INCREASING SCALABILITY IN ALGORITHMS FOR CENTRALIZED AND DECENTRALIZED PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES: EFFICIENT DECISION-MAKING AND COORDINATION IN UNCERTAIN ENVIRONMENTS

A Dissertation Presented

by

CHRISTOPHER AMATO

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

DOCTOR OF PHILOSOPHY

May 2010

Department of Computer Science

INCREASING SCALABILITY IN ALGORITHMS FOR CENTRALIZED AND DECENTRALIZED PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES: EFFICIENT DECISION-MAKING AND COORDINATION IN UNCERTAIN ENVIRONMENTS

A Dissertation Presented

by

CHRISTOPHER AMATO

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Shlomo Zilberstein. His insight into conducting and presenting research as well as finding interesting directions for study has been invaluable. Equally helpful was his faith in letting me choose research questions that I found interesting and giving me the confidence to solve them.

I would also like to thank the other members of my committee. Victor Lesser's positive energy, vast knowledge of research and innovative view of the future has helped me become a more well-rounded researcher. My conversations with Sridhar Mahadevan have improved the rigor of my work and pointed me in the direction of even further scalability. Iqbal Agha's knowledge of optimization and insightful questions aided this thesis a great deal.

My frequent conversations with other members of the Resource-Bounded Reasoning lab and other students at UMass have also been very valuable. In particular, I would like to thank Martin Allen, Dan Bernstein, Alan Carlin, Marek Petrik, Akshat Kumar, Sven Seuken, Siddharth Srivastava, Raphen Becker, Jiaying Shen and Hala Mostafa for the many discussions ranging from the correctness of a proof and the validity of new fields of study to the value of the Red Sox rotation and the never ending Mac vs. PC debate.

I would like to thank the administrative staff for allowing my life in graduate school to function smoothly, especially Leeanne Leclerc and Michele Roberts.

This thesis would never have been completed without the support of my wife, Sarah Mary Amato. She put up with many late nights of me working on my laptop and many weekends where I would disappear all day to the local coffee shop to get work done (thanks Haymarket Cafe!). I also appreciated her proofreading various papers and listening to many presentations, I could not have hoped for a more attentive and positive audience. The cats were not particularly helpful, but deserve to be thanked nonetheless. When they weren't fighting, one was likely to be sitting on my laptop or on a stack of papers, but I appreciated having Betelgeuse and Schwag around.

Lastly, but certainly not least, I would like to thank my parents. They are the ones that encouraged me to chase my dreams and seek a career that I truly love. They also gave me the gift of curiosity and put up with my incessant question asking. I am forever in their debt for the support they have given me over the years.

ABSTRACT

INCREASING SCALABILITY IN ALGORITHMS FOR CENTRALIZED AND DECENTRALIZED PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES: EFFICIENT DECISION-MAKING AND COORDINATION IN UNCERTAIN ENVIRONMENTS

MAY 2010

CHRISTOPHER AMATO

B.A., TUFTS UNIVERSITY

B.S., UNIVERSITY OF MASSACHUSETTS AMHERST

M.S., UNIVERSITY MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

As agents are built for ever more complex environments, methods that consider the uncertainty in the system have strong advantages. An example of such an environment is robot navigation in which agents may have sensors and actuators that are imperfect, leading to partial information and stochastic actions. This uncertainty is also common in domains such as medical diagnosis and treatment, inventory management, sensor networks and e-commerce. Developing effective frameworks for reasoning under uncertainty is a thriving research area in artificial intelligence. When a single decision maker is present, the partially observable Markov decision process

(POMDP) model is a popular and powerful choice. When choices are made in a decentralized manner by a set of decision makers, the problem can be modeled as a decentralized partially observable Markov decision process (DEC-POMDP). While POMDPs and DEC-POMDPs offer rich frameworks for sequential decision making under uncertainty, the computational complexity of each model presents an important research challenge.

As a way to address this high complexity, we have developed several solution methods based on utilizing domain structure, memory-bounded representations and sampling. These approaches address some of the major bottlenecks for decision-making in real-world uncertain systems. The methods include a more efficient optimal algorithm for DEC-POMDPs as well as scalable approximate algorithms for POMDPs and DEC-POMDPs. The optimal approach, incremental policy generation, uses reachability in the domain structure to more efficiently generate solutions for a given problem by focusing the search on only decisions that may contribute to the optimal solution. This approach can also be used in any other optimal or approximate dynamic programming algorithm for DEC-POMDPs, improving its scalability. We also show that approximate dynamic programming can be improved by sampling to determine areas that are likely to be visited by other agent policies.

Other approximate algorithms we present include a memory-bounded approach that represents optimal fixed-size solutions for POMDPs and DEC-POMDPs as nonlinear programs. We describe how this representation can be solved optimally as well as approximately. We also show that increased solution quality and scalability can also be achieved by using the more compact and structured representation of Mealy machines. Mealy machines can be used in any infinite-horizon approach that uses finite-state controllers as a solution representation, allowing memory to be utilized more efficiently and domain information to be automatically integrated into the solution representation. We can also make use of domain structure in the form of goals.

This work can provide an optimal solution for DEC-POMDPs under the assumptions that designated actions cause the problem to terminate and negative rewards are given for all other non-terminal actions. If these assumptions are weakened, our sample-based algorithm can be used. We have derived bounds for this algorithm to determine the number of samples that are needed to ensure optimality is approached. Lastly, we have merged a memory-bounded representation with sampling to learn domain structure and produce concise, high quality results.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . **iv**

ABSTRACT . . . . **vi**

LIST OF TABLES . . . . **xiv**

LIST OF FIGURES . . . . **xvii**

CHAPTER

1. INTRODUCTION **1**

1.1 Overview 1
1.2 Models of decision making 2
1.3 Overview of relevant work 3

1.3.1 Relevant fields 3
1.3.2 Solution methods 6

1.4 Summary of contributions 8
1.5 Thesis organization 10

2. SINGLE AGENT DECISION-MAKING UNDER UNCERTAINTY **12**

2.1 Markov decision processes 12
2.2 Partially observable Markov decision processes 13

2.2.1 The POMDP model 13
2.2.2 Solution representations 14

2.3 Optimal POMDP solution methods 16

2.3.1 Important concepts 16
2.3.2 Value iteration 18

2.3.3 Policy Iteration 20

2.4 Approximate solution methods 21

2.4.1 Controller-based methods 21
2.4.2 Point-based methods 23

2.5 Additional models and approaches 24

3. DECENTRALIZED DECISION-MAKING UNDER UNCERTAINTY 26

3.1 Decentralized POMDPs 26

3.1.1 The DEC-POMDP model 26
3.1.2 Solution representations 27

3.2 Optimal DEC-POMDP solution methods 30

3.2.1 The difficulty of extending POMDP methods 30
3.2.2 Generalized belief states 31
3.2.3 Finite-horizon dynamic programming 32
3.2.4 Policy iteration 34
3.2.5 Heuristic search approaches 36

3.3 Approximate solution methods 37

3.3.1 Approximate dynamic programming 37
3.3.2 Controller-based methods 38

3.4 Additional modeling assumptions 39

4. INCREMENTAL POLICY GENERATION: MORE EFFICIENT DYNAMIC PROGRAMMING FOR DEC-POMDPS 41

4.1 Overview 41
4.2 Incremental policy generation 42
4.3 Algorithm 45

4.3.1 Finite-horizon case 45
4.3.2 Infinite-horizon case 48

4.4 Analysis 49

4.4.1 Finite-horizon case 49
4.4.2 Infinite-horizon case 51

4.5 Incorporating Start State Information 54
4.6 Experiments 56

4.6.1 Finite-horizon case 56
4.6.2 Infinite-horizon case 59

4.7 Discussion 61

5. OPTIMIZING FIXED-SIZE CONTROLLERS 63

5.1 Overview 63
5.2 Disadvantages of previous algorithms 65

5.2.1 POMDP approximate methods 66
5.2.2 DEC-POMDP approximate methods 67

5.3 Nonlinear optimization formulations 67

5.3.1 NLP formulation for POMDPs 69
5.3.2 NLP formulation for DEC-POMDPs 71
5.3.3 Scaling up the NLP formulations 73

5.4 Incorporating correlation in the decentralized model 74
5.5 Methods for solving the NLP 76

5.5.1 Finding a globally optimal solution 77
5.5.2 Finding a locally optimal solution 78

5.6 Experiments 79

5.6.1 POMDP results 80
5.6.2 DEC-POMDP results 86

5.7 Discussion 93

6. POLICIES AS MEALY MACHINES 96

6.1 Overview 96
6.2 Automata with output 97
6.3 Finite-state controllers 98

6.3.1 Moore machines 99
6.3.2 Mealy machines 100

6.4 Mealy controllers for (DEC-)POMDPs 102

6.4.1 Optimal POMDP and DEC-POMDP methods 103

6.4.2 Bounded policy iteration 104
6.4.3 Nonlinear programming 106

6.5 Solving the Mealy machine more efficiently 109
6.6 Experiments 111
6.7 Discussion 115

7. APPROXIMATE DYNAMIC PROGRAMMING APPROACHES FOR DEC-POMDPS 118

7.1 Heuristic policy iteration 118

7.1.1 Directed pruning 119
7.1.2 Belief set generation 120
7.1.3 Algorithmic approach 121
7.1.4 Experiments 123

7.2 Incremental policy generation (IPG) with approximate dynamic programming 126

7.2.1 Integrating with approximate dynamic programming 126
7.2.2 Experiments 127

7.3 Approximate infinite-horizon incremental policy generation: Mixing IPG with heuristic policy iteration 129

7.3.1 Algorithmic approach 130
7.3.2 Experiments 132

7.4 Discussion 132

8. ACHIEVING GOALS IN DEC-POMDPS 135

8.1 Overview 135
8.2 Related work 137
8.3 Indefinite-horizon DEC-POMDPs 137

8.3.1 Overview 137
8.3.2 Indefinite-horizon framework and solution 139

8.4 Goal-directed DEC-POMDPs 142

8.4.1 Overview 142
8.4.2 Algorithmic approach 143
8.4.3 Algorithm description 145
8.4.4 Example 147

8.5 Analysis 149
8.6 Experiments 152

8.6.1 Experimental Setup 152
8.6.2 Results 153

8.7 Discussion 155

9. CONCLUSION 157

9.1 Summary of contributions 157
9.2 Future work 159

9.2.1 Theory and Algorithms 160
9.2.2 Applications 162

APPENDICES

A. DEC-POMDP DOMAIN DESCRIPTIONS 164

A.1 Recycling robot problem 164
A.2 Two agent tiger problem 165
A.3 Meeting in a grid problems 165
A.4 Box pushing problem 166
A.5 Rover problems 166

B. AMPL FORMULATIONS FOR NLP 168

B.1 The POMDP formulation 168

B.1.1 NLP model 168
B.1.2 Problem data 169

B.2 The DEC-POMDP formulation 171

B.2.1 NLP model 171
B.2.2 Problem data 172

BIBLIOGRAPHY 173

LIST OF TABLES

Table **Page**

2.1 The linear program used by BPI. Variables x_a and $x_{q',a,o}$ represent $P(a|q)$ and $P(q',a|q,o)$ for a given node, q. 22

3.1 The linear program that determines if agent i tree, q_i is dominated by comparing its value to other trees for that agent, $\hat{q}_i$. The variable $x(q_{-i},s)$ is a distribution over trees of the other agents and system states. 33

3.2 Policy Iteration for DEC-POMDPs. 35

4.1 Size, running time and value produced for each horizon on the test domains. For dynamic programming algorithms the size is given as the number of of policy trees before and after pruning (if different) and only one number is shown if both agent trees are the same size. For top-down approaches the size of the final Bayesian game is provided. 57

4.2 Infinite-horizon results of applying policy iteration and incremental policy generation (IPG). The number of nodes generated is shown in parentheses beside the number retained after pruning and iterations for which pruning could not be completed with the given amount of time and memory are marked with *. 60

5.1 The NLP defining an optimal fixed-size POMDP controller. Variable $x(q',a,q,o)$ represents $P(q',a|q,o)$, variable $z(q,s)$ represents $V(q,s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation. 69

5.2 The NLP defining a set of optimal fixed-size DEC-POMDP controllers. For each agent i, variable $x(q_i,a_i)$ represents $P(a_i|q_i)$, variable $y(q_i,a_i,o_i,q_i')$ represents $P(q_i'|q_i,a_i,o_i)$ and variable $z(\vec{q},s)$ represents $V(\vec{q},s)$ where $\vec{q^0}$ represents the initial controller node for each agent. 71

5.3 The more scalable NLP representation for POMDPs. Variable $x_F(q', a, o)$ represents $P(q', a|q_0, o)$, variable $x_R(q', q, o)$ represents $P(q'|q_0, o)$ for a fixed action, a_q at node q, variable $z(q, s)$ represents $V(q, s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation. 73

5.4 The nonlinear program for including a correlation device in our DEC-POMDP NLP formulation. For each agent i, variable $x(q_i, a_i, c)$ represents $P(a_i|q_i, c)$, variable $y(q_i, a_i, o_i, q_i', c)$ represents $P(q_i'|q_i, a_i, o_i, c)$, variable $z(\vec{q}, s, c)$ represents $V(\vec{q}, s, c)$, $\vec{q^0}$ represents the initial controller node for each agent and $w(c, c')$ represents $P(c'|c)$. The other constraints are similar to those above with the addition of a sum to one constraint for the correlation device. 76

5.5 Values, representation sizes and running times (in seconds) for the set of benchmark problems. Results for other algorithms were taken from the following sources: BPI [86], biased BPI [85], HSVI [104], HSVI2 [105], PERSEUS [108], PBPI [56], PBVI [83], Q_{MDP} [108] 81

5.6 Values, representation sizes and running times (in seconds) for the original Hallway and Hallway2 problems that reset after the goal has been reached. 85

5.7 Box pushing problem values and running times (in seconds) for each controller size using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS. An ‘x’ signifies that the approach was not able to solve the problem and because the fixed action formulation cannot solve 1 node controllers, an ‘n/a’ is given. 92

6.1 Nonlinear program representing an optimal Mealy machine POMDP policy of a given size with variables $x(a, q)$ represent $P(q, a|q_0, o^*)$, $y(q', a, q, o)$ represents $P(q', a|q, o)$ and $z(q, o, s)$ represents $V(q, o, s)$. 107

6.2 Nonlinear program representing an optimal Mealy machine DEC-POMDP policy of a given size with variables $x(\vec{a}, \vec{q})$ representing $P(\vec{q}, \vec{a}|\vec{q}^0, o^*)$, $z(\vec{q}, \vec{o}, s)$ representing $V(\vec{q}, \vec{o}, s)$ and $y(q_i', a_i, q_i, o_i)$ representing $P(q_i', a_i|q_i, o_i)$ for each agent i. 108

6.3 Results for POMDP problems comparing Mealy and Moore machines and other algorithms. The size for the machines is the number of nodes in the controller. The size for other algorithms is the number of planes or belief points. The time is in seconds. 113

6.4 Results for DEC-POMDP problems comparing Mealy and Moore machines and other algorithms. The size refers to the number of nodes in the controller and the time is in seconds. 114

6.5 Results for Mealy and Moore machines of different sizes for DEC-POMDP benchmarks. A blank entry means that the controller of that size could not been computed given the resource restrictions of the NEOS server. 116

7.1 Heuristic Policy Iteration for DEC-POMDPs. 122

7.2 The linear program used to determine if a node q for agent i is dominated at each point b and all initial nodes of the other agents' controllers. As node q may be dominated by a distribution of nodes, variable $x(\hat{q}_i)$ represents $P(\hat{q}_i)$, the probability of starting in node $\hat{q}$ for agent i. 123

7.3 Running time and value produced for each horizon using PBIP with and without incremental policy generation (IPG). 128

8.1 The values produced by each method along with the number of nodes in the controllers and the running time in seconds. 154

LIST OF FIGURES

Figure **Page**

2.1 Examples of POMDP policies represented as (a) a policy tree and (b) a deterministic finite-state controller. 14

2.2 Value vector illustrations for a problem with 2 states. In (a) the values for policies p_1-p_5 is presented. In (b), the maximum value for all belief states is highlighted. In (c) only policies that contribute to this maximum value are retained and the remaining policies are pruned. 18

2.3 Policy trees and value vectors in value iteration: the first step (a), the second step, an exhaustive backup (b), and after pruning (c). 19

2.4 Controllers and value vectors in policy iteration: the previous step controller (a), the backed up controller (for action 1 only) (b), and after pruning (c). 20

3.1 A set of horizon three policy trees (a) and two node stochastic controllers (b) for a two agent DEC-POMDP. 28

4.1 Example of (a) limiting states after actions are chosen and observations seen and (b) pruning with a reduced state set, S'. 47

5.1 Hallway problem values for various controllers sizes (top) and running times (bottom) using the NLP formulation with and without fixed actions at each node. Bars are used in the left figure to represent the minimum and maximum values for each size. 84

5.2 Recycling robots values for various controller sizes (top) and running times (bottom) using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS 88

5.3 Multiagent tiger problem values for various controller sizes (top) and running times (bottom) using NLP methods with and without a 2 node correlation device and BFS 89

5.4 Meeting in a grid problem values for various controller sizes (top) and running times (bottom) using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS 91

6.1 Deterministic Moore and Mealy machines 100

7.1 Comparison of the dynamic programming algorithms on (a) the two agent tiger problem, (b) the meeting in a grid problem and (c) the box pushing problem. The value produced by policy iteration with and without bounded backups as well as our heuristic policy iteration with and without optimizing the NLP were compared on each iteration until the time or memory limit was reached. 125

7.2 Running times for different values of MaxTrees on the Box Pushing problem with horizon 10. 129

7.3 Comparison of the heuristic and optimal incremental policy generation algorithms on (a) the two agent tiger, (b) recycling robot and (c) the box pushing problems. 133

8.1 An example of a controller generated from a set of trajectories. The trajectories all begin at an initial state and end when a goal condition has been satisfied. The initial controller is then created from these trajectories and redundant nodes are combined to produce the reduced controller. This controller is then optimized by evaluating it in conjunction with the controllers of the other agents and optimizing the action choices and transition choices as shown in a) and b) respectively. 148

CHAPTER 1

INTRODUCTION

1.1 Overview

Developing effective methods for reasoning under uncertainty is a thriving research area in artificial intelligence. In domains of this kind, agents must choose a plan of action based on partial or uncertain information about the world. Due to stochastic actions and noisy sensors, agents must reason about many possible outcomes and the uncertainty surrounding them. When multiple agents are present, they must also reason about the choices of the others and how they may affect the environment. Even single agent problems may require sophisticated reasoning to determine a high quality solution.

Some single agent example domains include robot control [100], medical diagnosis [51] and machine maintenance [34]. In the area of robot control, partial observability is used to model sensors that provide uncertain and incomplete information about the state of the environment. In a medical setting, the internal state of the patient is often not known with certainty. The machine maintenance problem seeks to find a cost-effective strategy for inspection and replacement of parts in a domain where partial information about the internal state is obtained by inspecting the manufactured products. Additional applications are discussed in [26].

Solving problems in which a group of agents must operate collaboratively in an environment based solely on local information is also an important challenge. When the agents must rely on uncertain partial knowledge about the environment, this is particularly difficult. The solution of this problem is then decentralized and each agent must cope with uncertainty about the other agents as well as imperfect information about the system state. The agents seek to optimize a shared objective function, but develop plans of action that are based on solely local information.

Many real world applications require decentralized solutions. Some examples include distributed robot control [14, 35] and networking problems [17, 47]. In multiagent domains, robot sensors not only provide uncertain and incomplete information about their own state, but also about the location and knowledge of the other robots. This lack of information leads to different perspectives on the state of the environment and complicates the planning process. Similarly, in a decentralized network, each node must make decisions about when and where to send packets without full awareness of the knowledge and actions of nodes in the rest of the network.

1.2 Models of decision making

To represent single agent problems in which a sequence of decisions must be made, the Markov decision process (MDP) can be used [88]. On each step, the agent observes the system state, chooses an action, receives a reward and the system transitions to a new state. This model allows for stochastic actions and assumes the agent has complete knowledge of the current state of the environment at each step. Due the Markov property, at any state the dynamics of the system depend only on the previous state and not a possibly arbitrary history of states. The goal of these problems is to maximize the cumulative reward over a finite or infinite number of steps, also called the *horizon* of a problem.

An extension of MDPs to deal with uncertainty about the system state is the partially observable Markov decision process (POMDP) model [57]. At each step, instead of seeing the system state, the agent observes a possibly noisy signal. These signals, or observations, can be used to estimate the system state. Using this estimate, a discrete state POMDP can be modeled as continuous state MDP.

When decentralized decision making is desired, the decentralized partially observable Markov decision process (DEC-POMDP) can be used [16]. In this model, each agent receives a separate observation and action choices are based solely on this local information. The dynamics of the system and the global reward depend on the actions taken by all of the agents. Thus, the agents seek to maximize a shared objective function while making choices based on local information. Also, because each agent receives local information, an estimation of the system state is generally not able to be calculated. More detail on each of these models is presented in Chapters 2 and 3.

1.3 Overview of relevant work

We first discuss several disciplines that helped to develop POMDPs and DEC-POMDPs and work on similar problems. We then describe recent work on developing algorithms for these models. Additional details on algorithms for POMDPs and DEC-POMDPs can be found in Chapters 2 and 3.

1.3.1 Relevant fields

The POMDP and DEC-POMDP models have grown from work in many research areas. We give a brief overview of the previous work that lead to these models and how the current work on these models applies to these fields.

Control Theory

Both POMDPs and DEC-POMDPs can be thought of as control problems. In control theory, agents are typically termed controllers and costs are used rather than rewards. Also, continuous systems are typically considered, while we will consider the discrete case (discrete states, actions and observations). Nevertheless, the mathematical basis for both models and their solutions derives from control theory.

The POMDP model has been studied in control theory since at least the 1960s. In 1965, Åstöm [10] discovered a method to convert POMDPs into continuous state MDPs, leading to future algorithmic breakthroughs.

A version of DEC-POMDPs has been studied in control theory for decades. Inspired by cooperative team models by Marschak [69] and Radner [90], nondynamical systems were first examined and systems with dynamics were studied later as team theory problems [52]. These problems represent decentralized stochastic control problems with nonclassical information structure due to the fact that the histories of the agents is not shared. While these problems have been studied extensively over the past four decades, algorithms have not been presented until recently. Current work includes a description of a subclass of nondynamical problems which allow the optimal solution to be more easily identified [31] and an approximate solution method for the dynamical problem [30].

Operations Research

POMDPs have been studied extensively in operations research. Sondik developed the first optimal algorithm for finite-horizon POMDPs (One-pass algorithm) [106] and Smallwood and Sondik [103] developed the first value iteration approach based on the piecewise linearity and convexity of the value function. These ideas were later extended to the discounted infinite-horizon case [107]. Other versions of value iteration were later developed (as surveyed in [25]) and the use of finite-state controllers was studied in infinite-horizon POMDPs [84]. Approximate methods were also developed [66]. While a large amount of research has been conducted on the centralized model, we are unaware of any work on the decentralized model in the operations research community.

Game theory

The DEC-POMDPs can be modeled as games where all the agents (players) share the same payoff function (pure coordination games). They can be represented as partially observable stochastic games (POSGs) or sequential games with imperfect information. Radner first introduced a one-shot (nondynamical) version of the problem [90] based on earlier work by Marschak [69]. These common payoff games have been studied, but few algorithms for solving them have been developed. Recently, there has been work on extending DEC-POMDPs to competitive problems by allowing the agents to possess different payoff functions using Bayesian games to model reasoning about other agents [39].

Artificial Intelligence – Planning and Multiagent Systems

Solving POMDPs or DEC-POMDPs can also be thought of from a planning perspective. In planning, a set of actions must be determined to achieve a goal. Classical planning has assumed complete knowledge of a static environment, deterministic results for actions and the lack of feedback during execution (open-loop planning). Due to these assumptions, a plan only needs to define a sequence of actions that reaches the goal. Methods such as theorem proving or search are then used to determine a plan of actions that reach a specified goal. If probabilistic actions are used, probabilistic planning methods can be employed. When the conditions of the system are completely observable at each step, these problems can be formulated as MDPs. As feedback is used, solving MDPs is a form of closed-loop planing. If uncertainty about the system is introduced, observations can be used and contingent planning can be conducted. This allows for plans that are robust to errors in the execution. For more details on various forms of planning see [37].

Solving POMDPs represents a form of contingent planning as the state of the system is not known with certainty and thus actions must be contingent on the sequence of observations that are seen. POMDPs are also probabilistic as the outcomes of actions is not known with certainty. There has been less work on planning for multiple agents in the AI community. DEC-POMDPs represent a cooperative multiagent planning model which is both closed-loop and probabilistic.

DEC-POMDPs can also be viewed from a multiagent systems viewpoint. Multiagent systems is a broad field that encompasses autonomous agents dealing with problems such as coordination, task allocation and communication in dynamic systems [115]. These problems are typically solved with protocols, agent organizations and other heuristic approaches. In contrast, we use a decision-theoretic approach to model the general cooperative multiagent problem. This allows for a mathematical framework for solving problems with a well defined optimal solution.

