp. 985 – 991). Acapulco, Mexico: Morgan Kaufmann.

Poupart, P., & Boutilier, C. (2002). Value-directed compression of POMDPs. *15th Neural Information Processing Systems (NIPS-15)*. Vancouver, Canada.

Poupart, P., Boutilier, C., Patrascu, R., & Schuurmans, D. (2002). Piecewise linear value function approximation for factored mdps. *The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*. Edmonton, Canada.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Reed, B. (1992). Finding approximate separators and computing tree-width quickly. *24th Annual Symposium on Theory of Computing* (pp. 221–228). ACM.

Roy, N., & Gordon, G. (2002). Exponential family pca for belief compression in POMDPs. *15th Neural Information Processing Systems (NIPS-15)*. Vancouver, Canada.

Roy, N., & Thrun, S. (2000). Coastal navigation with mobile robots. *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference* (pp. 1043 – 1049). Denver, Colorado: MIT Press.

Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, *65*(3), 487 – 516.

Sallans, B., & Hinton, G. E. (2001). Using free energies to represent q-values in a multi-agent reinforcement learning task. *Advances in Neural Information Processing Systems 13* (pp. 1075–1081).

Schneider, J., Wong, W., Moore, A., & Riedmiller, M. (1999). Distributed value functions. *The Sixteenth International Conference on Machine Learning* (pp. 371–378). Bled, Slovenia: Morgan Kaufmann.

Schuurmans, D., & Patrascu, R. (2001). Direct value-approximation for factored MDPs. *Advances in Neural Information Processing Systems (NIPS-14)* (pp. 1579–1586). Vancouver, Canada.

Schweitzer, P., & Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications, 110*, 568 – 582.

Shapley, L. (1953). Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America, 39*, 1095–1100.

Sharma, R., & Poole, D. (2003). Efficient inference in large discrete domains. *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-03)*. Acapulco, Mexico: Morgan Kaufmann.

Shelton, C. R. (2001). Policy improvement for POMDPs using normalized importance sampling. *Proceedings of the Seventeenth International Conference on Uncertainty in Artificial Intelligence* (pp. 496–503).

Simon, H. A. (1981). *The sciences of the artificial*. Cambridge, Massachusetts: MIT Press. second edition.

Singh, S., & Cohn, D. (1998). How to dynamically merge Markov decision processes. *Advances in Neural Information Processing Systems*. The MIT Press.

Sondik, E. J. (1971). *The optimal control of partially observable Markov decision processes*. Doctoral dissertation, Stanford University, Stanford, California.

St-Aubin, R., Hoey, J., & Boutilier, C. (2001). APRICODD: Approximate policy construction using decision diagrams. *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference* (pp. 1089–1095). Denver, Colorado: MIT Press.

Stiefel, E. (1960). Note on Jordan elimination, linear programming and Tchebycheff approximation. *Numerische Mathematik*, *2*, 1 – 17.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*. Denver, Colorado: MIT Press.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Tadepalli, P., & Ok, D. (1996). Scaling up average reward reinforcmeent learning by approximating the domain models and the value function. *Proceedings of the Thirteenth International Conference on Machine Learning*. Bari, Italy: Morgan Kaufmann.

Tatman, J. A., & Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, *20*(2), 365–379.

Thrun, S., & O'Sullivan, J. (1996). Discovering structure in multiple learning tasks: The TC algorithm. *ICML-96*.

Tsitsiklis, J. N., & Van Roy, B. (1996a). *An analysis of temporal-difference learning with function approximation*Technical Report LIDS-P-2322). Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

Tsitsiklis, J. N., & Van Roy, B. (1996b). Feature-based methods for large scale dynamic programming. *Machine Learning*, *22*, 59–94.

Van Roy, B. (1998). *Learning and value function approximation in complex decision processes*. Doctoral dissertation, Massachusetts Institute of Technology.

Watkins, C. J. (1989). *Models of delayed reinforcement learning*. Doctoral dissertation, Psychology Department, Cambridge University, Cambridge, United Kingdom.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3), 279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256.

Williams, R. J., & Baird, L. C. I. (1993). *Tight performance bounds on greedy policies based on imperfect value functions* (Technical Report). College of Computer Science, Northeastern University, Boston, Massachusetts.

Wolpert, D., Wheller, K., & Tumer, K. (1999). General principles of learning-based multi-agent systems. *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)* (pp. 77–83). Seattle, WA, USA: ACM Press.

Yannakakis, M. (1991). Expressing combinatorial optimization problems by linear programming. *Journal of Computer and System Sciences*, *43*, 441–466.

Yedidia, J., Freeman, W., & Weiss, Y. (2001). Generalized belief propagation. *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference* (pp. 689–695). Denver, Colorado: MIT Press.

Yoon, S. W., Fern, A., & Givan, B. (2002). Inductive policy selection for first-order MDPs. *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-02)*. Edmonton, Canada: Morgan Kaufmann.

Yost, K. (1998). *Solving large-scale allocation problems with partially observable outcomes*. Doctoral dissertation, Naval Postgraduate School.

Zhang, N., & Poole, D. (1999). On the role of context-specific independence in probabilistic reasoning. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)* (pp. 1288–1293). Morgan Kaufmann.

Zhang, T. (2002). Covering number bounds of certain regularized linear function classes. *Journal of Machine Learning Research, 2*, 527–550.

Index