1.3.2 Solution methods

Developing effective algorithms for POMDPs is an active research area that has seen significant progress [19, 25, 45, 56, 57, 64, 70, 83, 85, 86, 104, 105, 108]. Despite this progress, it is generally accepted that exact solution techniques are limited to toy problems due to high memory requirements (approximately 10 states for infinite-horizon problems and 100 states for horizon 100 problems). Consequently, approximation algorithms are often necessary in practice. Approximation methods perform well in many problem domains, but have some known disadvantages. For instance, point-based methods [56, 83, 108, 104, 105] perform better with a short horizon or a small reachable belief space. Controller-based algorithms rely on local search methods based on gradient ascent [70] and linear programming [86] or require fine tuning of heuristic parameters to produce high-valued solutions [85].

There has also been promising work on exact and approximate algorithms for DEC-POMDPs [15, 17, 20, 23, 47, 72, 81, 75, 77, 95, 110, 112]. Like the POMDP case, the previous exact methods for finite-horizon [47, 112] and infinite-horizon [15] could only solve toy problems due to high resource requirements (approximately 2 states and agents for horizon 4 problems). Approximate approaches can perform well on some problems, but either perform poorly or are limited to short horizons or small problem sizes. For instance, Szer and Charpillet's algorithm [110] is limited to very small solutions because of high memory and running time requirements and Bernstein *et al.*'s method [17] is more scalable, but it often produces low-quality results. It is worth noting that many POMDP algorithms cannot be easily extended to apply to DEC-POMDPs. One reason for this is that the decentralized nature of the DEC-POMDP framework results in a lack of a shared belief state, making it impossible to properly estimate the state of the system. Thus, a DEC-POMDP cannot be formulated as a continuous state MDP and algorithms that use this formulation do not generalize.

Therefore, progress has been made with both models, but previous approaches were not sufficiently scalable to provide high quality solutions to large problems. As POMDPs are on the verge of being applicable to a larger number of real-world problems, a small amount of scalability may be all that is needed to allow the model to transition from being primarily theoretical to becoming a practical tool which is more widely used. The DEC-POMDP model, by virtue of being newer and more complex, does not yet have algorithms at the same level of scalability as the POMDP model. Consequently, the relative newness of the model also results in many open questions that may lead to much better algorithmic performance for both optimal and approximate approaches. These improvements will lay the foundation for future decentralized algorithms for a wide range of applications.

1.4 Summary of contributions

In this thesis, we improve the state-of-the-art for POMDP and DEC-POMDP algorithms in several ways. These contributions increase the scalability of POMDP and DEC-POMDP algorithms, while also increasing their solution quality.

More efficient dynamic programming for DEC-POMDPs We have developed a method called *incremental policy generation* that improves the performance of optimal dynamic programming for DEC-POMDP by drastically reducing the necessary search space through reachability analysis. This is accomplished by utilizing the information each agent possesses at each problem step to eliminate unreachable parts of the state space. This allows an agent to automatically use domain information to reduce the resource requirements for constructing optimal solutions. As this is a general dynamic programming approach, it can be used in any finite- or infinite-horizon dynamic programming algorithm for DEC-POMDPs, improving the resulting performance. This approach was first discussed in [7].

Optimizing fixed-size controllers We also show how to represent optimal memory-bounded solutions for POMDPs and DEC-POMDPs as nonlinear programs. These fixed-size solutions are an effective way of combating the intractable memory requirements of current algorithms for both models. We demonstrate how an optimal solution can be found for these models, but as a wide range of nonlinear program solvers may be used, we also discuss more efficient approximate methods. We demonstrate high-quality approximate results for both centralized and decentralized POMDPs with this method. The POMDP approach was first described in [4], while the DEC-POMDP approach appeared in [3]. These approaches, along with a more scalable representation and expanded results, were presented in [5].

Policies as mealy machines Existing controller-based approaches for centralized and decentralized POMDPs are based on automata with output known as Moore machines. We show that several advantages can be gained by utilizing another type of automata, the Mealy machine. Mealy machines are more powerful than Moore machines, provide a richer structure that can be exploited by solution methods and can be easily incorporated into current controller-based approaches. To demonstrate this, we adapted some existing controller-based algorithms to use Mealy machines and obtained results on a set of benchmark domains. The Mealy-based approach always outperformed the Moore-based approach and often out-performed the state-of-the-art algorithms for both centralized and decentralized POMDPs. Part of the discussion of this representation will be published in [6].

Approximate dynamic programming approaches for DEC-POMDPS Approximate DEC-POMDP algorithms can be improved by incorporating our incremental policy generation method or by sampling the state space and other agents' policies. Many approximate algorithms use dynamic programming [23, 32, 95, 94, 111], which allows incremental policy generation to be applied and thus the efficiency of the resulting algorithm to be improved [7]. We also demonstrate how sampling can be used to improve dynamic programming in a infinite-horizon setting, reducing the number of policies that are considered and in general improving solution quality. This heuristic policy iteration algorithm was presented in [15].

Achieving goals in DEC-POMDPs We have also developed algorithms for DEC-POMDPs in which the agents share joint goals. We first model these problems as indefinite-horizon, where the problems terminate after some unknown number of steps. Under assumptions that each agent has a set of terminal actions and the rewards for non-terminal actions are negative, we describe an optimal indefinite-horizon algorithm. When these assumptions are relaxed, but the agents continue to seek a goal, we describe an approximate sample-based algorithm. We provide bounds on the number of samples necessary to ensure that an optimal solution can be found with high probability. We also demonstrate experimentally that even when the number of samples is less than this bound, high-valued results can be found on a range of goal based problems. These approaches first appeared in [8].

1.5 Thesis organization

The thesis is organized as follows. Background information is provided in Chapters 2 and 3. Chapter 2 describes the POMDP model as well as related optimal and approximate solution approaches for both the finite and infinite-horizon cases. This chapter also includes other related single-agent work. Chapter 3 then presents the DEC-POMDP model and discusses how some concepts can be extended to the decentralized problem, but fundamental differences remain. We also present an overview of optimal and approximate algorithms that have been developed for DEC-POMDPs and other related work.

We begin discussing the contributions of this thesis in Chapter 4. This chapter presents a discussion of how to improve the efficiency of dynamic programming for DEC-POMDPs with reachability analysis. The finite- and infinite-horizon algorithms are presented as well as proof of correctness and experimental results.

Chapter 5 discusses how finite-state controllers can be optimized to provide fixed-memory solutions to POMDPs and DEC-POMDPs. We present an optimal formulation as a nonlinear program and then discuss optimal and approximate solutions for this representation. We provide experimental results which show the benefits of approximate solutions for generating concise, high quality solutions.

Chapter 6 introduces the use of Mealy machines in controller-based approaches for POMDPs and DEC-POMDPs. We describe what the difference is between a Mealy machine and the commonly used Moore machine as well as how to adapt algorithms to use Mealy machines. We also provide experimental results showing the increased solution quality this representation allows.

In Chapter 7 we describe our approximate dynamic programming approaches for DEC-POMDPs. These algorithms improve scalability by incorporating the reachability in chapter 4 into approximate dynamic programming as well as utilizing sampling of the state space and agent policies.

Chapter 8 describes our method for achieving goals in DEC-POMDPs. We first discuss our indefinite-horizon approach and prove it will generate an optimal solution. We then discuss our sample-based algorithm and provide bounds on the number of samples needed to approach optimality. We conclude the chapter with experimental results showing that our sample-based approach can outperform other DEC-POMDP algorithms on a range of goal based problems.

We conclude in Chapter 9. This provides a summary of the thesis contributions as well as possible future work.

Two appendices are also included. The first provides a more detailed description of the DEC-POMDP benchmark problems used in this thesis. The second provides additional information about the AMPL formulation used for representing the nonlinear program (NLP) for POMDPs and DEC-POMDPs as well as the benchmark domains.

CHAPTER 2

SINGLE AGENT DECISION-MAKING UNDER UNCERTAINTY

As the DEC-POMDP model is an extension of the POMDP model, we begin with an overview of MDPs and POMDPs. We also discuss POMDP solution representations as well as optimal and approximate algorithms. We end this chapter by briefly discussing related work on planning and learning in partially observable domains.

2.1 Markov decision processes

An MDP is a sequential model for decision making in fully observable environments. It can be defined with the following tuple: $M = \langle S, A, P, R, T \rangle$, with

- S, a finite set of states with designated initial state distribution b_0
- A, a finite set of actions
- P, the state transition model: $P(s'|s,a)$ is the probability of transitioning to state s' if action a is taken in state s
- R, the reward model: $R(s,a)$ is the immediate reward for taking action a in state s
- T, a horizon or number of steps after which the problem terminates

We consider the case in which the decision making process unfolds over a finite or infinite sequence of stages, the number of which is denoted by T. At each stage the agent observes the state of the system and selects an action. This generates an immediate reward and the system transitions stochastically to a new state. The objective of the agent is to maximize the expected discounted sum of rewards received over all steps of the problem. In the infinite-horizon problem (which proceeds for an infinite number of steps), we use a discount factor, $0 \leq \gamma < 1$, to maintain finite sums.

Many algorithms have been developed for solving MDPs exactly and approximately. The most common approaches include dynamic programming and linear programming, which can be used to solve finite-horizon MDPs in polynomial time. For more details see Puterman [88].

2.2 Partially observable Markov decision processes

POMDPs extend MDPs to environments in which agents receive an observation rather than the exact state of the system at each step. In this section, we first give a formal description of the POMDP model and then describe methods of representing finite and infinite-horizon solutions.

2.2.1 The POMDP model

A POMDP can be defined with the following tuple: $M = \langle S, A, P, R, \Omega, O, T \rangle$, with

- S, a finite set of states with designated initial state distribution b_0
- A, a finite set of actions
- P, the state transition model: $P(s'|s, a)$ is the probability of transitioning to state s' if action a is taken in state s
- R, the reward model: $R(s, a)$ is the immediate reward for taking action a in state s
- Ω, a finite set of observations

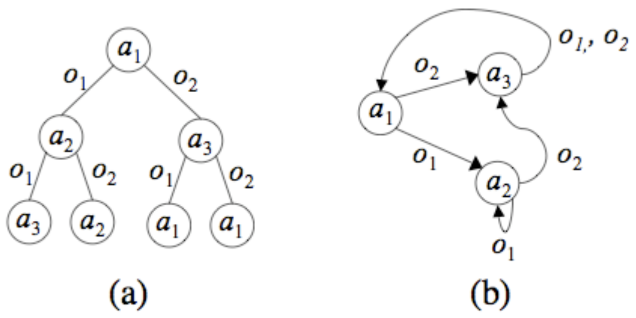


Figure 2.1. Examples of POMDP policies represented as (a) a policy tree and (b) a deterministic finite-state controller.

- O, the observation model: $O(o|s', a)$ is the probability of observing o if action a is taken and the resulting state is s'
- T, a horizon or number of steps after which the problem terminates

As with an MDP, we consider the case in which the problem unfolds over a finite or infinite sequence of stages. At each stage, the agent receives some observation about the state of the system and then chooses an action. Like an MDP, this yields an immediate reward and the system transitions stochastically. But in this case, because the state is not directly observed, it may be beneficial for the agent to remember the observation history. This will help the agent by allowing it to calculate a probability distribution over states at any step (the belief state). The objective of maximizing expected discounted sum of rewards received remains the same and thus a discount factor, γ, is used for the infinite-horizon problem.

2.2.2 Solution representations

To represent policies for finite-horizon POMDPs, policy trees can be used. An agent's policy tree can be defined recursively. The tree begins with an action at the root and a subtree is defined for each observation that the agent could see. This continues until the horizon of the problem is achieved at the leaf nodes. Thus, the agent's choice of actions is defined by a path through the tree that depends on the observations that it sees. An example of this representation can be seen in Figure 2.1(a). A policy tree can be evaluated at any state, s, by starting at the root and weighing the subtrees by their likelihood as follows

$$V(q,s) = R(a_q,s) + \sum_{s'} P(s'|a_q,s) \sum_{o} P(o|a_q,s') V(q^o,s')$$

where a_q is the action defined at the root of tree q, while q^o is the subtree of q that is visited after o has been seen.

For infinite-horizon POMDPs, finite-state controllers can be used instead of policy trees. These controllers are an elegant way of representing policies using a finite amount of memory [45]. The state of the controller is based on the observation sequence seen, and in turn the agent's actions are based on the state of its controller. To help distinguish states of the finite-state controller from states of the POMDP, we will refer to controller states as nodes. These controllers can address one of the main causes of intractability in POMDP exact algorithms by not storing whole observation histories. Thus, states of the controller can encapsulate key information about the observation history in a fixed number of nodes. An example of a deterministic finite-state controller is shown in Figure 2.1(b).

We also allow for stochastic transitions and action selection, as this can help to make up for limited memory [101]. We provide an example showing this phenomenon in a previous paper [4]. The finite-state controller can formally be defined by the tuple $\langle Q, \psi, \eta \rangle$, where Q is the finite set of controller nodes, $\psi : Q \rightarrow \Delta A$ is the action selection model for each node, mapping nodes to distributions over actions, and $\eta : Q \times A \times O \rightarrow \Delta Q$ represents the node transition model, mapping nodes, actions and observations to distributions over the resulting nodes. Thus, starting at a given controller node, an action is taken based on the node's action selection distribution and after an observation is seen, the controller transitions to another node based on the node transition model. This process of stochastically selecting actions and transitioning continues for the infinite steps of the problem. The value of a node q at state s, given action selection and node transition probabilities $P(a|q)$ and $P(q'|q,a,o)$, is given by the following Bellman equation:

$$V(q,s) = \sum_a P(a|q)\Big[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_o O(o|s',a) \sum_{q'} P(q'|q,a,o)V(q',s')\Big]$$

2.3 Optimal POMDP solution methods

In this section we discuss optimal dynamic programming methods for solving POMDPs: value iteration and policy iteration. We first present some important concepts used in these algorithms and then describe the algorithms themselves.

2.3.1 Important concepts

We will describe how discrete state POMDPs can be represented as continuous state MDPs as well as how to represent the values of policies across these continuous states.

Belief state MDP

A key concept in POMDPs is that the observation seen at each step can be used to update an estimate of the state. This estimate is called a *belief state* and will be denoted b. After the agent chooses action a in belief state b and observes o the probability that the resulting state is s' can be updated as follows:

$$b'(s') = \frac{P(o|a,s') \sum_{s \in S} P(s'|s,a)b(s)}{P(o|b,a)},$$

where

$$P(o|b,a) = \sum_{s' \in S} \left[P(o|a,s') \sum_{s \in S} P(s'|s,a)b(s) \right].$$

The belief state is a sufficient statistic of the observation history. Because of this, we can transform a discrete state POMDP into a continuous state MDP, keeping track of the distribution of states at each step rather than all possible observation histories. This allows the continuous state MDP to be solved using the tools described below.

Value vectors

We can represent the value of a policy, p, using a vector which provides the value at each state of the problem, s, denoted as $V(p,s)$. We provide a graphical representation of these vectors for a problem with 2 states in Figure 2.2. Because we are seeking to maximize the value at each belief state of the system, we can choose the best policy at each belief state, which is represented by the piecewise linear convex upper surface formed by these vectors. The maximum value at a belief state b is given by

$$V(b) = \max_p \sum_s b(s)V(p,s)$$

This upper surface is highlighted in Figure 2.2(b).

What about policies that are not part of this maximum value? These policies have lower value for all possible belief states and as such would never be used in an optimal solution. This leads to the concept of *pruning dominated vectors*. If a vector has lower value than some distribution of other vectors for all belief states, then the associated policy plays no part in an optimal solution as the other vectors would always be better choices. A linear program can be used to determine dominated policies (shown below). This linear program seeks to find a distribution of vectors that has higher value for all belief states. This is done for a policy p by maximizing a value ϵ, which represents the value difference between the vector for p and some distribution of other vectors $\hat{p}$. If a positive ϵ value can be found, then there is some distribution of vectors $\hat{p}$ that has greater value at all belief states b. In addition to ensuring that the belief states are valid probability distributions, the the linear program can be represented as

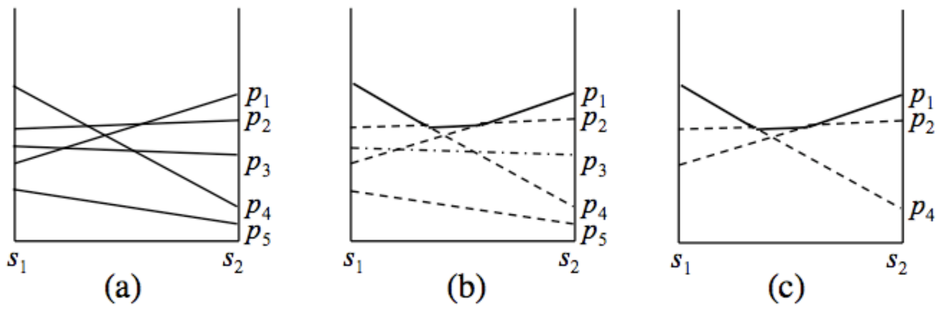


Figure 2.2. Value vector illustrations for a problem with 2 states. In (a) the values for policies p_1-p_5 is presented. In (b), the maximum value for all belief states is highlighted. In (c) only policies that contribute to this maximum value are retained and the remaining policies are pruned.

$$\max \epsilon$$

$$\text{subject to } \forall \hat{p} \quad \sum_s b(s)V(\hat{p},s) + \epsilon \leq \sum_s b(s)V(p,s)$$

We can then prune these dominated vectors as shown in Figure 2.2(c) without changing the value function for any belief state.

2.3.2 Value iteration

The removal of dominated policies leads to a dynamic programming algorithm for optimally solving POMDPs. This value iteration algorithm [106] builds up policies from the bottom up by using increasingly larger horizon policy trees. That is, on the first step of dynamic programming, a policy for the last step of the problem is sought. Because it is the last step, the agent chooses only a single action. Thus, all actions are considered and if any action is dominated across all belief states, it is removed from consideration. The remaining actions are those that have the best 1-step value for some belief state of the system.

On the next step, 2-step policies are constructed by building all possible horizon 2 trees that have the undominated actions as possibilities on the second step. Thus, if there are $|P|$ actions that were not dominated on the previous step, there will be $|A||P|^{|\Omega|}$ horizon 2 trees. This generation of next step trees is often referred to as a

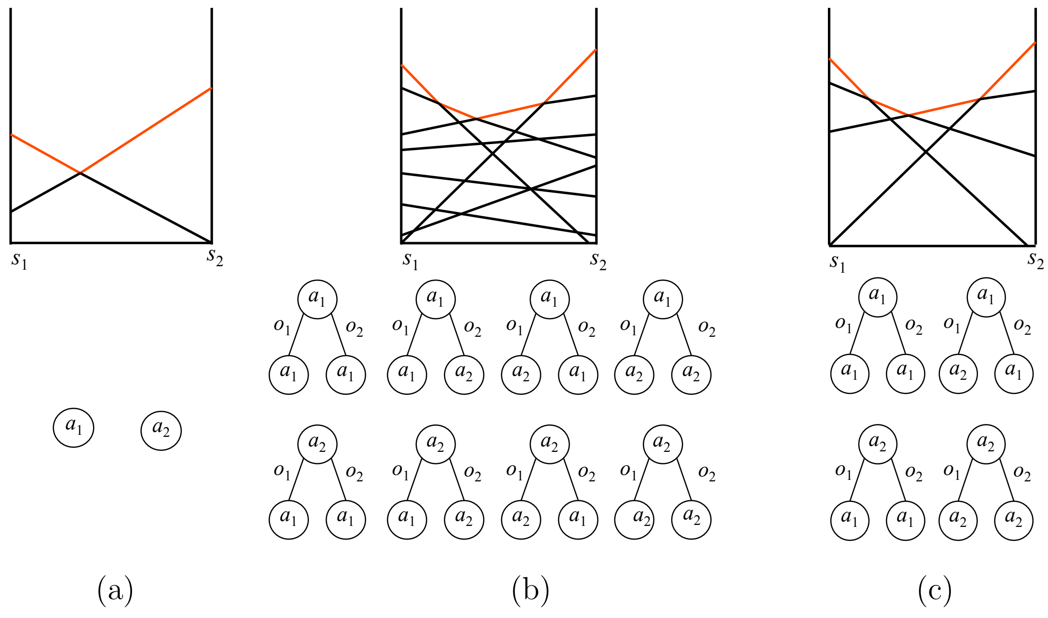


Figure 2.3. Policy trees and value vectors in value iteration: the first step (a), the second step, an exhaustive backup (b), and after pruning (c).

backup. These 2-step trees can then be evaluated and the dominated ones pruned. These remaining policies represent those that provide the best value for any possible belief state after $k-2$ steps of the solution have already been taken. Larger horizon trees can then be built by repeating the steps of backing up and pruning. The final set of trees includes a tree that is optimal for any given initial belief state for the desired horizon. An example of value iteration for two steps is show in Figure 2.3

Note that value iteration can be performed without storing the policy trees, but instead updating only the value vectors at each step. This is accomplished by choosing an action and then determining the values for each belief state that results from seeing the possible observations. It has been shown that the optimal value function is piecewise linear and convex and the dynamic programming backups retain these properties [103]. After k steps the optimal value function for a horizon k problem is known. An optimal policy can then be recovered by using a one-step lookahead, choosing actions with

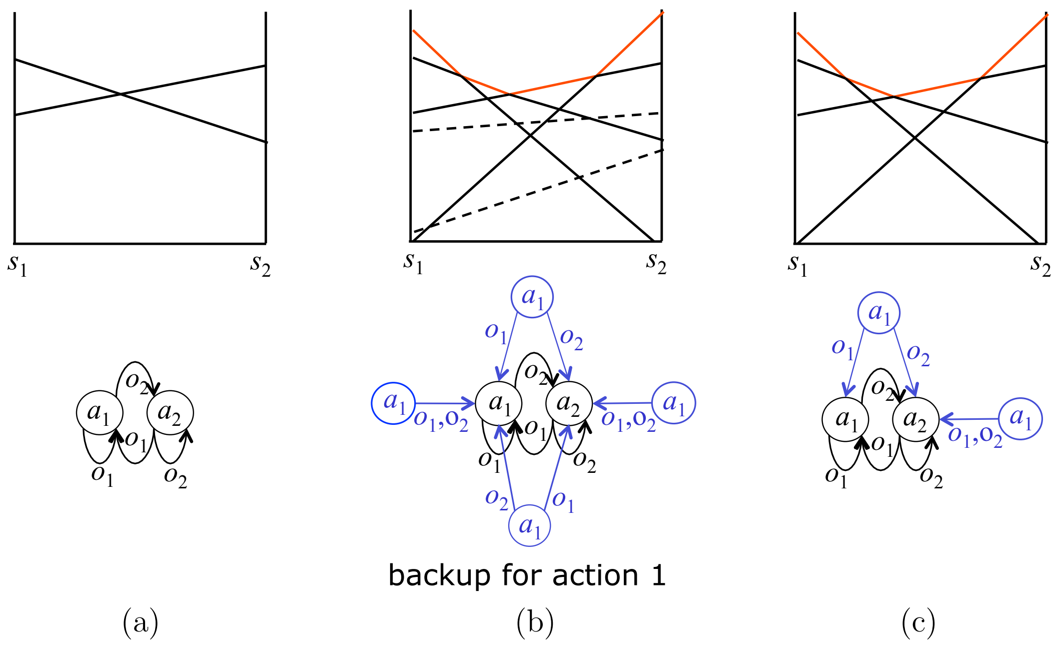


Figure 2.4. Controllers and value vectors in policy iteration: the previous step controller (a), the backed up controller (for action 1 only) (b), and after pruning (c).

$$\underset{a}{\operatorname{argmax}} \left[\sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o} P(o|b,a)V(\tau(b,o,a)) \right]$$

where $\tau(b,o,a)$ is the belief state that results from starting in b, choosing action a and observing o.

For infinite horizon problems, an ϵ-optimal value function can still be found in a finite number of steps with value iteration. This is accomplished by building the value up until it changes by only a small amount for any belief state. The possible value for further steps can then be bounded and convergence is achieved. This algorithm can also be made more efficient by building up the value function without using exhaustive backups. We will discuss one of these methods in Chapter 4.

2.3.3 Policy Iteration

Finding an optimal solution for infinite-horizon POMDPs have been shown to be undecidable because an infinitely sized controller may be required [67]. As a result, policy iteration [45] has been developed to produce ϵ-optimal solutions for infinite-horizon POMDPs. A controller is built up in a manner that is similar to that used in value iteration, but the algorithm is broken into policy improvement and evaluation phases. The controller first is initialized to be an arbitrary one-node controller. In the improvement phase, dynamic programming is used to enlarge the controller. This is the same backup used to create next step policy trees in value iteration and creates a node for each possible combination of action and transition for each observation to any of the nodes in the current controller. The controller can then be evaluated, but pruning is more complicated.

Using the test of dominance described earlier, nodes that have lesser or equal value for all states are pruned and replaced by nodes that (pointwise) dominate them. The incoming edges of these nodes are redirected to the dominating nodes, guaranteeing at least equal value. Because the transitions in the controller may have changed, we must reevaluate the controller. The value of starting the controller at any node can be evaluated for all states by using a system of Bellman equations (as shown in Section 2.2.2). This process continues until the controller is no longer changed by the improvement phase or the value of further improvement can be bounded by ϵ. An example of policy iteration is shown in Figure 2.4. Note that nodes for all actions are created during a backup, but only those for action 1 are shown for the sake of clarity.

2.4 Approximate solution methods

We first discuss controller-based methods for infinite-horizon POMDPs and then give a brief overview of point-based methods for finite- and infinite-horizon POMDPS.

2.4.1 Controller-based methods

Poupart and Boutilier [86] developed a method called bounded policy iteration (BPI) that uses a one step dynamic programming lookahead to attempt to improve

For a given node q and variables x_a and $x_{q',a,o}$
Maximize ϵ, subject to
Improvement constraints:
$\forall s\ V(q,s)+\epsilon \leq \sum_a \left[x_a R(s,a)+\gamma \sum_{s'} P(s'|s,a) \sum_o O(o|s',a) \sum_{q'} x_{q',a,o} V(q',s')\right]$
Probability constraints:
$\sum_a\ x_a = 1, \quad \forall a\ \sum_{q'} x_{q',a,o} = x_a, \quad \forall a\ x_a \geq 0, \quad \forall q', a, o\ x_{q',a,o} \geq 0$

Table 2.1. The linear program used by BPI. Variables x_a and $x_{q',a,o}$ represent $P(a|q)$ and $P(q',a|q,o)$ for a given node, q.

a POMDP controller without increasing its size. Like policy iteration (PI), this approach alternates between policy improvement and evaluation. It iterates through the nodes in the controller and uses a linear program, shown in Table 2.1, to examine the value of probabilistically taking an action and then transitioning into the current controller. If an improvement (a higher-valued action and transition) can be found for all states, the action selection and node transition probabilities are updated accordingly. The controller is then evaluated and the cycle continues until no further improvement can be found. BPI guarantees to at least maintain the value of a provided controller, but it also does not use start state information resulting in larger than necessary controllers that are not likely to be optimal.

To increase the performance of BPI, Poupart and Boutilier added two improvements. First, they allowed the controller to grow by adding nodes when improvements can no longer be made. This may permit a higher value to be achieved than what BPI can produce with a fixed-size controller. Second, they used a heuristic for incorporating start state knowledge which increases the performance of BPI in practice [85]. In this extension, termed biased BPI, improvement is concentrated in certain node and state pairs by weighing each pair by the (unnormalized) occupancy distribution, which can be found by solving the following set of linear equations:

$$o(q', s') = bp_0(q', s') + \gamma \sum_{q,s,a,o} o(q, s)P(s'|s, a)P(a|q)O(o|s', a)P(q'|q, a, o)$$

for all states and nodes. The value bp_0 is the probability of beginning in a node state pair. A factor, δ, can also be included, which allows the value to decrease by that amount in each node and state pair. This makes changes to the parameters more likely, as a small amount of value can now be lost. As a result, value may be higher for the start state and node, but as value can be lost for any pair, it could also be lower instead.

Meuleau et al. [70] have proposed another approach to improve a fixed-size controller. They used gradient ascent (GA) to change the action selection and node transition probabilities in order to increase value. A cross-product MDP is created from the controller and the POMDP by considering the states of the MDP to be all combinations of the states of the POMDP and the nodes of the controller while actions of the MDP are based on actions of the POMDP and deterministic transitions in the controller after an observation is seen. The value of the resulting MDP can be determined and matrix operations allow the gradient to be calculated. The gradient can then be followed in an attempt to improve the controller.

2.4.2 Point-based methods

Point-based methods have recently gained a high level of popularity. A range of methods have been developed such as point-based value iteration (PBVI) [83], PERSEUS [108], heuristic search value iteration (HSVI) [104] and point-based policy iteration (PBPI) [56]. All these techniques except PBPI are approximations of value iteration for finite-horizon POMDPs, but they can find solutions for sufficiently large horizons that they can effectively produce infinite-horizon policies.

PBVI, PERSEUS, HSVI and PBPI fix the number of belief points considered at each step of value or policy iteration. That is, at each step, the value vector of each belief point under consideration is backed up and the vector that has the highest value at that point is retained. This permits the number of vectors to remain constant and the value function is still defined over the entire belief space. If the right points are chosen, it may help concentrate the value iteration on certain regions in the belief space that will be visited by an optimal policy. PBVI selects the set of belief points by sampling from the action and observation sets of the problem while PERSEUS backs up only a randomly selected subset of these points. Thus, PERSEUS is often faster and produces a more concise solution than PBVI. PBVI has been shown to be optimal for sufficiently large and well constructed belief point sets, but this is intractable for reasonably sized problems. HSVI chooses points based on an additional upper bound. This algorithm will also converge to the optimal solution, but again this often requires an intractable amount of resources. PBPI uses Hansen's policy iteration, but with PBVI rather than exact improvement over the whole belief space in the improvement phase. This allows faster performance and sometimes higher values than the original PBVI.

2.5 Additional models and approaches

In addition to planning, research has also focused on learning in POMDPs. For instance, reinforcement learning methods for MDPs [109] have been extended to POMDPs [63]. Learning the model is often difficult [97], resulting in a number of approaches which learn the policy directly [11, 71]. Bayesian reinforcement learning techniques have recently been studied to incorporate domain knowledge into the learning algorithm [87].

Because of the difficulty in learning POMDPs, predictive state representations (PSRs) [65] have also been explored. This representation facilitates learning by using a more concise representation of test (action and observation sequence) predictions instead of the belief state used in a POMDP [102, 117].

Other approaches have also been used to combat the high complexity of solving POMDPs. These include table-based value function estimates [19], value function approximation with dimensionality reduction [93, 62] and other value function approximation methods [50]. Online planning can also be used to quickly produce approximate results while acting [91].

CHAPTER 3

DECENTRALIZED DECISION-MAKING UNDER UNCERTAINTY

In this chapter, we first describe the DEC-POMDP model, highlighting the similarities and differences compared to the POMDP model. We formalize solution representations for DEC-POMDPs using policy trees and finite-state controllers. We also discuss what concepts from POMDPs continue to hold for DEC-POMDPs and which do not. Finally, we present the optimal dynamic programming algorithms for finite- and infinite-horizon DEC-POMDPs as well as an overview of approximate approaches. Lastly, we discuss subclasses of DEC-POMDPs and other modeling assumptions. For a more detailed description of the various specialized DEC-POMDP models as well as their relation to other multiagent models, see [96].

3.1 Decentralized POMDPs

The DEC-POMDP model is an extension of MDPs and POMDPs in which a set of agents must make decisions based on local information. The dynamics of the system and the objective function depend on the actions of all agents.

3.1.1 The DEC-POMDP model

A DEC-POMDP can be defined with the tuple: $\langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O, T \rangle$ with

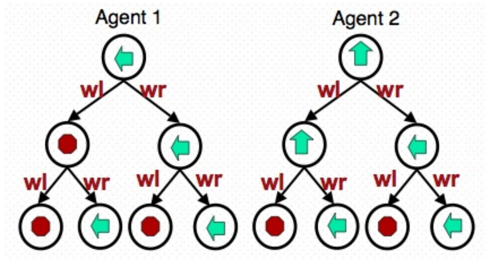
- I, a finite set of agents
- S, a finite set of states with designated initial state distribution b_0
- A_i, a finite set of actions for each agent, i

- P, a set of state transition probabilities: $P(s'|s, \vec{a})$, the probability of transitioning from state s to s' when the set of actions $\vec{a}$ are taken by the agents
- R, a reward function: $R(s, \vec{a})$, the immediate reward for being in state s and taking the set of actions $\vec{a}$
- Ω_i, a finite set of observations for each agent, i
- O, a set of observation probabilities: $O(\vec{o}|s', \vec{a})$, the probability of seeing the set of observations $\vec{o}$ given the set of actions $\vec{a}$ was taken which results in state s'
- T, a horizon or number of steps after which the problem terminates

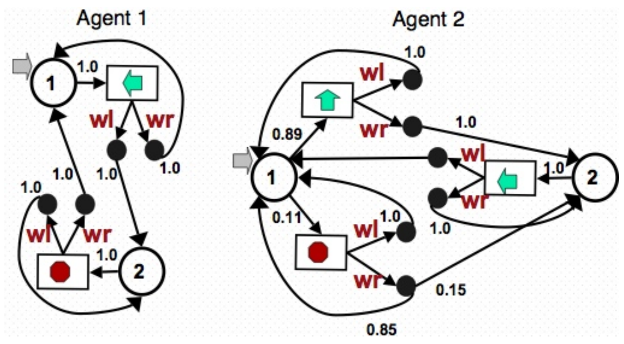
A DEC-POMDP involves multiple agents that operate under uncertainty based on different streams of observations. Like a POMDP, a DEC-POMDP unfolds over a finite or infinite sequence of stages. At each stage, every agent chooses an action based purely on its local observations, resulting in an immediate reward for the set of agents and an observation for each individual agent. Again, because the state is not directly observed, it may be beneficial for each agent to remember its observation history. A *local policy* for an agent is a mapping from local observation histories to actions while a *joint policy* is a set of policies, one for each agent in the problem. Like the POMDP case, the goal is to maximize the total cumulative reward, beginning at some initial distribution over states b_0. For the infinite horizon case a discount factor, γ, is again used to maintain finite sums. Note that in general the agents do not observe the actions or observations of the other agents, but the rewards, transitions and observations depend the decisions of all agents. This makes solving a DEC-POMDP much more difficult than solving a POMDP.

3.1.2 Solution representations

For finite-horizon problems, local policies can be represented using a policy tree as seen in Figure 3.1a. This is similar to policy trees for POMDPs, but now each agent possesses its own tree that is independent of the other agent trees. Actions are represented by the arrows or stop figures (where each agent can move in the given



(a)



(b)

Figure 3.1. A set of horizon three policy trees (a) and two node stochastic controllers (b) for a two agent DEC-POMDP.

direction or stay where it is) and observations are labeled “wl” and “wr” for seeing a wall on the left or the right respectively. Using this representation, an agent takes the action defined at the root node and then after seeing an observation, chooses the next action that is defined by the respective branch. This continues until the action at a leaf node is executed. For example, agent 1 would first move left and then if a wall is seen on the right, the agent would move left again. If a wall is now seen on the left, the agent does not move on the final step. A policy tree is a record of the the entire local history for an agent up to some fixed horizon and because each tree is independent of the others it can be executed in a decentralized manner.

These trees can be evaluated by summing the rewards at each step weighted by the likelihood of transitioning to a given state and observing a given set of observations. For a set of agents, the value of trees $\vec{q}$ while starting at state s is given recursively by:

$$V(\vec{q}, s) = R(\vec{a}_{\vec{q}}, s) + \sum_{s'} P(s'|\vec{a}_{\vec{q}}, s) \sum_{\vec{o}} P(\vec{o}|\vec{a}_{\vec{q}}, s')V(\vec{q}^{\vec{o}}, s')$$

where $\vec{a}_{\vec{q}}$ are the actions defined at the root of trees $\vec{q}$, while $\vec{q}^{\vec{o}}$ and are the subtrees of $\vec{q}$ that are visited after $\vec{o}$ have been seen.

While this representation is useful for finite-horizon problems, infinite-horizon problems would require trees of infinite height. Another option is to condition action selection on some internal memory state. These solutions can be represented as a set of local finite-state controllers (seen in Figure 3.1b). Again, these controllers are similar to those used by POMDPs except each agent possesses its own independent controller. The controllers operate in a very similar way to the policy trees in that there is a designated initial node and following the action selection at that node, the controller transitions to the next node depending on the observation seen. This continues for the infinite steps of the problem. Throughout this thesis, controller states will be referred to as nodes to help distinguish them from system states.

An infinite number of nodes may be required to define an optimal infinite-horizon DEC-POMDP policy, but we will discuss a way to produce solutions within ϵ of the optimal with a fixed number of nodes. While deterministic action selection and node transitions are sufficient to define this ϵ-optimal policy, when memory is limited stochastic action selection and node transition may be beneficial. A simple example illustrating this for POMDPs is given by Singh [101], which can be easily extended to DEC-POMDPs. Intuitively, randomness can help an agent to break out of costly loops that result from forgetfulness.

A formal description of stochastic controllers for POMDPs and DEC-POMDPs is given in Chapter 5, but an example can be seen in Figure 3.1b. Agent 2 begins at node 1, moving up with probability 0.89 and staying in place with probability 0.11. If the agent stayed in place and a wall was then seen on the left (observation "wl"), on the next step, the controller would transition to node 1 and the agent would use the same distribution of actions again. If a wall was seen on the right instead (observation "wr"), there is a 0.85 probability that the controller will transition back to node 1 and a 0.15 probability that the controller will transition to node 2 for the next step. The finite-state controller allows an infinite-horizon policy to be represented compactly by remembering some aspects of the agent's history without representing the entire local history.

We can evaluate the joint policy by beginning at the initial node and transition through the controller based on the actions taken and observations seen. The value for starting in nodes $\vec{q}$ and at state s with action selection and node transition probabilities for each agent, i, is given by the following Bellman equation: $V(\vec{q}, s) =$

$$\sum_{\vec{a}} \prod_{i} P(a_i|q_i) \Big[R(s, \vec{a}) + \gamma \sum_{s'} P(s'|\vec{a}, s) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q'}} \prod_{i} P(q'_i|q_i, a_i, o_i) V(\vec{q'}, s') \Big]$$

Note that the values can be calculated offline in order to determine a controller for each agent that can then be executed online in a decentralized manner.

3.2 Optimal DEC-POMDP solution methods

The decentralized nature of DEC-POMDPs makes them fundamentally different than POMDPs. We explain some of these differences, describe what extensions can be made from POMDPs and then discuss the optimal dynamic programming algorithms.

3.2.1 The difficulty of extending POMDP methods

In a DEC-POMDP, the decisions of each agent affect all the agents in the domain, but due to the decentralized nature of the model each agent must choose actions based solely on local information. Because each agent receives a separate observation that does not usually provide sufficient information to efficiently reason about the other agents, solving a DEC-POMDP optimally becomes very difficult. For example, each agent may receive a different piece of information that does not allow a common state estimate or any estimate of the other agents' decisions to be calculated. These state estimates are crucial in single agent problems, as they allow the agent's history to be summarized concisely (as belief states), but they are not generally available in DEC-POMDPs. The lack of state estimates (and thus the lack of a sufficient statistic) requires agents to remember whole action and observation histories in order to act optimally. This also means that we can't transform DEC-POMDPs into belief state MDPs and must use a different set of tools to solve them. Another way to consider this problem is that there is a third type of uncertainty in DEC-POMDPs. Beyond uncertain action outcomes and state uncertainty, there is also uncertainty about what choices the other agents make and what information they have.

This difference is seen in the complexity of the finite-horizon problem with at least two agents, which is NEXP-complete [16] and thus in practice may require double exponential time. This is in contrast to the MDP which is P-complete [78] and the POMDP which is PSPACE-complete [78]. Like the infinite-horizon POMDP, optimally solving an infinite-horizon DEC-POMDP is undecidable as it may require infinite resources, but ϵ-optimal solutions can be found with finite time and memory. Nevertheless, introducing multiple decentralized agents causes a DEC-POMDP to be significantly more difficult than a single agent POMDP.

3.2.2 Generalized belief states

As mentioned above, from an agent's perspective, not only is there uncertainty about the state, but also about the policies of the other agents. If we consider the possible policies of the other agents as part of the state of the system, we can form a *generalized belief state.* The generalized belief space then includes all possible distributions over states of the system and policies of the other agents. Thus, for two agents, the value of an agent's policy p at a given generalized belief state b_G is given by

$$V(p, b_G) = \sum_{q,s} b_G(s, q) V(p, q, s)$$

It is worth noting that if all other agents have known policies, the generalized belief state is the same as a POMDP belief state and we can solve the DEC-POMDP for the remaining agent as a POMDP.

As with POMDPs, the generalized belief state allows the values of policies to be represented as vectors over the all states of the system and policies of the other agents. If the set of policies possessed by the other agents is Q this new space is $S \times Q$. This results in a much larger dimensionality than that of POMDPs, but once policies are evaluated, they can be pruned in this space. Intuitively, this is because if a policy is not useful for any state of the system and set of possible policies used by the other agents, then it can be removed.

The difficulty with generalized belief states is that when the sets of policies change for the agents, the generalized belief states also change. Thus, if we add other policies for the other agents an agent's policies may no longer be able to be pruned. Conversely, if we remove policies for the other agents, certain policies may now be prunable. So without considering all possible policies for the other agents for a given horizon, we cannot reliably determine which policies are useful. Because this would amount to an exhaustive policy search, more efficient options are desired. We present one such option below.

3.2.3 Finite-horizon dynamic programming

Hansen et al. [47] developed a dynamic programming algorithm to optimally solve a finite-horizon DEC-POMDP. While this removes very weakly dominated strategies in partially observable stochastic games, DEC-POMDPs are a subclass in which the agents share a common payoff function and thus an optimal policy will be retained with this procedure. It is similar to the value iteration approach for POMDPs (see 2.3.2), except generalized belief states are used which results in a more complicated pruning step.

In this algorithm, a set of T-step policy trees, one for each agent, is generated from the bottom up. That is, on the Tth (and last) step of the problem, each agent will perform just a single action, which can be represented as a 1-step policy tree.

For agent i's given tree q_i and variables ϵ and $x(q_{-i}, s)$
Maximizes ϵ, given: $\forall\ \hat{q}_i$

$$\sum_{q_{-i},s} x(q_{-i}, s)V(\hat{q}_i, q_{-i}, s) + \epsilon \leq \sum_{q_{-i},s} x(q_{-i}, s)V(q_i, q_{-i}, s)$$

$$\sum_{q_{-i},s} x(q_{-i}, s) = 1 \quad \text{and} \quad \forall q_{-i}, s\ x(q_{-i}, s) \geq 0$$

Table 3.1. The linear program that determines if agent i tree, q_i is dominated by comparing its value to other trees for that agent, $\hat{q}_i$. The variable $x(q_{-i}, s)$ is a distribution over trees of the other agents and system states.

All possible actions for each agent are considered and each combination of these 1-step trees is evaluated at each state of the problem. Any action that has lower value than some other action for all states and possible actions of the other agents is then removed, or pruned. The linear program used to determine whether a tree can be pruned is shown in Table 3.1. We maximize ϵ while ensuring the variable x representing the generalized belief state remains a proper probability distribution and testing to see if there is some distribution of trees that has higher value for all states and policies of the other agents. Because there is always an alternative with at least equal value, regardless of system state and other agent policies, a tree q_i can be pruned if ϵ is nonpositive.

Note that the test for dominance is used for trees of a given horizon. This ensures that we consider all possible policies for a given horizon and remove those that are not useful no matter what policies are chosen by the other agents. If policies are removed, then as noted above, the belief space becomes smaller for the other agents and more policies may be able to be pruned. Thus, we can keep testing for dominated policies for each agent until no agent is able to prune any further policies.

On the next step, all 2-step policy trees are generated. This is done for each agent by an *exhaustive backup* of the current trees. That is, for each action and each resulting observation some 1-step tree is chosen. If an agent has $|Q_i|$ 1-step trees,

$|A_i|$ actions, and $|\Omega_i|$ observations, there will be $|A_i||Q_i|^{|\Omega_i|}$ 2-step trees. After this exhaustive generation of next step trees is completed for each agent, pruning is again used to reduce their number.

To generate trees for a given horizon, we can continue backing the trees up and pruning until the desired horizon is reached. The resulting set of trees will contain an optimal solution for that horizon and any initial state of the system. This is due to the fact that we considered the possible trees at each step and removed only those that are not useful no matter what policies the other agents choose at that step. This conservative removal of trees ensures that we can safely remove trees that will be suboptimal at a given step.

This algorithm is similar to the value iteration for POMDPs, but here, the policy trees must be retained because it is no longer possible to recover the policy from the value function. This can be seen by noticing the one-step lookahead in the POMDP case relies on calculating the belief state after an action is chosen. Because it is not possible to calculate a belief state in the DEC-POMDP case, and because the actions must be chosen based on local information, the optimal value function is not sufficient to generate a DEC-POMDP policy. Also, as noted above, the trees retained by each agent depend on the pruning conducted by the other agents. As a result, the generalized belief state changes after each agent's pruning step and pruning is conducted for each agent until changes are no longer made.

3.2.4 Policy iteration

This policy iteration approach [15] is similar to the finite-horizon dynamic programming algorithm, except finite-state controllers are used as policy representations. It is also similar the policy iteration approach for POMDPs discussed in Section 2.3.3. The algorithm can produce an ϵ-optimal in a finite number of steps.

Input: A joint controller, and a parameter ϵ.

1. Evaluate the joint controller by solving a system of linear equations.
2. Perform an exhaustive backup to add deterministic nodes to the local controllers.
3. Perform pruning on the controller.
4. If $\frac{\beta^{t+1}|R_{\max}|}{1-\beta} \leq \epsilon$, where t is the number of iterations so far, then terminate. Else go to step 1.

Output: A joint controller that is ϵ-optimal for all states.

Table 3.2. Policy Iteration for DEC-POMDPs.

The algorithm is initialized with an arbitrary joint controller. In the first part of an iteration, the controller is evaluated for all states of the system and current policies of the other agents via the solution of a system of linear equations. Next, an exhaustive backup is performed to add nodes to the local controllers. An exhaustive backup adds nodes to the local controllers for all agents at once. For each agent i, $|A_i||Q_i|^{|\Omega_i|}$ nodes are added to the local controller, one for each one-step policy. Thus, the joint controller grows by $|Q_c| \prod_i |A_i||Q_i|^{|O_i|}$ joint nodes. Note that repeated application of exhaustive backups amounts to a brute force search in the space of deterministic policies. This converges to optimality, but is obviously quite inefficient.

Pruning then takes place. Because an agent does not know which node of the controller any of the other agents will be in, pruning must be completed over the generalized belief space. That is, a node for an agent's controller can only be pruned if there is some combination of nodes that has higher value for all states of the system and at all nodes of the other agents' controllers. If this condition holds, edges to the removed node are then redirected to the dominating nodes. Because a node may be dominated by a distribution of other nodes, the resulting transitions may be stochastic rather than deterministic. The updated controller is evaluated and pruning continues until no agent can remove any further nodes. Due to discounting, this process will eventually converge to a solution within ϵ of optimal.

In contrast to the single agent case, there is no Bellman residual for testing convergence to ϵ-optimality. We resort to a simpler test for ϵ-optimality based on the discount rate and the number of iterations so far. Let $|R_{\max}|$ be the largest absolute value of an immediate reward possible in the DEC-POMDP. The algorithm terminates after iteration t if $\frac{\beta^{t+1}|R_{\max}|}{1-\beta} \leq \epsilon$. At this point, due to discounting, the value of any policy after step t is less than ϵ. The complete algorithm is sketched in Table 3.2.

Convergence is proved in [15] and rather than just pruning other *value-preserving transformations* can be performed. These ensure that the value is not lost for any node or state of system. Other than pruning, another example is to use linear programming to attempt to improve the value of the controller without increasing its size (termed DEC-BPI and described in 3.3.2).

3.2.5 Heuristic search approaches

Instead of computing the policy trees from the bottom up as is done by dynamic programming methods, they can also be built from the top down. This is the approach of multiagent A* (MAA*), which is an optimal algorithm built on heuristic search techniques [112]. The search is conducted by using an upper bound on the value of different policies (often POMDP or MDP solutions) and then choosing the policies in a best-first ordering. One step of the policy is then fixed and an upper bound on this partial policy is found. Again, the highest-valued policy is chosen and another action choice is fixed. This continues until a fully defined policy is found that has value that is higher than any of the partial policies.

More scalable versions of MAA* have also been proposed using the framework of Bayesian games to provide improved heuristics [75, 77]. The recent work on using a generalized version of MAA* with clustering [77] improves scalability by representing other agent policies more concisely without losing value. Because this thesis focuses on dynamic programming and controller-based approaches, the details of these methods is not discussed further.

3.3 Approximate solution methods

In this section, we discuss approximate dynamic programming methods for finite-horizon DEC-POMDPs and controller-based methods for infinite-horizon problems.

3.3.1 Approximate dynamic programming

The major limitation of dynamic programming approaches is the explosion of memory and time requirements as the horizon grows. This occurs because each step requires generating and evaluating all joint policy trees before performing the pruning step. Approximate dynamic programming techniques somewhat mitigate this problem by keeping a fixed number of local policy trees, using a parameter called MaxTrees, for each agent at each step [95]. This merges top down (heuristic search) and bottom up (dynamic programming) by using some heuristic to choose a top down policy for each agent up to a given horizon. The trees that have the highest value at the states that result from these heuristic policies are then found by using dynamic programming. This can be conducted in an iterative fashion. That is, for a problem of horizon T, heuristic policies can be used for the first $T-1$ steps and dynamic programming can find the best 1-step trees (actions) for the resulting states. Then, heuristic policies can be used for the first $T-2$ steps and the 1-step trees can be built up to 2 steps by dynamic programming. This can continue until horizon T trees have been constructed. Because only a fixed number of trees are retained at each step, this results in a suboptimal, but much more scalable algorithm.

A number of approaches are based on this idea. The first of which, *memory bounded dynamic programming* (MBDP), has linear space complexity, but still requires an exhaustive backup of policies at each step. To combat this drawback, [94, 23] suggested reducing the exponential role of local observations. This is accomplished by performing the exhaustive backup only over a small number of local observations. *Point-based incremental pruning* (PBIP) replaces the exhaustive backup with a branch-and-bound search in the space of joint policy trees [32].

3.3.2 Controller-based methods

The two approximate algorithms that can solve infinite-horizon discounted DEC-POMDPs are those of Bernstein et al. [17] and Szer and Charpillet [110]. Bernstein et al.'s approach, called bounded policy iteration for decentralized POMDPs (DEC-BPI), is a multiagent extension of Poupart and Boutilier's BPI algorithm [86]. Like BPI, it is an approximate algorithm that seeks to improve stochastic finite-state controllers. Szer and Charpillet describe an approach that searches for optimal fixed-size deterministic controllers.

DEC-BPI, like BPI for POMDPs, improves a set of fixed-size controllers by using linear programming. This is done by iterating through the nodes of each agent's controller and attempting to find an improvement. The algorithm and linear program are very similar to those of BPI except the optimization is done over not just the states of the problem, but also the nodes of the other agents' controllers (the generalized belief states). That is, the linear program searches for a probability distribution over actions and transitions into the agent's current controller that increases the value of the controller for any initial state and any initial node of the other agents' controllers. If an improvement is discovered, the node is updated based on the probability distributions found. Each node for each agent is examined in turn and the algorithm terminates when no agent can make any further improvements.

Szer and Charpillet have developed a best-first search algorithm that finds deterministic finite-state controllers of a fixed size. This is done by calculating an approximate value for the controller given the current known deterministic parameters and filling in the remaining parameters one at a time in a best-first fashion. The authors prove that this technique will find the optimal deterministic finite-state controller of a given size.

3.4 Additional modeling assumptions

DEC-POMDPs can be simplified by assuming that the state of the problem is fully observed by each agent, resulting in a multiagent MDP (MMDP) [21]. There has been work on efficient planning solutions for these problems (modeled as factored MDPs) [43, 44] as well as reinforcement learning [29, 58, 114] and Bayesian reinforcement learning [27].

One less restrictive assumption is that each agent can fully observe its own local state, but not the global state. This is the DEC-MDP model [1, 13], which has been shown to have more efficient solutions in some cases, but in general has the same complexity as the general model (NEXP-complete) [16]. If we consider problems in which the agents have independent transitions and observations and a structured reward model, the complexity drops to NP-complete [14]. A more efficient algorithm can be found in [82] and other modeling assumptions and the resulting complexity are studied in [42]. Recent work has shown that subclasses of DEC-POMDPs which have independent rewards, but dependent observations and transitions as well as those with certain structured actions retain the same complexity as the general problem [2].

Another way of simplifying DEC-POMDPs is to assume agents interact in a limited way based on locality. This assumption often considers agents that can only interact with their neighbors and thus is more scalable in the number of agents in the problem. This has been studied using the networked distributed POMDP (ND-POMDP) model for the finite-horizon case [61, 68, 73] and well as a more general factored model [76].

Communication and learning have also been studied in the DEC-POMDP model. While communication can be modeled using observations in the general model, it has been explicitly studied in several instances [12, 40, 41, 89, 92, 119, 120]. Recent work has also examined using communication for online planning in DEC-POMDPs [118]. Learning has been explored in the context of model free reinforcement learning [33, 80].

A generalization of DEC-POMDPs to the case where agents may have different rewards results in a partially observable stochastic game (POSG). Thus, if we assume POSGs with common payoffs for all agents, it is equivalent to the DEC-POMDP model. Algorithms have been developed for these common payoff POSGs as well [35, 74, 75]. Another game theoretic view of multiagent POMDPs from the Bayesian perspective is the I-POMDP model [39]. Also, by merging some ideas from game theory with those from optimization, a mixed integer linear programming approach has been developed [9].

CHAPTER 4

INCREMENTAL POLICY GENERATION: MORE EFFICIENT DYNAMIC PROGRAMMING FOR DEC-POMDPS

While the dynamic programming methods described in the previous section can provide optimal or ϵ-optimal solutions, their high resource usage results in limited scalability. To improve the efficiency of dynamic programming algorithms, we describe a new backup algorithm that is based on a reachability analysis of the state space. This method, which we call incremental policy generation, can be used to produce an optimal solution for the finite-horizon case or an ϵ-optimal solution for the infinite-horizon case. For finite-horizon problems, initial state information can be utilized to further increase scalability without sacrificing optimality. Our experiments show that planning horizon or the number of steps of dynamic programming can be increased due to a marked reduction in resource consumption. We discuss using incremental policy generation in approximate algorithms in Sections 7.2 and 7.3.

4.1 Overview

While several algorithms have been developed for DEC-POMDPs, both optimal and approximate algorithms suffer from a lack of scalability. As discussed in the previous chapter, many of these algorithms use dynamic programming to build up a set of possible policies from the last step until the first. This is accomplished by "backing up" the possible policies at each step and pruning those that have lower value for all states of the domain and all possible policies of the other agents. Unlike the POMDP case, the current dynamic programming used for DEC-POMDPs exhaustively generates all $t + 1$ step policies given the set of t step policies. This is very inefficient, often leading to a majority of policies being pruned after they are generated. Because many unnecessary policies are generated, resources are quickly exhausted, causing only very small problems and planning horizons to be solved optimally.

To combat this lack of scalability, we propose a more efficient dynamic programming method that generates policies based on a state space reachability analysis. This method generates policies incrementally for each agent based on those that are useful for each given action and observation. The action taken and observation seen may limit the possible next states of the system no matter what actions the other agents choose, allowing only policies that are useful for these possible states to be retained. This approach may allow a smaller number of policies to be generated, while maintaining optimality. Because solutions are built up for each action and observation, we call our method *incremental policy generation.* The incremental nature of our algorithm allows a larger number of dynamic programming steps to be completed and thus, can handle larger plan horizons or more backups in the infinite-horizon case.

Incremental policy generation is a general approach that can be applied to any DEC-POMDP algorithm that performs dynamic programming backups. These include optimal algorithms for finite [47, 20] and infinite-horizon [15] DEC-POMDPs as well as many approximate algorithms such as PBDP [111], indefinite-horizon dynamic programming [8] and all algorithms based on MBDP [95] (e.g. IMBDP [94], MBDP-OC [23] and PBIP [32]). All of these algorithms can be made more efficient by incorporating incremental policy generation.

4.2 Incremental policy generation

As mentioned above, one of the major bottlenecks of dynamic programming for DEC-POMDPs is the large resource usage of exhaustive backups. If policies for each agent can be generated without first exhaustively generating all next step possibilities, the algorithm would become more scalable. Incremental policy generation provides a method to accomplish this, while maintaining optimality.

Difference with POMDP methods

To improve the efficiency of dynamic programming for POMDPs, methods such as incremental pruning [24] have been developed, which incrementally generate and prune an agent's polices rather than use exhaustive backups. Unfortunately, these methods cannot be directly extended to DEC-POMDPs. This is because in the decentralized case, the state of the problem and the value of an agent's policy depend on the policies of all agents. Thus, pruning any agent's $(t+1)$-step policy requires knowing all $(t+1)$-step policies for the other agents. Therefore, all other agents would need complete exhaustive backups before one agent could perform incremental pruning. The savings in this approach would be minimal, leading us to consider other approaches.

DEC-POMDP approach

To solve this problem, we build up the policies for each agent by using a one-step reachability analysis. This allows all other agent policies to be considered, but without actually constructing them exhaustively and only evaluating them at reachable states. After an agent chooses an action and sees an observation, the agent may not know the state of the world, but it can determine which states are possible. For instance, assume an agent has an observation in which a wall is perfectly observed when it is to the left or right. If the agent sees the wall on the right, it may not know the exact state it is in, but it can limit the possibilities to those states with walls on the right. Likewise, in the commonly used two agent tiger domain [72], after an agent opens a door, the problem transitions back to the start state. Thus, after an open action is performed, the agent knows the exact state of the world.

So, how can we use this information? One way is to limit the policies generated during dynamic programming. We can first determine which states are possible after an agent chooses an action and sees an observation. Only subtrees that are useful for the resulting states need to be considered after that observation is seen when constructing trees that begin with the given action. To maintain optimality, we consider any first action for the other agents and then prune using the known possible subtrees for the other agents. This is described more formally below.

Limiting the state

Any agent can calculate the possible states that result from taking an action and seeing an observation. To determine the next state exactly, the agent would generally need to know the probability that the system is currently in any given state and that the other agents will choose each action. Since we do not assume the agents possess this information, the exact state can only be known in some circumstances, but the set of possible next states can often be limited.

For instance, the probability of a resulting state s' after agent i chooses action a_i and observes o_i is determined by:

$$P(s'|a_i, o_i) = \frac{\sum_{\vec{a}_{-i}, \vec{o}_{-i}, s} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a})P(\vec{a}_{-i}, s|a_i)}{P(o_i|a_i)} \tag{4.1}$$

with $P(\vec{a}_{-i}, s|a_i)$ representing the probability of the current state and that other agents choose actions $\vec{a}_{-i}$. The normalizing factor is:

$$P(o_i|a_i) = \sum_{\vec{a}_{-i}, \vec{o}_{-i}, s, s'} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a})P(\vec{a}_{-i}, s|a_i).$$

To determine all possible next states for a given action and observation, we can assume $P(\vec{a}_{-i}, s|a_i)$ is a uniform distribution and retain any state s' that has a positive probability. We will call the set of possible successor states S'.

Algorithm 1: Incremental policy generation: Finite-horizon

```
input  : A set of $t$-step policy trees for all agents, $Q$, and an action $a$
output: A set of $(t+1)$-step policy trees for a given agent, $\hat{Q}_i$
begin
    for each action, $a$ do
        for each observation, $o$ do
            $S' \leftarrow possibleStates(a, o)$
            $Q_i^{a,o} \leftarrow usefulTrees(S', Q)$
        $\hat{Q}_i^a \leftarrow \oplus_o Q_i^{a,o}$
    $\hat{Q}_i \leftarrow \cup_a Q_i^a$
    return $\hat{Q}_i$
end
```

Knowing the exact state

When $P(o_i|s, \vec{a}, s')P(s'|\vec{a}, s)$ is constant for the given a_i and o_i, then it does not depend on the state or other agents' actions. Thus, $\sum_{\vec{a}_{-i},s} P(\vec{a}_{-i}, s|a_i) = 1$ and the exact state is known by:

$$P(s'|a_i, o_i) = \frac{P(o_i|s, \vec{a}, s')P(s'|s, \vec{a})}{\sum_{s'} P(o_i|s, \vec{a}, s')P(s'|s, \vec{a})}$$

4.3 Algorithm

We first present the finite-horizon algorithm and then discuss how this can be extended to the infinite-horizon case.

4.3.1 Finite-horizon case

Our approach is summarized in Algorithm 1. For each agent, i, we assume we begin with a set of horizon t trees Q_i. We can create a set of horizon $t+1$ trees that has each action from the domain as a first action. Assuming we start with action a, we must decide which subtrees to choose after seeing each observation. Rather than adding all possibilities from Q_i as is done in the exhaustive approach, we only add trees from Q_i that are useful for some possible successor state s' and set of trees of the other agents. That is, agent i's tree q_i is retained if the value of choosing q_i is higher than choosing any other tree for some distribution of other agent trees, q_{-i} and next states of the system s'. This is formulated as:

$$\sum_{q_{-i},s'} x(q_{-i},s')V(q_i,q_{-i},s') > \sum_{q_{-i},s'} x(q_{-i},s')V(\hat{q}_i,q_{-i},s') \quad \forall\ \hat{q}_i \tag{4.2}$$

Where $x(q_{-i},s')$ is a distribution over trees of the other agents and successor states, $q_i, \hat{q}_i \in Q_i$, $q_{-i} \in Q_{-i}$ and $s' \in S'$ is as described in the previous section.

The solution to this formulation can be found by using a linear program similar to the one in Table 3.1 for the previous dynamic programming algorithm. But the new linear program has fewer variables and constraints, therefore making it faster to solve. This is because it uses the smaller set of trees from step t rather than step $t+1$ and $S' \subseteq S$. The method limits the possible subtrees after taking action a and seeing observation o to those that are useful for some distribution over successor states and other agent subtrees. The set of trees that we retain for the given action and observation are t-step trees for agent i, which we will call $Q_i^{a,o}$, leaving off the i for the action and observation to simplify the notation.

After generating all possible sets of trees for each observation with a fixed action, we then build the horizon $t+1$ trees for that action. This is accomplished by creating all possible trees that begin with the fixed action, a, followed by choosing any tree from the set $Q_i^{a,o}$ after o has been observed. The resulting number of trees for agent i at this step is given by $\prod_o |Q_i^{a,o}|$. In contrast, exhaustive generation of trees would produce $|Q_i|^{|\Omega_i|}$ trees. Once all trees for each action have been generated, we take the union of the sets for each action to produce the set of horizon $t+1$ trees for the agent. When we have the $(t+1)$-step trees for all the agents, we are then able to evaluate them and prune those that have lesser value for all initial states and trees of the other agents. This pruning step is exactly the same as that used in Hansen et al.'s dynamic programming algorithm and may further reduce the number of trees retained.

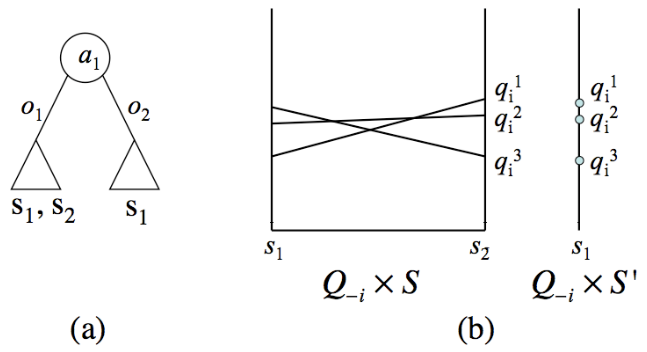


Figure 4.1. Example of (a) limiting states after actions are chosen and observations seen and (b) pruning with a reduced state set, S'.

Example

An illustration of the incremental policy generation approach is given in Figure 4.1. The algorithm (a) first determines which states are possible after a given action has been taken and each observation has been seen. After action a_1 is taken and o_1 is seen, states s_1 and s_2 are both possible, but after o_2 is seen, only s_1 is possible. Then, (b) the values of the trees from the previous step (t) are determined for each resulting state and each set of trees of the other agents.

The dimension of the value vectors of the trees is the same as the number of possible policies for the other agents times the number of states in the system, or $Q_{-i} \times S$. To clarify the illustration, we assume there is only one possible policy for the other agents and two states in the domain. As seen in the figure, when both states are considered, all trees for agent i are useful (not dominated) for some possible distribution. When only state 1 is possible, only q_i^1 is useful. This allows both q_i^2 and q_i^3 to be pruned for this combination of a_1 and o_2, reducing the number of possibilities to 1. Because there are still 3 useful subtrees for the combination of a_1 and o_1, the total number of possible trees starting with action a_1 is 3. This can be contrasted with the 9 trees that are possible using exhaustive generation.

Algorithm 2: Incremental policy generation: Infinite-horizon

```
input  : A set controllers for all agents, $C$ and an agent index, $i$
output: A backed up controller for the given agent, $C'_i$
begin
    for each action, $a$ do
        for each observation, $o$ do
            $S' \leftarrow possibleStates(a, o)$
            $C_i^{a,o} \leftarrow usefulCont(S', C)$
        $C_i^a \leftarrow \oplus_o C_i^{a,o}$
    $C'_i \leftarrow \cup_a C_i^a$
    return $C'_i$
end
```

4.3.2 Infinite-horizon case

Our infinite-horizon approach is summarized in Algorithm 2. Note that is is very similar to the finite-horizon algorithm except finite-state controllers are used instead of policy trees. For each agent, i, we first assume we have a controller C_i. We can create a backed up controller, C'_i, by first choosing an action. Assuming we start with action a we need to decide which node in C to choose after seeing each observation. Rather than adding all possibilities from C_i as is done in the exhaustive approach, we only transition to nodes that are useful for some distribution over possible successor states s' and initial nodes of the other agents. That is, agent i's node q_i is retained if the value of choosing q_i is higher than choosing any other node for some distribution of other agent nodes, q_{-i} and next states of the system s'. This in the same way as the finite-horizon case given in Equation 4.2 where $x(q_{-i}, s')$ is now a distribution over nodes of the other agents and successor states, $q_i, \hat{q}_i \in C_i$, $q_{-i} \in C_{-i}$ and $s' \in S'$. Thus, we limit the possibilities after taking action a and seeing observation o to those nodes that are useful for some such distribution. The set of nodes that we retain for the given action and observation for agent i we will call $C_i^{a,o}$.

Like the finite-horizon case, after generating all useful nodes for each observation with a fixed action, we can conduct a backup for that action. This is accomplished by creating all possible nodes which begin with the fixed action, a, followed by choosing any node from the set $C_i^{a,o}$ after o has been observed. The resulting number of nodes for agent i at this step is $\prod_o |C_i^{a,o}|$. In contrast, exhaustive generation would produce $|C_i|^{|\Omega|}$ nodes. Once all nodes for each action have been generated, we take the union of the sets for each action to get the backed up controller for the agent. We are then able to evaluate the backed up controllers and prune dominated nodes in the same way as described for policy iteration. Pruning may further reduce the number of nodes retained and increase their value.

4.4 Analysis

We first present the proof of optimality in the finite-horizon case and then provide the proof of ϵ-optimality in the infinite-horizon case.

4.4.1 Finite-horizon case

We first show that the incremental policy generation approach does not prune any t-step subtree of Q_i that would be part of an undominated $(t+1)$-step policy tree.

Lemma 4.4.1. *Any t-step tree that is not added by incremental policy generation is part of a dominated $(t+1)$-step tree.*

Proof. We will prove this for two agents, but this proof can easily be generalized to any number of agents. We show from agent 1's perspective that if tree q_1 has a subtree which is not included by our incremental policy generation algorithm, then q_1 must be dominated by some distribution of trees $\hat{q}_1$. This is determined by

$$V(q_1, q_2, s) \leq \sum_{\hat{q}_1} x(\hat{q}_1) V(\hat{q}_1, q_2, s) \ \forall q_2, s$$

This proof is based on the fact that in incremental policy generation, all reachable next states and all possible next step policies for the other agents are considered before removing a policy.

Consider the set of trees generated for agent 1 by starting with action a. If observation o_1^j was seen, where $q_1^{o_1^j}$ was not included in set $Q_1^{a_{q_1}, o_1^j}$, then we show that q_1 must be dominated by the set of trees that are identical to q_1 except instead of choosing $q_1^{o_1^j}$, some distribution over trees in $Q_1^{a_{q_1}, o_1^j}$ is chosen. We abbreviate $q_1^{o_1^j}$ by q_1' to simplify notation.

Because q_1' was not included in $Q_1^{a_{q_1}, o_1^j}$ we know there is some distribution of subtrees that dominates it for all subtrees of the other agent and at the subset of states that are possible after choosing action a_{q_1} and observing o_1.

$$V(q_1', q_2', s') \leq \sum_{\hat{q}_1'} x(\hat{q}_1') V(\hat{q}_1', q_2', s') \ \forall \ q_2', s' \tag{4.3}$$

Given this distribution $x(\hat{q}_1')$ we can create a probability distribution over subtrees that chooses the trees of $\hat{q}_1'$ when o_1^j is seen, but otherwise chooses the same subtrees that are used by q_1. The value of this tree is given by

$$R(a_{q_1}, a_{q_2}, s) + \sum_{s'} P(s'|a_{q_1}, a_{q_2}, s) \sum_{o_1, o_2} P(o_1, o_2|a_{q_1}, a_{q_2}, s') \sum_{\hat{q}_1'} x(\hat{q}_1', o_1) V(q_1^{o_1}, q_2^{o_2}, s')$$

Because the trees are otherwise the same, from the inequality in Equation 4.4, we know that

$$R(a_{q_1}, a_{q_2}, s) + \sum_{s'} P(s'|a_{q_1}, a_{q_2}, s) \sum_{o_1, o_2} P(o_1, o_2|a_{q_1}, a_{q_2}, s') V(q_1^{o_1}, q_2^{o_2}, s') \geq$$

$$R(a_{q_1}, a_{q_2}, s) + \sum_{s'} P(s'|a_{q_1}, a_{q_2}, s) \sum_{o_1, o_2} P(o_1, o_2|a_{q_1}, a_{q_2}, s') \sum_{\hat{q}_1'} x(\hat{q}_1', o_1) V(q_1^{o_1}, q_2^{o_2}, s')$$

This holds for any s and q_2 because Equation 4.4 holds for any initial state or possible policy tree of the other agent.

This can also be viewed as a distribution of trees, $x(\hat{q}_1)$, one for each different subtree with positive probability in $x(\hat{q}_1')$. Thus, the value of the original tree q_1 is less than or equal to the value of the distribution of trees for all q_2 and s.

$$R(a_{q_1}, a_{q_2}, s) + \sum_{s'} P(s'|a_{q_1}, a_{q_2}, s) \sum_{o_1,o_2} P(o_1, o_2|a_{q_1}, a_{q_2}, s')V(q_1^{o_1}, q_2^{o_2}, s') \leq$$

$$\sum_{\hat{q}_1} x(\hat{q}_1) \left[R(a_{\hat{q}_1}, a_{q_2}, s) + \sum_{s'} P(s'|a_{\hat{q}_1}, a_{q_2}, s) \sum_{o_1,o_2} P(o_1, o_2|a_{\hat{q}_1}, a_{q_2}, s')V(\hat{q}_1^{o_1}, q_2^{o_2}, s') \right]$$

or

$$V(q_1, q_2, s) \leq \sum_{\hat{q}_1} x(\hat{q}_1)V(\hat{q}_1, q_2, s) \ \forall q_2, s$$

□

Theorem 4.4.2. *Incremental policy generation returns an optimal solution for any initial state.*

Proof. This proof follows from Hansen et al.'s proof and Lemma 4.4.1. Dynamic programming with pruning generates and retains all optimal policies for any given horizon and initial state. Since Lemma 4.4.1 shows that incremental policy generation will not remove any policies that would not be pruned, our approach also retains all optimal policies. Thus, by choosing the highest valued policy for any horizon and initial state, our incremental policy generation algorithm provides the optimal finite-horizon solution. □

4.4.2 Infinite-horizon case

We first show that using incremental policy generation, only nodes that would be pruned after exhaustive generation are not added to the set of possible next nodes, $C_i^{a,o}$. This is very similar to the finite-horizon proof, but controllers are used in place of trees.

Lemma 4.4.3. *Any node that is is not added by the incremental pruning algorithm is dominated by some distribution of backed up nodes.*

Proof. We will show that if node p transitions to some node in the previous controller which is not included by the incremental policy generation algorithm, then p must be dominated by some distribution of nodes $\hat{p}$. For two agents, this is determined by:

$$V(p,q,s) \leq \sum_{\hat{p}} x(\hat{p})V(\hat{p},q,s)\ \forall q,s$$

Intuitively, this proof is based on the fact that all reachable next states and all possible next step policies for the other agents are considered before removing a policy. We will prove this for two agents, but this proof can easily be generalized to any number of agents.

Consider the set of nodes generated for agent 1 by starting with action a. If observation, o_1^j was seen, where $p_{o_1^j}$ was not included in our set C^{a_p,o_1^j}, then we show that p must be dominated by the set of nodes that are identical to p except instead of choosing $p_{o_1^j}$, some distribution over nodes in C^{a_p,o_1^j} is transitioned to instead. We will abbreviate $p_{o_1^j}$ by p' to simplify the notation.

Because p' was not included in C^{a_p,o_1^j} we know there is some distribution of nodes in C_1that dominates it for all nodes in C_2 and at the subset of states that are possible after choosing action a_p and observing o_1.

$$V(p',q',s') \leq \sum_{\hat{p}'} x(\hat{p}')V(\hat{p}',q',s')\ \forall\ q',s' \tag{4.4}$$

Given this distribution $x(\hat{p}')$ we can create a probability distribution over nodes which chooses the nodes of $\hat{p}'$ when o_1^j is seen, but otherwise transitions to the same nodes that are used by p. The value of this policy is given by

$$R(a_p,a_q,s) + \gamma \sum_{s'} P(s'|a_p,a_q,s) \sum_{o_1,o_2} P(o_1,o_2|a_p,a_q,s') \sum_{\hat{p}'} x(\hat{p}',o_1)V(p_{o_1},q_{o_2},s')$$

Because the nodes are otherwise the same, from the inequality in Equation 4.4, we know that

$$R(a_p, a_q, s) + \gamma \sum_{s'} P(s'|a_p, a_q, s) \sum_{o_1,o_2} P(o_1, o_2|a_p, a_q, s')V(p_{o_1}, q_{o_2}, s') \geq$$

$$R(a_p, a_q, s) + \gamma \sum_{s'} P(s'|a_p, a_q, s) \sum_{o_1,o_2} P(o_1, o_2|a_p, a_q, s') \sum_{\hat{p}'} x(\hat{p}', o_1)V(p_{o_1}, q_{o_2}, s')$$

This holds for for any s and q because Equation 4.4 holds for any initial state or node of the other agent.

This can also be viewed as a distribution of nodes, $x(\hat{p})$, one for each different node that is transitioned to with positive probability in $x(\hat{p}')$. Thus, the value of the original node p is less than or equal to the value of the distribution of nodes for all q and s. That is,

$$R(a_p, a_q, s) + \gamma \sum_{s'} P(s'|a_p, a_q, s) \sum_{o_1,o_2} P(o_1, o_2|a_p, a_q, s')V(p_{o_1}, q_{o_2}, s') \leq$$

$$\sum_{\hat{p}} x(\hat{p}) \left[R(a_{\hat{p}}, a_q, s) + \gamma \sum_{s'} P(s'|a_{\hat{p}}, a_q, s) \sum_{o_1,o_2} P(o_1, o_2|a_{\hat{p}}, a_q, s')V(\hat{p}_{o_1}, q_{o_2}, s') \right]$$

or

$$V(p, q, s) \leq \sum_{\hat{p}} x(\hat{p})V(\hat{p}, q, s)\ \forall q, s$$

□

Theorem 4.4.4. *The incremental policy generation algorithm returns an ϵ-optimal infinite-horizon solution for any initial state after a finite number of steps.*

Proof. This proof follows from Theorem 2 in [15], which states that policy iteration converges to an ϵ-optimal controller. This is due to the use of a discount factor, which ensures that after some finite number of steps, the value of a policy will not change more than some ϵ. Thus, if enough steps of dynamic programming are performed and all policies that could contribute to the optimal policy are retained, some set of initial nodes will approach the optimal value. The pruning steps are shown to not harm this convergence and Lemma 4.4.3 shows that nodes that are not generated by incremental policy generation would by pruned in policy iteration. Thus, the incremental policy generation algorithm can also provide ϵ-optimal infinite-horizon policies in a finite number of steps. $\square$

4.5 Incorporating Start State Information

Finite-horizon dynamic programming creates a set of trees for each agent that contains an optimal tuple of trees for any initial state distribution. This results in many trees being generated that are suboptimal for a known initial distribution. To combat this, start state information can be incorporated into the algorithm. We discuss an optimal method of including this information.

Optimal algorithm

As already mentioned, any agent i can determine its next possible states $P(s'|a_i, o_i)$ after taking an action a_i and perceiving an observation o_i. This allows the agent to focus its policy generation for the next step over those states only. Although this approach is more scalable than previous dynamic programming methods, further scalability can be achieved by incorporating information from a known start state.

It is well known that the reachable part of the state space can be limited by considering actual agent action sequences (histories) and the initial state information at hand [111]. However, the only attempt, suggested by Szer and Charpillet that exploits reachability by taking into account the other agents' histories leads to a worst-case complexity that is doubly exponential. We take a different approach, as we only want to determine the possible next states for each agent rather than constructing all possible trees of the desired horizon, T. Our approach generates possible states for any number of steps, $T-t$, without explicitly generating all horizon T policy trees.

Consider the case when we have already generated the policy trees of height $T-1$ for all agents. We can then use the start state as the only state in S to better limit S' in our incremental policy generation approach. That is, we know that we will begin in b_0, transition to some state s', and choose some height $T-1$ tree. Thus, in Equation 4.1, $P(\vec{a}_{-i}, s|a_i)$ becomes $b_0(s)P(\vec{a}_{-i}, |a_i)$ and requires using only the known initial state b_0 in addition to the unknown probability of action for the other agents. Then, by considering each initial action of the other agents, we can generate a smaller set of possible next states.

This idea can be extended to any step of dynamic programming. When we have trees of height t, with $t < T-1$ then we must consider all possible action sequences, or histories, of the agents rather than just single actions. We define a k-step history for agent i as $h_i^k = (a_i^1, o_i^1, \cdots, o_i^k, a_i^k)$. The probability of state s^{k+1} resulting after k steps of agent i's history can be defined recursively as:

$$P(s^{k+1}|h_i^k) \propto \sum_{\vec{a}_{-i}, \vec{o}_{-i}, s^k} P(o_i^k, \vec{o}_{-i}|s^k, a_i^k, \vec{a}_{-i}, s^{k+1})P(s^{k+1}|s^k, \vec{a})P(s^k|h_i^{k-1})$$

where a_i^k and o_i^k are the action taken and observation seen by agent i at the kth step of history h_i^k. The value can be normalized to give the true probability of each state after k steps and for $k=0$, $P(s) = b_0(s)$, the initial state probability.

This approach limits the set of reachable states, S', for agent i given both the initial state information and agent history. This is done for all possible histories of the agent and for all agents. Each agent's policy generation process then follows as described for incremental policy generation. Because the set of all possible histories of any agent is exponential with respect to the length of the histories, this method becomes impractical when $T-t$ is large. To combat this, start state information could be used only when $T - t$ is small and repeated state distributions can be ignored at each level.

Corollary 4.5.1. *Our incremental policy generation algorithm with start state information returns an optimal finite-horizon solution for the given initial state distribution.*

Proof. Because we generate all possible next states for each agent using all possible histories for all agents and the given initial state information, this corollary follows from Theorem 4.4.2 and Theorem 1 of [111]. $\square$

4.6 Experiments

We first provide results comparing incremental policy generation with optimal finite-horizon algorithms and then give results for the infinite-horizon case. The incremental policy generation results also make use of *observation compression*, which merges observations that provide the same information. For instance in the tiger problem after opening a door, each observation is possible, but they are chosen randomly. These observations provide the same information and thus, can be merged. More formally, two of agent i's observations o_i^1 and o_i^2 are the same for a given action a_i if for each set of other agent actions a_{-i}, s and s' $P(o_i^1|s, a_i, a_{-i}, s') = P(o_i^2|s, a_i, a_{-i}, s')$.

4.6.1 Finite-horizon case

We first compare incremental policy generation with other state-of-the-art finite-horizon algorithms. These are dynamic programming for POSGs [47] and the leading top-down approaches GMAA* [75] and clustering GMAA* [77], abbreviated as C-GMAA*. Each of the top-down approaches used the MDP heuristic. These algorithms were tested on the common Box Pushing problem [94], the Stochastic Mars Rover problem [8] and a new version of the Meeting in a Grid problem [17]. This version has a grid that is 3 by 3 rather than 2 by 2, the agents can observe walls in four directions

Hor.	DP for POSGs	Incremental Pol. Gen. (IPG)	IPG with Start State	GMAA*	C-GMAA*	Value
Meeting in a 3x3 Grid, $\lvert S\rvert = 81$, $\lvert A_i\rvert = 5$, $\lvert\Omega_i\rvert = 9$						
2	(5) 5 in 5s	5 in <1s	5 in 5s	x	9 <1s	0.000
3	x	5 in 16s	5 in 17s	x	121 <1s	0.133
4	x	40 in 42s	10 in 53s	x	x	0.433
5	x	(25960)* in 2555s	(148) 148,145 in 600s	x	x	0.896
Box Pushing, $\lvert S\rvert = 100$, $\lvert A_i\rvert = 4$, $\lvert\Omega_i\rvert = 5$						
2	(128) 8 in 14s	8 in 2s	(4) 2,3 in 1s	25 in $< 1s$	4 in $< 1s$	17.60
3	x	(320,256) 256 in 1159s	(6) 5,6 in 6s	x	25 in 5s	66.08
4	x	x	(233,239) 233 in 1138s	x	x	98.59
Stochastic Mars Rover, $\lvert S\rvert = 256$, $\lvert A_i\rvert = 6$, $\lvert\Omega_i\rvert = 8$						
2	x	(150, 672)* in 72s	(16,20) 12,15 in 83s	x	1 <1s	5.80
3	x	x	(396, 534)* in 389s	x	4 <1s	9.38
4	x	x	x	x	11.11 in 103s	10.18

Table 4.1. Size, running time and value produced for each horizon on the test domains. For dynamic programming algorithms the size is given as the number of of policy trees before and after pruning (if different) and only one number is shown if both agent trees are the same size. For top-down approaches the size of the final Bayesian game is provided.

rather than two and they must meet in the top left or bottom right corner rather than in any square. The resulting problem has 81 states, 5 actions and 9 observations. We chose these domains to explore the ability of the algorithms to solve large problems. More details on these domains is provided in the appendix.

The dynamic programming algorithms were run with up to 2GB of memory for up to 2 hours. We record the number of trees generated by the backup before and after pruning as well as the running time of the algorithm. When the available resources were exhausted during the pruning step, this is denoted with *. The incremental policy generation (IPG) approach using start state information only used the start state for steps when $T - t \leq T/2$ for current tree height t and horizon T. The top-down algorithm results for the Box Pushing domain are taken from [77] and the other results are provided courtesy of Matthijs Spaan. Because search is used in top-down approaches rather than generating multiple trees, the average size of the Bayesian game used to calculate the heuristic on the final step is provided along with the running time. Running times may vary as these algorithms were run on a different machine, but we expect the trends to remain consistent.

The results in Table 4.1 show that on these problems, incremental policy generation is always more scalable than the previous dynamic programming approach and generally more scalable than the leading top-down approach, C-GMAA*. Dynamic programming (DP) for POSGs can only provide a solution of horizon two for the first two problems and horizon one for the larger Mars Rover problem. GMAA* is similarly limited because the search space quickly becomes intractable as the horizon grows. Because the structure of these problems permits clustering of agent policies, C-GMAA* runs very quickly and is able to improve scalability over GMAA*, especially in the Mars Rover problem. In the other domains, IPG with start state information can reach larger horizons than the other approaches. There is a small amount of overhead by using start state information, but this approach is generally faster and more scalable than the other dynamic programming methods because fewer trees are retained at each step. IPG without start state information is similarly faster and more scalable than the previous dynamic programming algorithm because it is also able to retain fewer trees. These results show that incorporating incremental policy generation greatly improves the performance of dynamic programming, allowing it outperform the leading top-down approach on two of the three test problems.

4.6.2 Infinite-horizon case

We also present results comparing policy iteration with incremental policy generation on a set of infinite-horizon problems. These problems are the two agent tiger [72], the recycling robot [3] and the box pushing [94] domains.

The results are shown in Table 4.2. Each algorithm was run until the given memory (2GB) or time (4 hours) was exhausted and the number of nodes in each agent's controller, the running time and the resulting values are provided. If the number of nodes generated was greater than the number retained after pruning was completed, the size after generation is given in parentheses. If the given number of iterations could not be completed, an 'x' is displayed. The initial policies for each algorithm consisted of each agent repeating the 'open left' action in the tiger problem, the 'pick up large can' action in the recycling robots problem and the 'turn left' action in the box pushing problem.

It can be seen that the incremental policy generation approach requires less time on each dynamic programming iteration and allows more iterations to be completed. This results in higher solution values for the incremental approach. The results also show that the incremental approach often generates the exact same controllers without the need for pruning. That is, while policy iteration produces a large number of nodes and then prunes many of them, the incremental approach arrives at the same controller without the added memory of generating an exhaustive set of nodes or the

Iteration	Policy Iteration	Incremental Generation	Value
Two Agent Tiger, $\lvert S\rvert = 2$, $\lvert A_i\rvert = 3$, $\lvert\Omega_i\rvert = 2$			
0	1 in $<$ 1s	1 in $<$1s	-150.00
1	3 in $<$1s	3 in $<$1s	-137.00
2	(27) 15 in 1s	15 $<$1s	-117.85
3	(675) 255 in 465s	255 in 70s	-98.90
4	x	(65535)* in 3057s	-90.54
Recycling Robots, $\lvert S\rvert = 4$, $\lvert A_i\rvert = 3$, $\lvert\Omega_i\rvert = 2$			
0	1 in $<$ 1s	1 in $<$ 1s	0
1	3 in $<$ 1s	3 in $<$ 1s	3.01
2	(27) 6 in 1s	(8) 6 in $<$ 1s	25.66
3	(108) 20 in 6s	20 in 1s	26.04
4	(1200) 72 in 542s	72 in 6s	26.69
5	(15552)* in 869s	272 in 201s	27.20
6	x	(1092) 1058 in 5016s	27.67
7	x	(4759)* in 5918s	28.10
Box Pushing, $\lvert S\rvert = 100$, $\lvert A_i\rvert = 4$, $\lvert\Omega_i\rvert = 5$			
0	1 in $<$ 1s	1 in $<$ 1s	-2.00
1	(4) 2 in 2s	2 in 1s	-2.00
2	(128) 9 in 209s	9 in 7s	12.84
3	x	(137,137) 131,128 in 4501s	67.87

Table 4.2. Infinite-horizon results of applying policy iteration and incremental policy generation (IPG). The number of nodes generated is shown in parentheses beside the number retained after pruning and iterations for which pruning could not be completed with the given amount of time and memory are marked with *.

extra time needed for pruning these nodes. Even when nodes are removed in the incremental case, only a small number need to be pruned.

4.7 Discussion

In this chapter, we introduced the incremental policy generation approach, which is a more efficient way to perform dynamic programming backups in DEC-POMDPs. This is achieved by using state information from each possible action taken and observation seen to reduce the number of trees or controller nodes considered at each step. We proved that this approach can be used to provide an optimal finite-horizon or infinite-horizon solution and showed that this results in an algorithm that is faster and can scale to larger horizons than the current dynamic programming approaches. We also showed that in two of three finite-horizon test domains, it solves problems with larger horizon than the leading optimal top-down approach, clustering GMAA*.

Incremental policy generation is a very general approach that can improve the efficiency of any DEC-POMDP algorithm that uses dynamic programming. To test this generality, we also incorporated our approach into the leading approximate algorithm, PBIP and developed an approximate dynamic programming approach for infinite-horizon DEC-POMDPs. These approaches are described in Sections 7.2 and 7.3. As many other algorithms use dynamic programming, incremental policy generation can increase scalability in a large number of algorithms.

Because incremental policy generation uses state information from the actions and observations, it should work well when a small number of states are possible for each action and observation. In contrast, clustering GMAA* uses the value of agent policies to cluster action and observation histories. Since these approaches use different forms of problem structure, it may be possible to combine them either by producing more focused histories when making use of start state information or better heuristic policies for use with MBDP-based approaches. Other work has also been done to compress policies rather than agent histories, improving the efficiency of the linear program used for pruning [20]. By also incorporating incremental policy generation, this combination of techniques could be applied to further scale up dynamic programming algorithms.

CHAPTER 5
OPTIMIZING FIXED-SIZE CONTROLLERS

In this chapter, we discuss modeling and solving fixed-size solution representations for infinite-horizon POMDPs and DEC-POMDPs using nonlinear optimization. This allows these problems to be solved by utilizing powerful and well-studied nonlinear programming approaches. We first give an overview of our approach and discuss some relevant work. We then provide the nonlinear programming representations and proof that they represent optimal fixed-sized solutions. We discuss optimal and approximate methods for solving the representation and finally present experimental results comparing approximate solutions with other state-of-the-art algorithms. We demonstrate that our approach can outperform POMDP approaches on certain problem types and can significantly outperform previous DEC-POMDP approaches on a range of problems. We discuss improvements to this representation in the following chapter.

5.1 Overview

Due to the high complexity of finding solutions to infinite-horizon POMDPs and DEC-POMDPs, using fixed-size solutions is an appealing choice. In particular, using finite-state controllers to model solutions has been shown to be effective for both POMDPs [70, 86] and DEC-POMDPs [17, 110]. This approach facilitates scalability as it offers a tradeoff between solution quality and the available resources. That is, for a given amount of memory, a controller can be found that is optimal for that size. Large controllers may be needed to provide a near optimal solution in some problems, but our experiments suggest that smaller controllers produce high quality solutions in a wide array of problems.

Unlike other controller-based approaches for POMDPs and DEC-POMDPs, our formulation defines an optimal controller for a given size. This is accomplished by formulating the problem as a nonlinear program (NLP), and exploiting existing nonlinear optimization techniques to solve it. Nonlinear programming is an active field of research that has produced a wide range of techniques that can efficiently solve a variety of large problems [18]. In the POMDP case, parameters are optimized for a fixed-size controller which produces the policy for that problem. In the DEC-POMDP case, a set of fixed-size independent controllers is optimized, which when combined, produce the policy for the DEC-POMDP. We present an overview of how to solve these problems optimally, but as this would often be intractable in practice, we also evaluate an effective approximation technique using standard NLP solvers.

One premise of our work is that an optimal formulation of the problem facilitates the design of solution techniques that can overcome the limitations of previous controller-based algorithms and produce better stochastic controllers. The general nature of our formulation allows a wide range of solution methods to be used. This results in a search that is more sophisticated than those previously used in controller-based methods. Our approach also provides a framework for which future algorithms can be developed.

In order to increase the solution quality of our memory-bounded technique for DEC-POMDPs, we examine the benefits of introducing a *correlation device*, which is a shared source of randomness. This allows a set of independent controllers to be correlated in order to produce higher values, without sharing any additional local information. Correlation adds another mechanism in an effort to gain the most possible value with a fixed amount of space. This has been shown to be useful in order to increase value of fixed-size controllers [17] and we show that is also useful when combined with our NLP approach.

5.2 Disadvantages of previous algorithms

We first discuss the shortcomings of optimal algorithms and then approximate POMDP and DEC-POMDP algorithms. An overview of the POMDP approaches is presented in Sections 2.3 and 2.4, while the DEC-POMDP approaches are reviewed in Sections 3.2 and 3.3.

The optimal policy iteration algorithms for POMDPs and DEC-POMDPs are able to converge to ϵ-optimal solutions in a finite number of steps, but this number is often impractically large. These methods often exhaust available memory after only a few steps due to the exhaustive backups that are performed. In the POMDP case, many unnecessary nodes are retained at each step because the algorithm cannot take advantage of an initial state distribution and must attempt to improve the controller for any initial state. Thus, nodes that are useful for states that will never be visited are not pruned and continually increase the size of the agent's controller. This phenomenon is exacerbated in the DEC-POMDP case by the fact that nodes must be retained if they are useful for any possible set of initial nodes of the other agents as well as any system state, resulting in many nodes being retained that will not be part of an optimal solution given the initial system state. Even if many nodes can be removed at each step, a number that is exponential in the previous controller size must be generated and tested, which is intractable for even moderate controller sizes. Thus, the lack of adequate pruning and the need for exhaustive backups renders the policy iteration algorithms applicable to only small problems. Incorporating incremental policy generation from the previous chapter will increase the efficiency, but for many problems an ϵ-optimal solution will still not be able to be found.

5.2.1 POMDP approximate methods

The linear program used by bounded policy iteration (BPI) may allow for controller improvement, but local maxima are likely to be found since it performs only a one step lookahead while keeping the controller values fixed. Also, because the improvement must be over all states it can be difficult to find parameters that meet this requirement. Adding the heuristics used in biased BPI reduce this necessity by optimizing over weighted states as well as allowing value to be lost. The disadvantage of biased BPI is that the heuristics are not useful in all problems and for cases when they are, domain knowledge or a large amount of experimentation may be necessary to properly set them.

Gradient ascent (GA) has similar problems with local maxima as well as other concerns. Meuleau et al. must construct a cross-product MDP from the controller and the underlying POMDP in a complex procedure to calculate the gradient. Also, the authors' representation does not take into account the probability constraints and thus does not calculate the true gradient of the problem. Due to this complex and incomplete gradient calculation, GA can be time consuming and error prone. Techniques more advanced than gradient ascent may be used to traverse the gradient, but these shortcomings remain.

Point-based methods perform very well on many problems, but rely on choosing points that are highly representative of the belief space. If there is a large amount of reachable belief space and the values of different areas of that space are diverse, many points would be needed to reasonably approximate the value. As more points are used, the algorithms become more similar to value iteration and thus become less tractable.

5.2.2 DEC-POMDP approximate methods

Current DEC-POMDP algorithms also have significant drawbacks. As DEC-BPI is an extension of BPI, its disadvantages are similar to those stated above. In general, it allows memory to remain fixed, but provides only a locally optimal solution. This is due to the linear program considering the old controller values from the second step on and the fact that improvement must be over all possible states and initial nodes for the controllers of the other agents. As the number of agents or size of controllers grows, this later drawback is likely to severely hinder improvement. Also, start state knowledge is not used in DEC-BPI. While a heuristic like that used in biased BPI could be added, its applicability often depends on the domain and a more principled solution is desirable.

Szer and Charpillet's approach is limited due to the fact that it searches for optimal deterministic controllers. Since it focuses on deterministic controllers, larger controller sizes may often be required in order to produce values comparable to those generated by a stochastic counterpart. Since the approach relies on search to find an optimal set of controllers of a fixed size, it has a very high space and time requirements. These drawbacks indicate that a new approach is needed to improve the scalability and solution quality of POMDP and DEC-POMDP approximate methods.

5.3 Nonlinear optimization formulations

Contrary to previous approaches for POMDPs ad DEC-POMDPs, our approach defines an optimal fixed-size solution. This is achieved by creating a set of new variables that represent the values of each node (or set of nodes in the DEC-POMDP case) and state pair. Intuitively, this allows changes in the controller probabilities to be reflected in the values of the nodes of the controller. This is in contrast to backups used by BPI and DEC-BPI which iteratively improve the probabilities and easily get stuck in local optima. Our approach allows both the values and probabilities in the controller to be optimized in one step, thus representing the optimal fixed-size controller. To ensure that these values are correct given the controller parameters, nonlinear constraints (based on the Bellman equation) must be added. The resulting NLP is generally harder to solve, but many robust and efficient algorithms can be applied.

Compared to point-based approaches, our formulation does not need to choose a set of points and may be able to cover the belief space better in some problems. That is, while point-based methods work well when there is a small reachable belief space or when the chosen points are very representative of the whole space, the NLP approach seeks to optimize the value of a controller for a specific initial point. For domains in which point-based methods cannot find representative belief points, our approach may still be able to construct high quality controllers. Also, since point-based methods rely on finite-horizon dynamic programming, it may be difficult for these methods to complete the number of backups necessary to approximate the value function well. As our approach uses finite-state controllers, it is more suitable for finding infinite-horizon policies.

As stated above, we believe that our approach can provide high quality, concise solutions and provides a promising framework for more specialized algorithms to be developed. Our NLP is also able to take advantage of the start state distribution of the problem, making better use of the limited controller size. Lastly, because the method searches for stochastic controllers, it is able to find higher-valued, more concise controllers than search in the space of deterministic controllers. In the rest of this section, we present a formal description of the NLP for POMDPs and then show how it can be extended to DEC-POMDPs. We also show how correlation can be incorporated into the DEC-POMDP representation.

For variables: $x(q',a,q,o)$ and $z(q,s)$

Maximize $\sum_s b_0(s)z(q_0,s)$, subject to

The Bellman constraints: $\forall q,s\ z(q,s) =$

$$\sum_a \left[\left(\sum_{q'} x(q',a,q,o) \right) R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_o O(o|s',a) \sum_{q'} x(q',a,q,o) z(q',s') \right]$$

Probability constraints:

$$\forall q,o \sum_{q',a} x(q',a,q,o) = 1, \quad \forall q',a,q,o\ x(q',a,q,o) \geq 0,$$

$$\forall q,o,a \sum_{q'} x(q',a,q,o) = \sum_{q'} x(q',a,q,o_k)$$

Table 5.1. The NLP defining an optimal fixed-size POMDP controller. Variable $x(q',a,q,o)$ represents $P(q',a|q,o)$, variable $z(q,s)$ represents $V(q,s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation.

5.3.1 NLP formulation for POMDPs

Unlike BPI, which alternates between policy improvement and evaluation, the nonlinear program improves and evaluates the controller in one step. The value of an initial node is maximized at an initial state distribution using parameters for the action selection probabilities at each node $P(a|q)$, the node transition probabilities $P(q'|q,a,o)$, and the values of each node in each state $V(q,s)$. To ensure that the value variables are correct given the action and node transition probabilities, nonlinear constraints must be added to the optimization. These constraints are the Bellman equations given the policy determined by the action selection and node transition probabilities. Linear constraints are used to maintain proper probabilities.

To reduce the representation complexity, the action selection and node transition probabilities are merged into one, with

$$P(q',a|q,o) = P(a|q)P(q'|q,a,o) \text{ and } \sum_{q'} P(q',a|q,o) = P(a|q)$$

This results in a quadratically constrained linear program. QCLPs may contain quadratic terms in the constraints, but have a linear objective function. They are a subclass of general nonlinear programs that has structure which algorithms can exploit. This produces a problem that is often more difficult to solve than a linear program, but simpler than a general nonlinear program. The QCLP formulation permits a large number of algorithms to be applied. Because the QCLP is a subclass of the general NLP and to use the same terminology as the DEC-POMDP case, we will refer to the QCLP formulation as an NLP.

Table 5.1 describes the NLP which defines an optimal fixed-size controller for the POMDP case. The value of a designated initial node is maximized given the initial state distribution and the necessary constraints. The first constraint represents the Bellman equation for each node and state which maintains correct values as probability parameters change. The second and third constraints ensure that the x variables represent proper probabilities which sum to one and have value greater than 0. The last constraint guarantees that action selection does not depend on the resulting observation which has not yet been seen. That is, the action probabilities are equal for all observations and thus for any observation, o_k, the value $\sum_{q'} P(q', a|q, o_k) = P(a|q)$.

Theorem 5.3.1. *An optimal solution of the NLP in Table 5.1 results in an optimal stochastic controller for the given size and initial state distribution.*

Proof. To prove this, we must show that an optimal solution of the NLP provides a valid finite-state controller and that the value of this controller is optimal for the given controller size. The NLP defines the controller parameters as $P(a|q) = \sum_{q'} x(q', a, q, o_k)$ and $P(q'|q, a, o) = \frac{x(q',a,q,o_k)}{\sum_{q'} x(q',a,q,o_k)}$ with $q, q', q_0 \in Q$. The probability constraints guarantee that these values are nonnegative and sum to one. Because the parameters are valid, this ensures that a valid stochastic finite-state controller is defined by the NLP. These probabilities are utilized in the Bellman equation for each node q and state s, ensuring that the correct value is determined for each node and

For variables of each agent i: $x(q_i, a_i)$, $y(q_i, a_i, o_i, q'_i)$ and $z(\vec{q}, s)$

Maximize $\sum_s b_0(s) z(\vec{q^0}, s)$, subject to

The Bellman constraints: $\forall \vec{q}, s \; z(\vec{q}, s) =$

$$\sum_{\vec{a}} \prod_i x(q_i, a_i) \Big[R(s, \vec{a}) + \gamma \sum_{s'} P(s'|s, \vec{a}) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q'}} \prod_i y(q_i, a_i, o_i, q'_i) z(\vec{q'}, s') \Big]$$

And probability constraints for each agent i:

$$\forall q_i \sum_{a_i} x(q_i, a_i) = 1, \quad \forall q_i, o_i, a_i \sum_{q'_i} y(q_i, a_i, o_i, q'_i) = 1$$
$$\forall q_i, a_i \; x(q_i, a_i) \geq 0, \quad \forall q_i, o_i, a_i \; y(q_i, a_i, o_i, q'_i) \geq 0$$

Table 5.2. The NLP defining a set of optimal fixed-size DEC-POMDP controllers. For each agent i, variable $x(q_i, a_i)$ represents $P(a_i|q_i)$, variable $y(q_i, a_i, o_i, q'_i)$ represents $P(q'_i|q_i, a_i, o_i)$ and variable $z(\vec{q}, s)$ represents $V(\vec{q}, s)$ where $\vec{q^0}$ represents the initial controller node for each agent.

state pair. Lastly, because the value of the controller is maximized at the initial state distribution of the problem, an optimal solution of NLP must provide an optimal stochastic controller for the the initial distribution and given size. $\square$

5.3.2 NLP formulation for DEC-POMDPs

The nonlinear program for DEC-POMDPs is an extension of the one defined above which incorporates multiple controllers. That is, the NLP seeks to optimize the value of a set of fixed-size controllers given an initial state distribution and the DEC-POMDP model. The variables for this problem are the action selection and node transition probabilities for each node of each agent's controller as well as the joint value of a set of controller nodes. Hence, these variables are for each agent i, $P(a_i|q_i)$ and $P(q'_i|q_i, a_i, o_i)$ and for the set of agents and any state, $V(\vec{q}, s)$. Similar to the way our POMDP approach differs from BPI, this approach differs from DEC-BPI in that it explicitly represents the node values as variables, thus allowing improvement and evaluation to take place simultaneously. To ensure that the values are correct given the action and node transition probabilities, nonlinear constraints must be added to the optimization. These constraints are the DEC-POMDP Bellman equations given the policy determined by the action and transition probabilities. We must also ensure that the necessary variables are valid probabilities.

The NLP representation for DEC-POMDPs is shown formally for an arbitrary number of agents in Table 5.2. The value of the initial set of nodes at the initial state distribution is maximized subject to Bellman and probability constraints. The Bellman constraints ensure that correct values are maintained for each set of nodes and state of the domain. The probability constraints ensure that the action and node transition values are proper probabilities.

Theorem 5.3.2. *An optimal solution of the NLP in Table 5.2 results in optimal stochastic controllers for the given size and initial state distribution.*

Proof. Like the POMDP case, the optimality of the controllers follows from the NLP constraints and maximization of given initial nodes at the initial state distribution. We will show that an optimal solution of the NLP provides a valid set of finite-state controllers and that the value of these controllers is optimal for the given size. The controller parameters are defined in the NLP as $P(a_i|q_i) = x(q_i, a_i)$ and $P(q_i'|q_i, a_i, o_i) = y(q_i', a_i, q_i, o_i)$ with $q, q', q_0 \in Q$. The probability constraints for each agent guarantee that these values are nonnegative and sum to one. Because the parameters are independent and valid for each agent, the NLP defines a valid set of stochastic finite-state controllers. The validity of these probabilities ensures that the value produced by Bellman constraints is correct for each set of nodes $\vec{q}$ and state s. Because the value of the set of controllers is maximized at the initial state distribution of the problem, an optimal solution of NLP must provide an optimal set of stochastic controllers for the initial distribution and given size. $\square$

For variables: $x_F(q', a, o), x_R(q', q, o)$ and $z(q, s)$

Maximize $\sum_s b_0(s) z(q_0, s)$, subject to

The Bellman constraints: $\forall q \neq q_0, s\ z(q_0, s) =$

$$\sum_a \left[\left(\sum_{q'} x_F(q', a, o_k) \right) R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_o O(o|s', a) \sum_{q'} x_F(q', a, o) z(q', s') \right]$$

$\forall q \neq q_0, s\ z(q, s) =$

$$R(s, a_q) + \gamma \sum_{s'} P(s'|s, a_q) \sum_o O(o|s', a_q) \sum_{q' \neq q_0} x_R(q', q, o) z(q', s')$$

Probability constraints:

$$\forall o \sum_{q'} x_F(q', a, o) = 1, \quad \forall q \neq q_0, o \sum_{q'} x_R(q', q, o) = 1$$

$$\forall q', a, o\ x_F(q', a, o) \geq 0, \quad \forall q', q \neq q_0, o\ x_R(q', q, o) \geq 0$$

$$\forall o, a \sum_{q'} x_F(q', a, o) = \sum_{q'} x_F(q', a, o_k)$$

Table 5.3. The more scalable NLP representation for POMDPs. Variable $x_F(q', a, o)$ represents $P(q', a|q_0, o)$, variable $x_R(q', q, o)$ represents $P(q'|q_0, o)$ for a fixed action, a_q at node q, variable $z(q, s)$ represents $V(q, s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation.

5.3.3 Scaling up the NLP formulations

In order to solve NLP representations with a larger number of nodes, we can fix the actions chosen at some nodes of the controller. An example of a POMDP formulation with fixed actions is given in Table 5.3. The initial node of the controller contains both stochastic action selection and stochastic transitions, but the remaining nodes have fixed actions. Because the remaining nodes maintain stochastic transitions, there can be very little or no expressiveness lost in this representation. This is because the action selection probabilities can be merged with the transition probabilities and then this single parameter can be found. If the actions are fixed correctly, the resulting controller will be equivalent to one with stochastic actions and transitions.

The DEC-POMDP NLP formulation can be similarly adapted by beginning at a fully stochastic node in each agent's controller and then transitioning to nodes that have fixed actions. In both cases, a more efficient representation can be constructed by never allowing the initial node to transition to more than one node with a given fixed action. It makes no difference in the controller if the initial node transitions to any specific node with that fixed action as the nodes can be reordered without affecting the value. Because these NLPs have fewer variables, high quality solutions may be found for controller sizes that are too large when using the representation from Tables 5.1 and 5.2.

5.4 Incorporating correlation in the decentralized model

In order to improve the performance of a set of controllers, a shared source of randomness in the form of a *correlation device* can be added. As an improvement to DEC-BPI, Bernstein et al. show that higher-valued controllers can sometimes be achieved when incorporating this approach without sharing any local information. As an example of a correlation device, imagine that before each action is taken, a coin is flipped and both agents have access to the outcome. Each agent can then use the new information to affect their choice of action. Along with stochasticity, correlation is another means of increasing value when memory is limited. For instance, assume the agents' actions are randomized so that each chooses action A 50% of the time and action B 50% of the time. This results in the joint actions of (A, A), (A, B), (B, A) and (B, B) each 25% of the time. When a coin is added, the agents can correlate their actions based on what is seen, such as allowing each agent to perform action A when "heads" is seen and and action B when "tails" is seen. As each signal occurs 50% of the time, this permits joint actions (A, A) and (B, B) to also take place 50% of the time. This type of policy is impossible without the correlation device showing that the policies can be randomized and correlated to allow higher value to be attained.

It is worth noting that there are many ways a correlation device can be implemented in practice. These implementations include each agent using a random number generator with a common seed, or agreeing before execution starts on a list of random numbers. In each of these examples, each agent has access to the random signal at each step, but no local information is shared and thus the execution remains decentralized. While a correlation device permits higher value to be found given a fixed-size controller, a larger set of uncorrelated controllers can represent the same value. Essentially, this is done by incorporating the correlation signal into the agents' controllers. Additional details can be found in [15].

More formally, a correlation device provides extra signals to the agents and operates independently of the DEC-POMDP. That is, the correlation device is a tuple $\langle C, \psi \rangle$, where C is a set of states and $\psi : C \rightarrow \Delta C$ is a stochastic transition function that we will represent as $P(c'|c)$. At each step of the problem, the device transitions and each agent can observe its state.

The independent local controllers defined above (in Table 5.2) can be modified to make use of a correlation device. This is done by making the parameters dependent on the signal from the correlation device. For agent i, action selection is then $P(a_i|q_i, c)$ and node transition is $P(q_i'|q_i, a_i, o_i, c)$. For n agents, the value of the correlated joint controller beginning in nodes $\vec{q}$, state s and correlation device state c is defined as

$$V(\vec{q}, s, c) = \sum_{\vec{a}} \prod_i P(a_i|q_i, c) \cdot$$
$$\left[R(s, \vec{a}) + \gamma \sum_{s'} P(s'|\vec{a}, s) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q'}} \prod_i P(q_i|q_i, a_i, o_i, c) \sum_{c'} P(c'|c) V(\vec{q'}, s', c') \right]$$

The NLP formulation can be extended to include a correlation device. The optimization problem, shown in Table 5.4 *without* the probability constraints, is very

For variables of each agent i: $x(q_i,a_i,c)$, $y(q_i,a_i,o_i,q_i',c)$, $z(\vec{q},s,c)$ and $w(c,c')$

Maximize $\sum_s b_0(s)z(\vec{q^0},s,c)$, subject to

The Bellman constraints:

$$\forall \vec{q}, s\ z(\vec{q},s,c) = \sum_{\vec{a}} \prod_i x(q_i,a_i,c) \cdot$$

$$\left[R(s,\vec{a}) + \gamma \sum_{s'} P(s'|s,\vec{a}) \sum_{\vec{o}} O(\vec{o}|s',\vec{a}) \sum_{\vec{q'}} \prod_i y(q_i,a_i,o_i,q_i',c) \sum_{c'} w(c,c') z(\vec{q'},s',c)\right]$$

Table 5.4. The nonlinear program for including a correlation device in our DEC-POMDP NLP formulation. For each agent i, variable $x(q_i,a_i,c)$ represents $P(a_i|q_i,c)$, variable $y(q_i,a_i,o_i,q_i',c)$ represents $P(q_i'|q_i,a_i,o_i,c)$, variable $z(\vec{q},s,c)$ represents $V(\vec{q},s,c)$, $\vec{q^0}$ represents the initial controller node for each agent and $w(c,c')$ represents $P(c'|c)$. The other constraints are similar to those above with the addition of a sum to one constraint for the correlation device.

similar to the previous NLP. A new variable is added for the transition function of the correlation device and the other variables now include the signal from the device. The Bellman equation incorporates the new correlation device signal at each step, but the other constraints remain the same. A new probability constraint is added to ensure that the transition probabilities for each state of the correlation device sum to one.

5.5 Methods for solving the NLP

Many efficient constrained optimization algorithms can be used to solve large NLPs. Constrained optimization seeks to minimize or maximize an objective function based on equality and inequality constraints. When the objective and all constraints are linear, this is called a linear program (LP). As our POMDP formulation has a linear objective function, but contains some quadratic constraints, it is a quadratically constrained linear program. The DEC-POMDP representation also has a linear objective, but has some nonlinear constraints whose degree depends on the number of agents in the problem. Unfortunately, both of these problems are nonconvex. Essentially, this means that there may be multiple local maxima as well as global maxima, causing finding the globally optimal solution to often be very difficult. In the next two subsections, we discuss a method to find these globally optimal solutions as well as a more practical method for finding locally optimal solutions.

5.5.1 Finding a globally optimal solution

One method that could be used to find optimal solutions for our NLP is *difference of convex functions* (DC) programming [55]. DC programming uses a general formulation that allows nonlinear and nonconvex constraints and is studied within the field of global optimization. This formulation requires an objective function and constraints that are a difference of two convex functions. Under mild conditions, it is possible to devise DC programming algorithms that converge to the globally optimal solution.

More formally, a function f is called DC if it is real-valued and defined over a convex set $C \subset \mathbb{R}^n$ and $\forall\, x \in C$, f can be decomposed such that $f(x) = p(x) - q(x)$ where p and q are convex functions on C. A *DC programming problem* then has the form

$$\min f(x) \text{ such that } x \in C \text{ and } g_i(x) \leq 0$$

where $C \subset \mathbb{R}^n$ is convex and objective function f and each constraint g_i are DC on C.

Proposition 5.5.1. *Optimal solution of the POMDP and DEC-POMDP NLP formulations can be defined as DC programming problems.*

Proof. First, the maximization in our formulations can be changed to a minimization by multiplying by a factor of -1. We can then define x and y over all nonnegative Real numbers and z over all Real numbers. This represents a convex set over which the variables are defined. We can represent each equality constraint in the NLP formulation as inequality constraints with opposite sign to achieve the form above. Horst and Tuy show that any function whose second partial derivatives are continuous is also DC (corollary I.1). Because the NLP objective function and all constraints are polynomials, this means they are infinitely differentiable and these derivatives are continuous. Thus, the variables are defined over a convex set and the objective and constraints are DC in both formulations. This allows our NLP representations to be written as DC programming problems. □

Therefore, POMDPs and DEC-POMDPs can be formulated as DC programming problems. This has mostly theoretical significance; we doubt that this formulation could be effective in practice. In principle, global optimization algorithms such as branch and bound can be used, but because of the large size of the resulting DC programing problem, it is unlikely that current optimal solvers can handle even small POMDPs or DEC-POMDPs. Nevertheless, it would be interesting to identify classes of these problems for which DC optimization is practical.

5.5.2 Finding a locally optimal solution

Since it may not be possible or feasible to solve the NLP optimally, locally optimal methods are often more useful in practice. A wide range of nonlinear programming algorithms have been developed that are able to efficiently solve nonconvex problems with many variables and constraints. Locally optimal solutions can be guaranteed, but at times, globally optimal solutions can also be found. For example, merit functions, which evaluate a current solution based on fitness criteria, can be used to improve convergence and the problem space can be made convex by approximation or domain information. These methods are much more robust than simpler methods such as gradient ascent, while retaining modest efficiency in many cases.

For this thesis, we used a freely available nonlinearly constrained optimization solver called *snopt* [38] on the NEOS server (www-neos.mcs.anl.gov). The algorithm finds solutions by a method of successive approximations called sequential quadratic programming (SQP). SQP uses quadratic approximations which are then solved with quadratic programming (QP) until a solution to the more general problem is found. A QP is typically easier to solve, but must have a quadratic objective function and linear constraints. In snopt, the objective and constraints are combined and approximated to produce the QP. A merit function is also used to guarantee convergence from any initial point.

Other techniques that take advantage of the structure of POMDPs and DEC-POMDPs in order to increase the scalability of certain solution methods would be beneficial. One promising idea is to use constraint partitioning [113] to break up a problem and solve the resulting pieces in such a way that allows a feasible solution to be found.

5.6 Experiments

For experimental comparison, we present an evaluation of the performance of our formulation using an off-the-shelf nonlinear program solver, snopt, as well as leading POMDP and DEC-POMDP approximation techniques. In these experiments we seek to determine how well the NLP formulation performs when used in conjunction with a generic solver such as snopt. The formulation is very general and many other solvers may be applied. While the POMDP results are intended to show that a simple solution of the NLP is competitive with current solution methods and can outperform point-based approaches on certain problems, the DEC-POMDP results show consistent improvement in solution quality over previous approaches in a range of problems.

5.6.1 POMDP results

In this section, we compare the results obtained using our new formulation with those of biased BPI, PBVI, PERSEUS, HSVI and PBPI. GA was also implemented, but produced significantly worse results and required substantially more time than the other techniques. Thus we omit the details of GA and focus on the more competitive techniques.

We also implemented a version of the fixed action formulation described in Table 5.3. For controller sizes larger than the number of available actions, we cycled through the actions and assigned the next available action to the next node. For smaller controller sizes, we randomly assigned actions to nodes. Throughout this section, we will refer to the optimization of our NLP using snopt as *NLO* for the fully stochastic case and as *NLO fixed* for the optimization with fixed actions.

Each of our NLP methods was initialized with ten random deterministic controllers and we report mean values and times for solving the given controller size. To slightly increase the performance of the solver, upper and lower bounds of $R_{max}/(1-\gamma)$ and $R_{min}/(1-\gamma)$ were added. The values $R_{max} = \max_{s,a} R(s,a)$ and $R_{min} = \min_{s,a} R(s,a)$ and the bounds represent repeating the best and worst possible immediate reward for the infinite steps of the problem. The results were found by using the NEOS server, which provides a set of machines with varying CPU speeds, but we expect the times to vary by only a small constant.

Benchmark problems

We first provide a comparison of the NLP formulations with leading POMDP approximation methods on common benchmark problems. The first three domains, which were introduced by Littman, Cassandra and Kaelbling [64], are grid problems in which an agent must navigate to a goal square. The set of actions consists of turning left, right or around, moving forward and staying in place. The observations

Algorithm	Value	Representation Size	Running Time
Tiger-grid $\lvert S\rvert = 36$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 17$			
HSVI	2.35	4860	10341
PERSEUS	2.34	134	104
HSVI2	2.30	1003	52
PBVI	2.25	470	3448
PBPI	2.24	3101	51
biased BPI	2.22	120	1000
NLO fixed	2.14	32	2411
BPI	1.81	1500	163420
NLO	1.79	14	1174
Q_{MDP}	0.23	n.a.	2.76
Hallway $\lvert S\rvert = 60$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 21$			
PBVI	0.53	86	288
HSVI2	0.52	147	2.4
HSVI	0.52	1341	10836
biased BPI	0.51	43	185
PERSEUS	0.51	55	35
BPI	0.51	1500	249730
NLO fixed	0.49	25	644
NLO	0.47	12	362
Q_{MDP}	0.27	n.a.	1.34
Hallway2 $\lvert S\rvert = 93$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 17$			
PERSEUS	0.35	56	10
HSVI2	0.35	114	1.5
PBPI	0.35	320	3.1
HSVI	0.35	1571	10010
PBVI	0.34	95	360
biased BPI	0.32	60	790
NLO fixed	0.29	18	251
NLO	0.28	13	420
BPI	0.28	1500	274280
Q_{MDP}	0.09	n.a.	2.23
Tag $\lvert S\rvert = 870$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 30$			
PBPI	-5.87	818	1133
PERSEUS	-6.17	280	1670
HSVI2	-6.36	415	24
HSVI	-6.37	1657	10113
biased BPI	-6.65	17	250
NLO fixed	-8.14	7	5669
BPI	-9.18	940	59772
PBVI	-9.18	1334	180880
NLO	-13.94	2	5596
Q_{MDP}	-16.90	n.a.	16.1

Table 5.5. Values, representation sizes and running times (in seconds) for the set of benchmark problems. Results for other algorithms were taken from the following sources: BPI [86], biased BPI [85], HSVI [104], HSVI2 [105], PERSEUS [108], PBPI [56], PBVI [83], Q_{MDP} [108]

consist of the different views of the walls in the each domain based on each agent's ability to observe walls in four directions and a unique observation for the goal state. There are also three additional observable landmarks in the Hallway problem. Both the observations and transitions are extremely noisy and the start state is a random non-goal state. Only the goal has a reward of 1 and the discount factor used was 0.95. Following the convention of previously published results, we consider the versions of the Hallway problems that stop after the goal has been reached and for the Tiger-grid problem reward only accumulates for the first 500 steps.

The other benchmark problem is a larger grid problem in which the goal is for the agent to catch and tag an opponent which attempts to move away [83]. The tagging agent has perfect knowledge of its state, but it cannot observe the opponent unless it is located in the same square. The agent can deterministically move in each of four directions and can use the "tag" action. The opponent attempts to move away from the agent in a fixed way, but may stay in the same location with probability 0.2. If the agent succeeds in tagging the opponent, a reward of 10 is given, but if the opponent is not in the same square when the tag action is taken a reward of -10 is given. All other actions result in a -1 reward and like the other benchmarks, the problem stops once the goal is reached (the opponent is successfully tagged) and a discount factor of 0.95 was used.

Table 5.5 shows the results from previously published algorithms and our NLP formulations. We provide the mean values and times for the largest controller size that is solvable with less than (approximately) 400MB of memory and under eight hours as these limits are imposed on the NEOS server. The values of PBPI in the table are the highest values reported for each problem, but the authors did not provide results for the Hallway problem. Because the experiments were conducted on different computers, solution times give a general idea of the speed of the algorithms. It is also worth noting that while most of the other approaches are highly optimized, a generic solver is used with our approach in these experiments.

In general, we see that the nonlinear optimization approach is competitive both in running time and value produced, but does not outperform the other techniques. Our method achieves 91%, 92%, 83% of maximum value in the first three problems, but does so with a much smaller representation size. A key aspect of the NLP approach is that high quality solutions can be found with very concise controllers. This allows limited representation size to be utilized very well, likely better than the other approaches in most problems.

The values in the table as well as those in Figure 5.1 show that our approach is currently unable to find solutions for large controller sizes. For example, in the Tag problem, a solution could only be found for a two node controller in the general case and a seven node controller when the actions were fixed. As seen in the figures, there is a near monotonic increase of value in the Hallway problem when compared with either controller size or time. As expected, fixing the actions at each node results in faster performance and increased scalability. This allows higher valued controllers to be produced, but scalability remains limited. As improved solvers are found, higher values and larger controllers will be solvable by both approaches. The other benchmark problems display similar trends of near monotonic improvement and the need for increased scalability to outperform other methods on these problems.

Infinite-horizon domains

Because the above benchmark problems are indefinite-horizon (or technically finite-horizon in the case of the Tiger-grid problem) methods such as point-based approaches that build up policies one step at a time will often perform well. Controller-based approaches may also perform well in indefinite-horizon problems because the controller may concisely represent a solution, but they cannot easily be used for finite-horizon

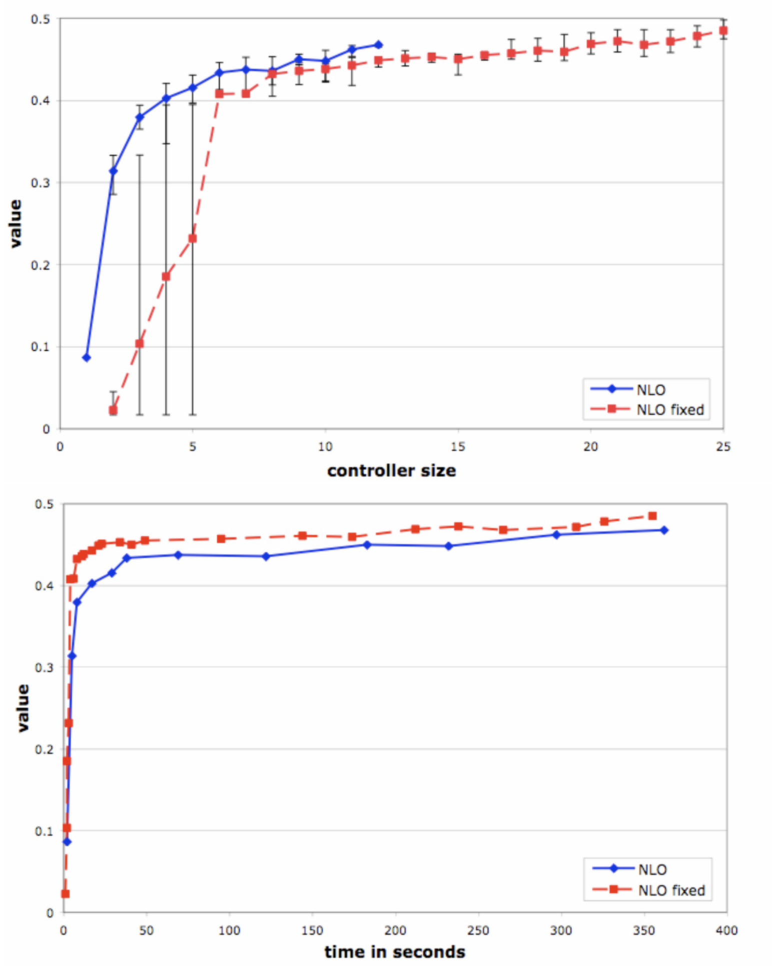


Figure 5.1. Hallway problem values for various controllers sizes (top) and running times (bottom) using the NLP formulation with and without fixed actions at each node. Bars are used in the left figure to represent the minimum and maximum values for each size.

problems. For the Tiger-grid problem, because the discount factor is 0.95, after 500 steps, the maximum reward is less than 1×10^{-11} so it can effectively be considered an infinite-horizon problem without affecting the final solution quality.

True infinite-horizon problems with higher discount rates pose a tougher challenge for point-based methods. In these problems, the effective horizon will be high and thus methods based on value iteration will require a large number of steps in order to construct a policy. Conversely, controller-based methods will have an advantage over value iteration methods. This is because the controller can concisely represent an

Algorithm	Value	Representation Size	Running Time
Hallway $\lvert S\rvert = 60$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 21$			
NLO fixed	51.98	18	185
NLO	48.62	7	232
HSVI2	48.45	112	3622
Hallway2 $\lvert S\rvert = 93$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 17$			
NLO fixed	1.97	13	309
NLO	1.66	6	163
HSVI2	1.18	2540	3627

Table 5.6. Values, representation sizes and running times (in seconds) for the original Hallway and Hallway2 problems that reset after the goal has been reached.

infinite-horizon solution and generating this controller often requires similar resources regardless of discount factor. In cases where an optimal or near-optimal policy is periodic, a controller-based method can represent a very high quality solution without the repetition in the policy that would be required for value iteration approaches.

To demonstrate this phenomenon, we examined the performance of the NLP approaches and HSVI2 in the original versions of the Hallway and Hallway2 problems which reset to the initial state after the goal has been reached. The discount factors for the Hallway domain were 0.999 and 0.99 for the Hallway2 domain. HSVI2 was chosen for comparison because of very high performance in each of the benchmark domains and availability of the algorithm. Software for HSVI2 was used from web site of Trey Smith which was run with a time limit of one hour.

On these problems, which we provide results for in Table 5.6, the NLP formulation provides results that have higher value, smaller representation and require less time than HSVI2. While the improvement is modest in the Hallway domain, it is much more substantial in the Hallway2 domain. As stated above, this is due to the high discount factor and the ability of a concise, periodic policy to perform well. Using fixed actions at each node after the first allows further performance gains. This occurs because the fixed action formulation is more scalable, permitting larger controllers to be solved. Overall, these results demonstrate the advantages of controller-based approaches in general and the NLP approach in particular in true infinite-horizon domains with discount factors close to one.

5.6.2 DEC-POMDP results

We also tested the nonlinear programming approach in four DEC-POMDP domains. In each experiment, we compare our NLP solutions using snopt with Bernstein et al.'s DEC-BPI and Szer and Charpillet's BFS for a range of controller sizes. We also implemented the NLP and DEC-BPI techniques with a correlation device of size two. Like the POMDP version, to improve scalability we also solved the NLP with a stochastic action at the initial node and fixed actions at the other nodes. The action assignments were made as described above with each action in turn fixed to each node when the number of nodes was greater than the number of actions and random action assignments in each node for smaller controller sizes.

Each NLP and DEC-BPI algorithm was run until convergence was achieved with ten different random deterministic initial controllers, and the mean values and running times are reported. Unlike the figure in the POMDP section, the figures in this section do not include maximum and minimum value bars. While these could have been provided, they are very similar for each of our NLP methods and are omitted to increase readability. Also, we do not always show the results for the largest solvable controller once the values plateau. The optimal deterministic controller returned by BFS is also reported if it could be found in under eight hours. The times reported for each NLP method can only be considered estimates due to running each algorithm on external machines with uncontrollable load levels, but we expect that they vary by only a small constant factor. Throughout this section we will refer to the optimization of our NLP with snopt as *NLO*, the optimization with the correlation device with two states as *NLO corr*, and these optimizations with fixed actions as *NLO fixed* and *NLO fixed corr*.

Recycling robots As seen in Figure 5.2, in this domain all the methods performed fairly well. While the DEC-BPI techniques converge over time to a value around 27, the other approaches converge to at least 30. BFS, the general formulation (NLO) and the formulation with fixed actions and a correlation device (NLO fixed corr) perform particularly well on this problem, converging to a value near 32 with very small controllers. We also see that these values were found quickly, often taking less than a minute for most methods. BFS could solve at most a controller of size three, but in this problem that is sufficient to produce a high-valued solution. We also see that while correlation helps in the case of fixed actions (NLO fixed corr), it does not improve solution quality in the general case (NLO corr). Fixing actions in the NLP approach (NLO fixed) allows larger controllers to be solved, but an increase in value is not found.

Multiagent tiger problem In this problem, DEC-BPI performed very poorly (never producing values higher than -50) causing us to remove the values from Figure 5.3 for the sake of clarity. We can also see the limitation of BFS in the figures. While it can do very well for three nodes, it cannot find solutions for larger controllers. In contrast, the NLP approaches demonstrate a trend of higher-valued solutions as controller size increases. This suggests that as more scalable solution methods for our formulation are found, the increase in solution quality will persist. The NLP approaches also make better use of time, quickly producing high solution quality and then value increases more slowly. We also see that while correlation is helpful, fixing actions does not improve solution quality. This is likely because a high quality solution for this domain will consist of many listen actions, but very few open actions. Our simple method of choosing actions does not take this information into account and

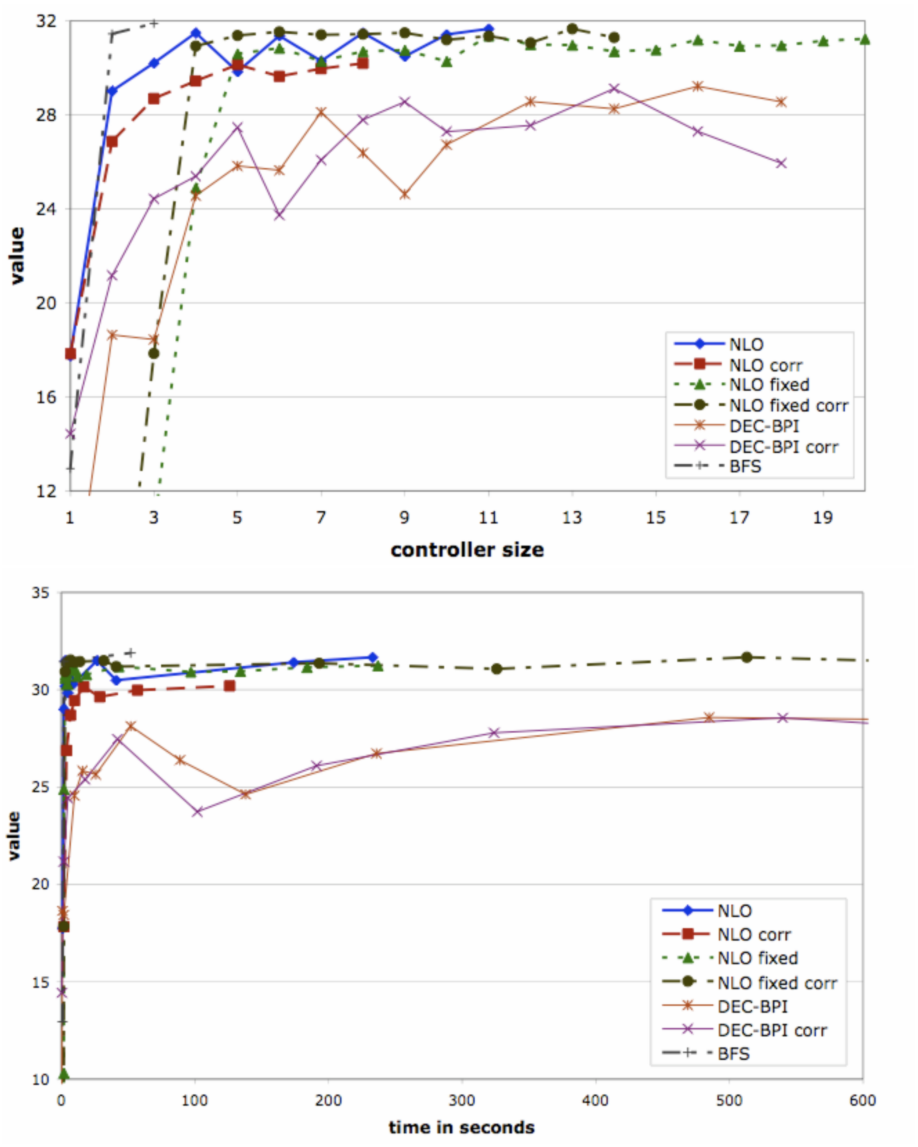
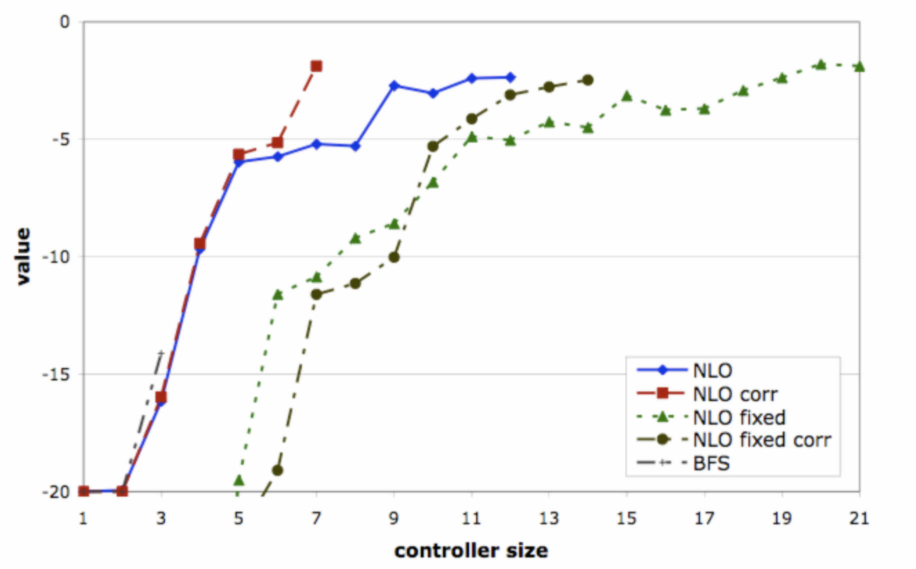


Figure 5.2. Recycling robots values for various controller sizes (top) and running times (bottom) using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS



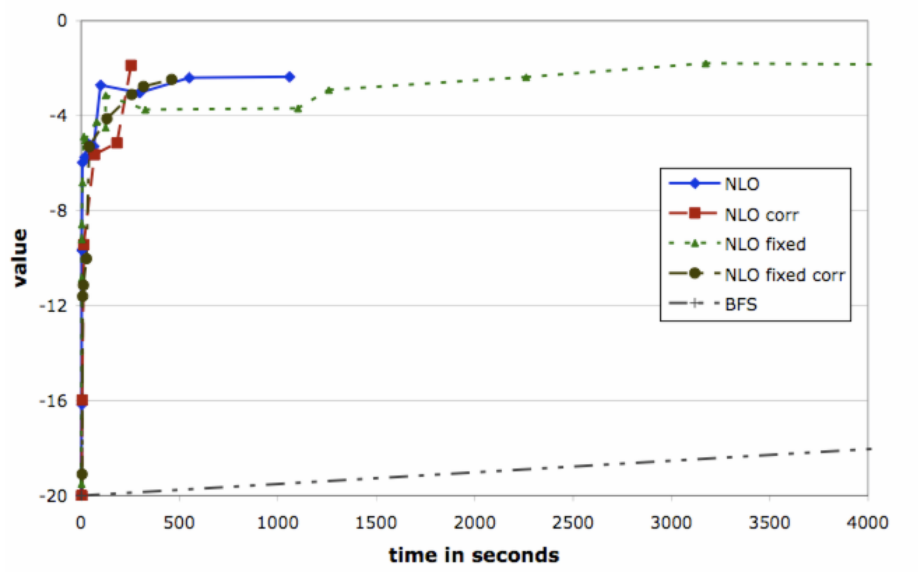
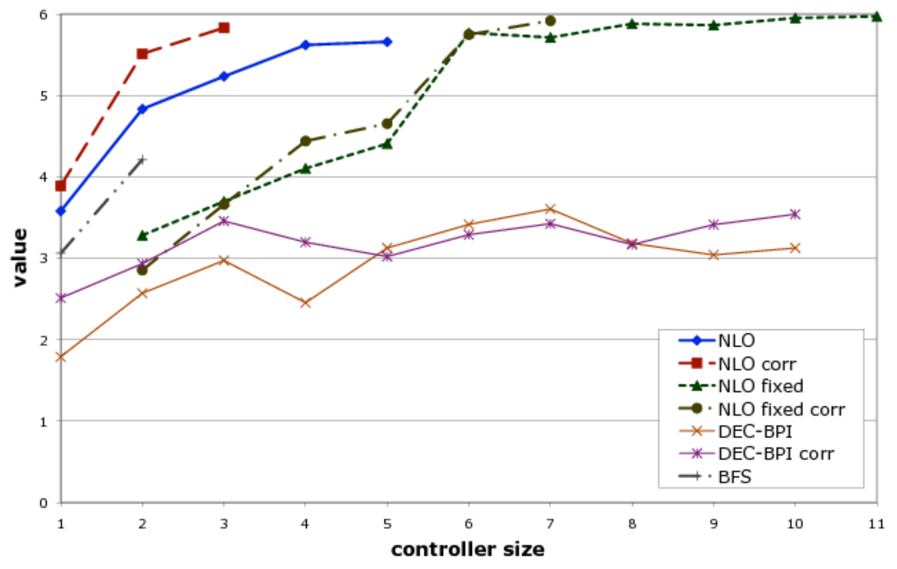


Figure 5.3. Multiagent tiger problem values for various controller sizes (top) and running times (bottom) using NLP methods with and without a 2 node correlation device and BFS

thus requires larger controllers to produce a high value. While displaying an upward trend, the solutions for this problem, particularly with fixed actions, are somewhat irregular. This is likely due to the fact that there is a large penalty for opening the wrong door which can significantly lower the average presented in the figure. Thus, there is a large variance in solution quality but the high performance of the NLP methods is clear.

Meeting in a grid The results for this example can be seen in Figure 5.4. The figures show that, in general, the NLP techniques and BFS outperform DEC-BPI. We also see that the fixed action NLO methods (NLO fixed and NLO fixed corr) perform especially well, producing the highest values. The standard NLP methods also perform well, as they provide nearly the same value as the fixed action methods. The figure on the bottom shows that for all methods, a near maximum value was reached quickly by all methods ($\sim$1 minute) and then, if larger controllers could be solved, value increased very slightly thereafter. We also see that correlation increases solution quality in this problem, particularly when using the general formulation (NLO corr).

Box pushing The results for this problem, which represents a larger infinite-horizon problem, are given in Table 5.7. Because this problem is so large and given the time and memory limitations, BFS is only able to provide a solution for a one node controller and both DEC-BPI and our general approach with a correlation device (NLO corr) could not solve the formulation for a four node controller. Nonetheless, our techniques perform very well. The general formulations (NLO and NLO corr) never produce lower value than DEC-BPI or BFS and often produce much higher value. Fixing actions allows solutions to be found very quickly, but because random actions are used for controller sizes 2-4, performance is poor for these sizes. Because larger controllers can be solved, these fixed action formulations (NLO fixed and NLO



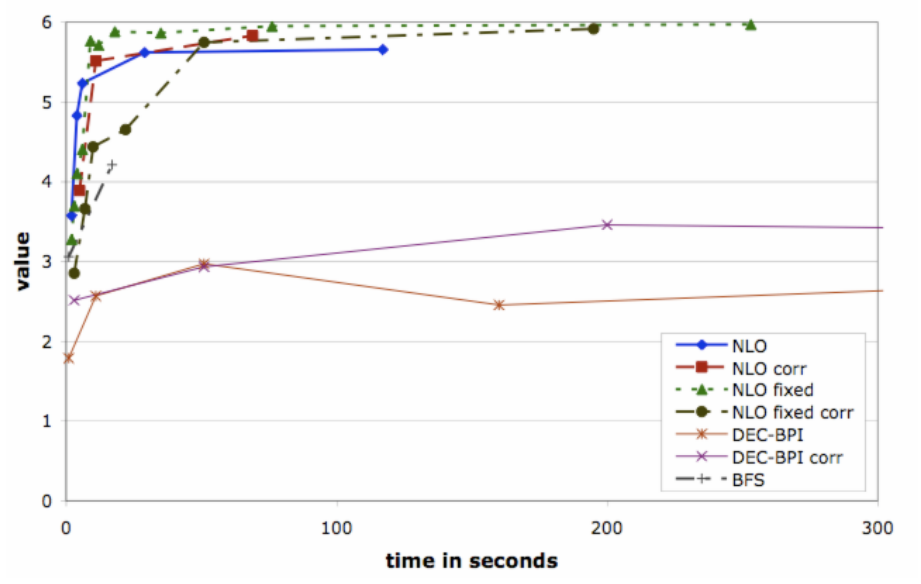


Figure 5.4. Meeting in a grid problem values for various controller sizes (top) and running times (bottom) using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS

Size	NLO	NLO corr	NLO fix	fix corr	DEC-BPI	BPI corr	BFS
Value							
1	-1.580	6.267	n/a	n/a	-10.367	-10.012	-2
2	31.971	39.825	-6.252	-10.865	3.293	4.486	x
3	46.282	50.834	5.097	5.300	9.442	12.504	x
4	50.640	x	18.780	25.590	7.894	x	x
5	x	x	53.132	53.132	14.762	x	x
6	x	x	73.247	x	x	x	x
7	x	x	80.469	x	x	x	x
Running Time							
1	20	21	n/a	n/a	26	87	1696
2	115	314	18	21	579	3138	x
3	683	1948	27	37	4094	15041	x
4	5176	x	44	108	11324	x	x
5	x	x	92	12189	27492	x	x
6	x	x	143	x	x	x	x
7	x	x	256	x	x	x	x

Table 5.7. Box pushing problem values and running times (in seconds) for each controller size using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS. An ‘x’ signifies that the approach was not able to solve the problem and because the fixed action formulation cannot solve 1 node controllers, an ‘n/a’ is given.

fixed corr) allow greater value to be found. The results from all domains suggest that the general formulation should be used when controller size is less than the number of actions in the problem, but the fixed action method can provide benefits for larger controller sizes. Also, all NLP methods required much less time than the DEC-BPI techniques when the problems were solvable. And in general, correlation provides only a slight increase in value for the various methods.

5.7 Discussion

In this chapter, we discussed a new approach for solving POMDPs and DEC-POMDPs by defining optimal fixed-size solutions as nonlinear programs. This permits these problems to be solved by a wide range of powerful NLP solution methods. Because our approach is able to incorporate a known start state and applies nonlinear optimization to a stochastic controller in the single agent case or a set of stochastic controllers for the multiagent case, it is able to make better use of the limited representation space than other approaches. Thus, for a large number of POMDPs and DEC-POMDPs this new formulation may allow very concise high quality solutions.

We showed that by using a generic nonlinear program solver, the NLP formulation can provide POMDP solutions that are competitive with leading techniques in terms of running time and solution quality. The NLP method also outperformed one such method (HSVI2) in two domains with high discount factors and no goal state in which the problem stops. In this type of domain, point-based algorithms and other approaches based on value iteration will require many backup steps to represent a policy and thus performance suffers. On the contrary, controller-based methods such as the NLP formulation, represent the policy as infinite-horizon, eliminating the need for these backups. In many true infinite-horizon domains, the stochastic and periodic policies of controller-based methods are likely to perform well.

The NLP approach has some other distinctive characteristics and advantages. In particular, it produces high value with significantly less memory. And because the controller is more compact, the approach may use less time despite the use of nonlinear optimization. This work opens up new promising research directions that could produce further improvement in both quality and efficiency. As the solver in our experiments has not been optimized for our NLP it is likely that even higher performance can be achieved in the future.

In the DEC-POMDP case, we demonstrated that our approach is able to produce solutions that are significantly higher-valued than the other infinite-horizon DEC-POMDP algorithms for a range of problems. In all problems, a variant of our NLP approach produced the highest value that was generated for any controller size of any method. This included a problem that is an order of magnitude larger than any used previously in infinite-horizon DEC-POMDP literature. The combination of start state knowledge and more advanced optimization techniques allows us to make efficient use of the limited space of the controllers. In addition, incorporating a correlation device as a shared source of randomness can further increase solution quality. The running times needed to produce solutions for each problem also show that our NLP approach is frequently the most efficient method to achieve any given value. These results show that our approach can quickly find compact, high-valued solutions for several different types of DEC-POMDPs. Lastly, as the number of agents in a given problem grows, the number of local maxima is likely to increase. Because solution of our NLP can avoid many of these suboptimal points, our representation should also perform well when the number of agents is increased beyond two.

In the future, we plan to explore more specialized algorithms that can be tailored for our optimization problem. While the performance achieved with a standard non-linear optimization algorithm is good, specialized solvers will likely further increase solution quality and scalability. For instance, in both POMDPs and DEC-POMDPs different optimization methods should be able to take advantage of the inherent problem structure. Also, we are interested in identifying important subclasses for which globally optimal solutions can be efficiently provided. More study is also needed on the cost effectiveness of the correlation device. This would allow us to know when a correlation device will provide better solutions than adding more nodes to the controller. For important classes of POMDPs and DEC-POMDPs, both of these improvements may allow optimal or near optimal fixed-size solutions to be found.

In the next chapter, we address some of these concerns by discussing a different controller representation. This Mealy representation will allow higher-valued controllers to be defined for the same number of controller nodes and some problem structure to be exploited. As a consequence, better performance can be achieved with the NLP formulation and this Mealy representation.

CHAPTER 6
POLICIES AS MEALY MACHINES

As mentioned in the previous chapter, finite-state controllers are popular and efficient solution representations for optimal and approximate approaches for POMDPs and DEC-POMDPs. Existing controller-based approaches are based on automata with output known as Moore machines. In this chapter, we discuss an alternate controller representation based on another type of automata, the Mealy machine. Mealy machines are more powerful than Moore machines, provide a richer structure that can be exploited by solution methods and can be easily incorporated into current controller-based approaches. To demonstrate this, we adapted some existing controller-based algorithms to use Mealy machines and obtained results on a set of benchmark domains. The Mealy-based approach always outperformed the Moore-based approach and often outperformed the state-of-the-art algorithms for both centralized and decentralized POMDPs. These findings provide fresh and general insights for the improvement of existing algorithms and the development of new ones.

6.1 Overview

Current ϵ-optimal approaches (see Chapter 4, as well as [15, 45]) are intractable except for very small problem sizes, leading a wide range of approximate methods to be developed for POMDPs [5, 70, 85, 86] and DEC-POMDPs [5, 15, 17, 110]. Finite-horizon methods have also been extended to solve infinite-horizon problems. For centralized POMDPs, these approaches, such as point-based methods [19, 83, 105, 108], build up a solution at each problem step until the discount factor causes further actions to not substantially change the quality of the solution. Point-based approaches often work well in practice, but rely on a manageable set of belief points and a solution that does not require a large number of steps to build. In contrast, controller-based approaches allow a concise, inherently infinite-horizon representation, that are more appropriate for certain problem types.

Approximation algorithms using finite-state controllers have been shown to provide high-quality solutions in a wide range of POMDPs and DEC-POMDPs. All current approaches are based on a type of automata with output called Moore machines. We show that these methods can be improved by using controllers based on Mealy machines rather than Moore machines. A Mealy machine allows a more concise representation as well as additional structure that can be exploited by the solution methods. Our contribution is very general since all existing algorithms that are based on controllers can be adapted to use Mealy machines, often leading to higher quality solutions. We demonstrate this improved performance on a range of benchmark problems for centralized and decentralized POMDPs.

6.2 Automata with output

An automaton with output produces an output string when processing an input string. There are two main types of machines, Moore and Mealy. Moore machines associate output symbols with nodes and Mealy machines associate output symbols with transitions.

Formally, a Moore machine is a tuple $\langle Q, \Sigma, \Omega, \delta, \lambda, q_0 \rangle$ where Q is the set of nodes, Σ and Ω are the input and output alphabets, $\delta : Q \times \Sigma \to Q$ is the *deterministic* transition function, $\lambda : Q \to \Omega$ is the *output* function, and $q_0 \in Q$ is the initial node. On an input string $a_1, \ldots, a_n \in \Sigma^*$, the machine outputs the string $\lambda(q_0), \ldots, \lambda(q_n) \in \Omega^*$ where $q_i = \delta(q_{i-1}, a_i)$ for $i = 1, \ldots, n$.

A Mealy machine is a tuple $\langle Q, \Sigma, \Omega, \delta, \lambda, q_0 \rangle$ where Q, Σ, Ω and q_0 are as before, but $\lambda : Q \times \Sigma \rightarrow \Omega$ is the output function that associates output symbols with transitions of the automaton. Given an input string $a_1, \ldots, a_n \in \Sigma^*$, the machine outputs $\lambda(q_0, a_1), \ldots, \lambda(q_{n-1}, a_n) \in \Omega^*$ where $q_i = \delta(q_{i-1}, a_i)$ for $i = 1, \ldots, n$.

Both models are equivalent in the sense that for a given machine of one type, there is a machine of the other type that generates the same outputs. However, it is known that Mealy machines are more *succinct* than Moore machines; given a Moore machine M_1, one can find an equivalent Mealy machine M_2 with the same number of nodes by constraining the actions taken at each node to be the same for all observations. However, given a (general) Mealy machine, the equivalent Moore machine has $|Q| \times |\Omega|$ nodes [54].

In this thesis, we are interested in *stochastic automata.* These automata are similar to those presented above except that probability distributions are used for transitions and output. That is, for a Moore machine the transition function is $\delta : Q \times \Sigma \rightarrow \Delta Q$ (where ΔQ is the set of probability distributions over Q) and the output function is $\lambda : Q \rightarrow \Delta\Omega$. For Mealy machines, we use a *coupled* or *joint* function $\tau : Q \times \Sigma \rightarrow \Delta(Q \times \Omega)$ that specifies both the transition and output functions. Note that these representations differ slightly from those in Section 2.2.2 in that the transition no longer depends on the action in the Moore case. This is a minor difference that does not change the theoretical or practical properties of the automata and is used to be consistent with how Moore machines have traditionally been represented.

6.3 Finite-state controllers

As discussed in Section 2.2.2, finite-state controllers can be used to represent policies for agents in an elegant way since an agent can be thought as a device that receives observations and produces actions. For POMDPs a single controller provides the policy of the agent, while for DEC-POMDPs a set of controllers, one per agent, provides the joint policy of the agents. We discuss the use of Moore and Mealy machines for stochastic control of POMDPs and DEC-POMDPs.

6.3.1 Moore machines

In the standard POMDP, observations are generated only after the application of actions. This feature makes the use of Moore machines natural for representing policies as shown in the following. Let $M = \langle Q, O, A, \delta, \lambda, q_0 \rangle$ be an stochastic Moore machine denoting the stochastic policy for the agent and assume that the environment is initially in state s_0. The first action applied by the agent, as dictated by M, is a_0 with probability $\lambda(q_0)(a_0)$. The system makes a transition to state s_1 with probability $P(s_1|s_0, a_0)$ and generates observation o_1 with probability $P(o_1|s_1, a_0)$. This observation triggers a transition in M to node q_1 with probability $\delta(q_0, o_1)(q_1)$, a new action a_1 with probability $\lambda(q_1)(a_1)$, and so on. The infinite-horizon discounted reward incurred by policy M when the initial state is s and the initial node of the automaton is q can be denoted by $V_M(q, s)$, or $V(q, s)$ when M is clear from context and satisfies:

$$V(q,s) = \sum_{\vec{a}} P(a|q) \Big[R(s,a) + \gamma \sum_{s',o,q'} P(s',o|s,a) P(q'|q,o) V(q',s') \Big]$$

where

$$P(a|q) = \lambda(q)(a), P(q'|q,o) = \delta(q,o)(q') \text{ and } P(s',o|s,a) = P(s'|s,a)P(o|s',a)$$

The value of M at the initial distribution is $V_M(b) = \sum_s b(s) V_M(q_0, s)$.

The DEC-POMDP case is similar except a Moore machine, M_i, is used to represent a policy for each agent i. The infinite-horizon discounted reward incurred by joint

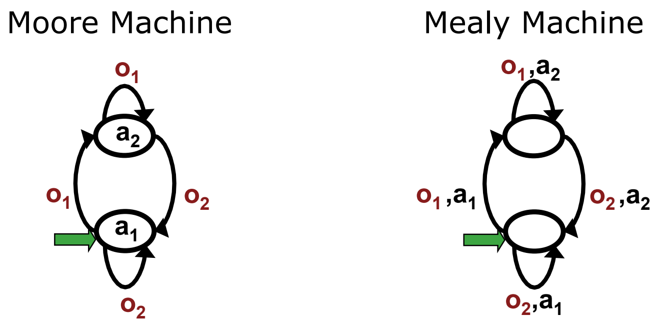


Figure 6.1. Deterministic Moore and Mealy machines

policy $\vec{M}$ when the initial state is s and the initial nodes of the automatons are $\vec{q}$, can be denoted by $V_{\vec{M}(\vec{q},s)}$, or $V(\vec{q},s)$ when $\vec{M}$ is clear from context and satisfies:

$$V(\vec{q},s) = \sum_a \prod_i P(a_i|q_i)\Big[R(s,\vec{a}) + \gamma \sum_{s',\vec{o},\vec{q}'} P(s',\vec{o}|s,\vec{a}) \prod_i P(q_i'|q_i,o_i) V(\vec{q'},s')\Big]$$

where

$$P(a_i|q_i) = \lambda_i(q_i)(a_i), P(q_i'|q_i,o_i) = \delta_i(q_i,o_i)(q_i') \text{ and } P(s',\vec{o}|s,\vec{a}) = P(s'|s,\vec{a})P(\vec{o}|s',\vec{a})$$

The value of $\vec{M}$ at the initial distribution is $V_{\vec{M}}(b) = \sum_s b(s)V_{\vec{M}}(\vec{q}_0,s)$. An example of a two node deterministic Moore machine for choosing actions a based on observations o is shown in Figure 6.1.

6.3.2 Mealy machines

Mealy machines cannot be used directly to represent policies since these assume that observations are generated before actions. To see this, recall that a Mealy machine generates an output only when it makes a transition. Thus, to generate an action, it first needs an observation.

This issue is easily solved by adding a new state s^*, new action a^* and new observation o^* so that the system starts at s^*. The rewards satisfy $R(s^*,a^*) = 0$, and $R(s^*, a) = -M$ if $a \neq a^*$ and $R(s, a^*) = -M$ if $s \neq s^*$ where M is a very large integer; the observations are $P(o^*|s, a^*) = 1$; and the transitions are $P(s^*|s^*, a) = 1$ if $a \neq a^*$ and $P(s|s^*, a^*) = b(s)$ where $b(\cdot)$ is the initial distribution of the POMDP.

The new POMDP is such that action a^* should be applied only once in state s^*. This application generates o^* as the first observation. Therefore, one can shift the stream of action-observations by one position so that it starts with observations followed by actions as required. Furthermore, since $R(s^*, a^*) = 0$, the value function of the new POMDP satisfies $V_\pi^{new}(b^*) = \gamma V_\pi(b)$ for any policy π, where b^* is the new initial distribution defined by $b^*(s^*) = 1$. A similar adjustment can be made in the DEC-POMDP case. Each agent would possess action a^* and observation o^* and the reward for any agent choosing any action other than a^* on the first step would be $-M$.

Let us illustrate the use of a Mealy machine $M = \langle Q, O, A, \tau, q_0 \rangle$ to control a (modified) POMDP. As we stated before, the first action to apply in the initial state s^* is a^* that generated the observation o^*. After this action, the resulting state is s with probability $b(s)$. At this point, M is used to control the process by generating a new action a and node q with probability $\tau(q_0, o^*)(q, a)$, the system changes state to s' and generates observation o with probabilities $P(s'|s, a)$ and $P(o|s', a)$, and the execution continues.

The infinite-horizon discounted value incurred by policy M when the initial state is s, the observation is o and the node of M is q is denoted by $V_M(q, o, s)$ and satisfies:

$$V(q, o, s) = \sum_{a,q'} P(q', a|q, o) \left[R(s, a) + \gamma \sum_{s',o'} P(s', o'|s, a) V(q', o', s') \right]$$

where $P(q', a|q, o) = \tau(q, o)(q', a)$. In this case, the value of M at b is $V_M(b) = \sum_s b(s) V_M(q_0, o^*, s)$. An example of a two node deterministic Mealy machine for choosing actions a based on observations o is shown in Figure 6.1.

For a given controller size, Mealy machines are more powerful than Moore machines in the following sense (a similar result also holds for DEC-POMDPs).

Theorem 6.3.1. *Let $P = \langle S, A, O, P, R, \gamma \rangle$ be a POMDP with initial state distribution b, and N a fixed positive integer. Then, for each stochastic Moore controller M_1 with N nodes, there is an stochastic Mealy controller M_2 with N nodes such that $V_{M_2}(b) \geq V_{M_1}(b)$.*

Proof. (sketch) This proof follows directly from the fact that a Mealy machine can represent a Moore machine exactly using the same number of nodes. As mentioned above, this is accomplished by constraining the Mealy machine to choose the same output for each input string. For POMDP control, this results in choosing the same action for each observation at a given node. Thus, there is always a Mealy machine that has the same value as a given Moore machine of any size N for any POMDP.

$\square$

We can also show that in some cases, a Mealy machine can represent higher-valued solutions with the same number of nodes. For instance, consider a POMDP whose optimal policy is reactive. That is, there is a mapping from observations to actions that produces the highest value. This can be achieved with a one node Mealy machine, but not with a one node Moore machine. No matter what action probabilities are defined, the Moore machine must use the same action distribution each step regardless of which observation is seen.

6.4 Mealy controllers for (DEC-)POMDPs

Until now, only Moore machines have been used to describe policies for POMDPs and DEC-POMDPs. As described in 2.3, 2.4, 3.2, 3.3, optimal approaches build up larger controllers over a number of steps using "backups" [15, 45], while approximate approaches typically seek to determine the action selection and node transition parameters that produce a high-valued fixed-size controller. Approximate POMDP methods include those utilizing gradient ascent [70], linear programming (LP) [85, 86] and nonlinear programming (NLP) [5]. Approximate DEC-POMDP algorithms have also been developed using heuristic search [110], linear programming [17] and nonlinear programming [5]. We conjecture that all of the above methods can be improved by adapting them to use Mealy machines. This would allow them to produce higher value with smaller controllers.

We first outline how optimal methods for centralized and decentralized POMDPs could be adapted to use Mealy machines. Approximate approaches can be improved by utilizing Mealy machines as well. We describe how bounded policy iteration (BPI) [86] and the NLP formulation from Chapter 5 can be adapted for solving POMDPs. We also discuss how decentralized BPI [17] and the NLP formulation for DEC-POMDPs can be adapted in a similar way that optimizes a controller for each agent while including constraints to ensure decentralized execution.

6.4.1 Optimal POMDP and DEC-POMDP methods

Optimal approaches would proceed exactly as before (since Mealy machines can represent Moore machines exactly), but after each "backup" an improvement step can be used to improve the value of the current controller. This improvement step is similar to the use of linear programming to increase the controller values that was shown to be effective by [15], but utilizing a Mealy controller formulation instead of a Moore formulation. That is, the action selection and node transition parameters are adjusted if improvements can be found for all states of the system (and all initial nodes of the other agents in the decentralized case). Because value is never lost, the linear programs used in BPI and DEC-BPI represent one such way of improving controller value as an improvement step. Backups and improvement steps can continue until an ϵ-optimal solution is found. Since Mealy machines can produce higher value for a given controller size, the improvement step is likely to provide higher-valued solutions and this can allow higher quality solutions to be found after the same number of backups.

6.4.2 Bounded policy iteration

As discussed in Section 2.4.1, bounded policy iteration (BPI) [86] utilizes linear programming to improve the action selection and node transition parameters of the controller. That is, it iterates through the nodes of the controller and seeks to find parameters that increase the value of that node for all states while assuming that the other parameters and values remain fixed. The algorithm terminates once no further improvements can be achieved. Improvements such as growing the controller or biasing the improvements based on information from the initial state can also be done, but for simplicity, we describe an adaptation of BPI that uses fixed-size controllers without bias.

The decentralized version of BPI (DEC-BPI) [17] is similar, but improvement must be found for all states of the system and all controllers nodes of the other agents. More details about this approach are discussed in Section 3.3.2.

POMDP case

The Mealy version of BPI breaks the problem in two parts: improving the first node and improving the other nodes. The parameters for the initial node q_0 can be improved by updating the probabilities of choosing actions and transitioning to other nodes from q_0. This is achieved with the following LP that has variables $\{x(a,q) = P(q,a|q_0,o^*) : q \in Q, a \in A\}$:

$$\max \quad \sum_{a,q} x(a,q) \sum_{s} b(s) \left[R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) V(q,o',s') \right]$$

$$\text{subject to} \quad \sum_{a,q} x(a,q) = 1\,, x(a,q) \geq 0 \text{ for } q \in Q, a \in A.$$

The parameters for other controller nodes q can be adjusted similarly with the following LP that has the variables $\{\epsilon\} \cup \{y(q', a, q, o) = P(q', a|q, o) : q' \in Q, a \in A\}$:

$$\max \epsilon$$

subject to

$$V(q, o, s) + \epsilon \leq \sum_{a,q'} y(q', a, q, o)\left[R(s, a) + \gamma \sum_{s',o'} P(s', o'|s, a)V(q', o', s')\right] \forall s \in S,$$

$$\sum_{a,q'} y(q', a, q, o) = 1, \; y(q', a, q, o) \geq 0 \text{ for } q' \in Q, a \in A$$

DEC-POMDP case

The Mealy version of DEC-BPI also breaks the problem in two parts: improving an agent's first node and improving the other nodes. This is still possible in the decentralized case because on the first step of the problem, all agents know the initial state so the decision can be centralized. After the first step, each agent is unaware of the observation seen and thus the action taken by the other agents. The parameters for agents' initial nodes $\vec{q^0}$ can be improved by updating the probabilities of choosing actions and transitioning to other nodes from $\vec{q^0}$. This is achieved with the following LP that has variables $\{x(\vec{a}, \vec{q}) = P(\vec{q}, \vec{a}|\vec{q^0}, \vec{o^*}) : \vec{q} \in \vec{Q}, \vec{a} \in \vec{A}\}$:

$$\max \quad \sum_{\vec{a},\vec{q}} x(\vec{a}, \vec{q}) \sum_s b(s)\left[R(s, \vec{a}) + \gamma \sum_{s',\vec{o'}} P(s', \vec{o'}|s, \vec{a})V(\vec{q}, \vec{o'}, s')\right]$$

$$\text{subject to} \quad \sum_{a_i,q_i} x(a_i, q_i) = 1\,, x(a_i, q_i) \geq 0 \text{ for } i \in I, q_i \in Q_i, a_i \in A_i.$$

The parameters for each agent i's other controller nodes q_i and observations o_i can be adjusted assuming fixed other agent controllers. This is completed with the following LP that has the variables $\{\epsilon\} \cup \{y(q'_i, a_i, q_i, o_i) = P(q'_i, a_i|q_i, o_i) : q'_i \in Q_i, a_i \in A_i\}$:

$$\max \epsilon$$

$$\text{subject to} \quad V(\vec{q},\vec{o},s)+\epsilon \le \sum_{\vec{a},\vec{q'}} y(q'_i,a_i,q_i,o_i)P(\vec{q'_{-i}},\vec{a}_{-i}|\vec{q}_{-i},\vec{o}_{-i})\Big[R(s,a)+$$

$$\gamma \sum_{s',\vec{o'},\vec{q'}} P(s',\vec{o'}|s,\vec{a})V(\vec{q'},\vec{o'},s')\Big] \; \forall s\in S, \vec{o}_{-i}\in \vec{O}_{-i}, q_{-i}\in \vec{Q}_{-i}$$

$$\sum_{a,q'} y(q'_i,a_i,q_i,o_i)=1,\; y(q'_i,a_i,q_i,o_i)\ge 0 \text{ for } q'_i\in Q_i, a_i\in A_i$$

Algorithmic benefits

In the Mealy machine version of BPI, we iterate through each pair $\langle q,o\rangle$ rather than just through each node q as is done with Moore machines. Similar to the Moore version, parameters are updated if better values can be found for all states of the system, but because each q and o is used, there are more opportunities for improvement and thus some local optima may be avoided. Also, the linear program used for the first node allows a known initial state to be directly incorporated in the solution. Lastly, because a Mealy machine is used, higher value can be represented with a fixed size, thus improving the potential of the algorithm for that size.

The mealy version of DEC-BPI has similar benefits. Start state information can also be incorporated because this is the lone step with common knowledge between all agents. Like the POMDP version, we expect better performance by incorporating Mealy machines due to increased representational power and the ability to make more incremental improvements.

6.4.3 Nonlinear programming

The nonlinear programming formulations for POMDPs and DEC-POMDPs are discussed in the previous chapter. They seek to optimize the action selection and node transition parameters for the whole controller or set of controllers in one step. This is achieved by allowing the values to change by defining them as variables in

For variables: $x(a,q')$, $y(q',a,q,o)$, $z(q,o,s)$

$$\text{Maximize} \quad \sum_s b(s) \sum_{a,q'} x(a,q') \left[R(s,a) + \gamma \sum_{s',o'} P(s'|s,a)P(o|s',a)z(q',o',s') \right]$$

subject to

$$\forall q,o,s \; z(q,o,s) = \sum_{a,q'} y(q',a,q,o) \left[R(s,a) + \gamma \sum_{s',o'} P(s'|s,a)P(o'|s',a)z(q',o',s') \right]$$

$$\sum_{a,q'} x(a,q') = 1, \;\; x(a,q') \geq 0 \;\forall a,q' \quad \sum_{q,o} y(q',a,q,o) = 1, \;\; y(q',a|q,o) \geq 0 \;\forall q,a,q',o$$

Table 6.1. Nonlinear program representing an optimal Mealy machine POMDP policy of a given size with variables $x(a,q)$ represent $P(q,a|q_0,o^*)$, $y(q',a,q,o)$ represents $P(q',a|q,o)$ and $z(q,o,s)$ represents $V(q,o,s)$.

the optimization. The fixed-size controller is then optimized with respect to a known initial state, $b(\cdot)$, using any NLP solver. An optimal solution of the NLP formulation would produce an optimal Moore controller or set of controllers for the given size. This is often intractable, causing approximate methods to be used.

POMDP case

The Mealy formulation of the NLP is shown in Table 6.1. This NLP defines an optimal Mealy controller of a given size by maximizing the value of the initial node at the initial state distribution. Constraints ensure proper values and probabilities. Its variables are $\{z(q,o,s) = V(q,o,s) : q \in Q, o \in O, s \in S\} \cup \{y(q',a,q,o) = P(q',a|q,o) : q,q' \in Q, a \in A, o \in O\} \cup \{x(a,q) = P(q,a|q_0,o^*) : q \in Q, a \in A\}$. Like the Moore case, this problem is difficult to solve optimally, but approximate solvers can be used to produce locally optimal solutions for a given size. Observe that for the NLP, one does not need to extend the sets A, O and S with a^*, o^* and s^* respectively.

For variables: $x(\vec{a},\vec{q}')$, $z(\vec{q},\vec{o},s)$ and $y(q_i',a_i,q_i,o_i)$ for each agent i

$$\text{Maximize} \quad \sum_s b(s) \sum_{\vec{a},\vec{q}'} x(\vec{a},\vec{q}') \left[R(s,\vec{a}) + \gamma \sum_{s',\vec{o}'} P(s'|s,\vec{a}) P(\vec{o}|s',\vec{a}) z(\vec{q}',\vec{o}',s') \right]$$

subject to

$$\forall \vec{q},\vec{o},s \; z(\vec{q},\vec{o},s) = \sum_{\vec{a},\vec{q}'} \prod_i y(q_i',a_i,q_i,o_i) \left[R(s,\vec{a}) + \gamma \sum_{s',\vec{o}'} P(s'|s,\vec{a}) P(\vec{o}'|s',\vec{a}) z(\vec{q}',\vec{o}',s') \right]$$

$$\sum_{a_i,q_i'} x(a_i,q_i') = 1, \quad x(a_i,q_i') \geq 0 \quad \forall i, a_i, q_i'$$

$$\sum_{q_i,o_i} y(q_i',a_i,q_i,o_i) = 1, \quad y(q_i',a_i|q_i,o_i) \geq 0 \quad \forall i, q_i, a_i, q_i', o_i$$

Table 6.2. Nonlinear program representing an optimal Mealy machine DEC-POMDP policy of a given size with variables $x(\vec{a},\vec{q})$ representing $P(\vec{q},\vec{a}|\vec{q}^0,o^*)$, $z(\vec{q},\vec{o},s)$ representing $V(\vec{q},\vec{o},s)$ and $y(q_i',a_i,q_i,o_i)$ representing $P(q_i',a_i|q_i,o_i)$ for each agent i.

DEC-POMDP case

The Mealy formulation for DEC-POMDPs is similar to that for POMDPs and is shown in Table 6.2. The value of the initial nodes of the agents' controllers is maximized at the initial problem state. The important thing to note is that the first actions and transitions can be centralized as all agents know this initial state and these actions and transitions take place only at this time. Constraints ensure that valid values and probabilities are produced. This NLP defines a set of optimal Mealy controllers of a given size. Its variables are $\{z(\vec{q},\vec{o},s) = V(\vec{q},\vec{o},s) : \vec{q} \in \vec{Q}, \vec{o} \in \vec{O}, s \in S\} \cup \{y(q_i',a_i,q_i,o_i) = P(q_i',a_i|q_i,o_i) : i \in I, q_i, q_i' \in Q_i, a_i \in A_i, o_i \in O_i\} \cup \{x(\vec{a},\vec{q}) = P(\vec{q},\vec{a}|\vec{q}^0,o^*) : i \in I, q_i \in Q_i, a_i \in A_i\}$. Note that variables are used for each agent's action selection and node transition parameters, ensuring that they are independent. Also, like the other NLPs, this problem is difficult to solve optimally, but approximate solvers can be used to produce locally optimal solutions for a given size. Also like the

POMDP case, the NLP one does not need to extend the sets A, O and S with a^*, o^* and s^* respectively.

Algorithmic benefits

As with the other approaches, Mealy machines provide a more efficient method to exploit a given controller size. The DEC-POMDP case includes the added benefit of a centralized first node, which will allow for higher solution quality to be achieved. For both NLP formulations, one drawback is that more nonlinear constraints are needed because the value function now includes three items (node, observation and state) instead of the two (node and state) required for Moore machines. This is exacerbated in the decentralized case as the value function now includes observations for each agent. In some cases, this may make the NLP more difficult to solve. In the next section we discuss how to use the structure of the Mealy machine in order to simplify the NLP and therefore minimize this problem and even make the Mealy formulations more efficient to solve than the Moore formulations.

6.5 Solving the Mealy machine more efficiently

When solving the linear program of BPI or the nonlinear program, the structure of the Mealy machine can be used to identify unreachable state-observation pairs or dominated actions that can be removed without any impact on optimality. The Mealy formulation can then be solved more efficiently while automatically determining important problem structure. Note that is cannot be done in the Moore case because these improvements depend on the current observation. We could add this structure to a Moore machine, but this would essentially result in transforming it into a Mealy machine.

For given transition and observation dynamics, certain state-observation pairs cannot occur; if $P(s', o|s, a) = 0$ for every s and a, then the pair $\langle s', o\rangle$ does not need to be considered and thus neither does $V(q,o,s')$. In the DEC-POMDP case, we can perform the same analysis to determine if tuples $\langle s', \vec{o} \rangle$ are reachable. In both cases, this may be able to drastically reduce the number of constraints in the linear or nonlinear program.

Actions can also be removed based on their values given the last observation seen. For example, for a robot in a grid, if the last observation seen is that there is a wall directly in front of it, then trying to move forward is unlikely to be helpful. To determine which actions are useful after an observation, upper and lower bounds for the value of actions at each state after an observation can be computed. Actions whose upper bound values are lower than the lower bound values of another action can be removed without affecting optimality. In the POMDP case, the upper bound can be found using heuristics such as the MDP value or crossproduct MDP [70] and random or previously found solutions can be used for lower bounds. In the DEC-POMDP case, the POMDP solution could serve as an upper bound, while again a previously found solution could represent a lower bound.

Once upper and lower bounds of the value function are computed, an upper bound (for the POMDP case) on choosing action a after being in state s and seeing o, $Q^U(o,s,a)$, is obtained with

$$Q^U(o,s,a) = R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) V^U(o',s')$$

while a lower bound $Q^L(o,s,a)$ is obtained similarly. An agent does not know the exact state it is in, and thus in general, an action must have an upper bound that is at most the lower bound for all states of the problem. That is, if there is a' such that $Q^U(o,s,a) \leq Q^L(o,s,a')$ $\forall s \in S$ then the action a is dominated by a' with respect to o because a higher-valued policy can always be formed by choosing a' rather than a. This can be refined by noticing that typically, a subset of states are reachable after observing o, requiring that value is examined only $\forall s \in S^o$ where S^o is the set of states possible after seeing o. In these instances, action a does not need to be considered, allowing the corresponding parameters to be removed from $\{P(q', a|q, o)\}$. The DEC-POMDP case is similar except the upper bound considers the best action choices for the other agents while the lower bound considers the worst action choices.

6.6 Experiments

We performed experiments on selected POMDP and DEC-POMDP benchmarks comparing approximate solutions using the Moore and Mealy LP and NLP formulations with other leading approximate algorithms. All NLP experiments were conducted on the NEOS server (http://neos.mcs.anl.gov) using the snopt solver. All Moore and Mealy experiments were initialized with random deterministic controllers and averaged over 10 trails. Because our Moore and Mealy implementations of BPI use fixed controllers instead of growing them, they will not perform as well (as seen in [85]). Nevertheless, the results show the difference in value produced by the use of Moore and Mealy machines. As described above, unreachable state-observation pairs and dominated actions were removed from the Mealy formulation. MDP and POMDP policies were used as upper bounds for POMDPs and DEC-POMDPs respectively, while reactive or previously found policies were used as lower bounds. Unless otherwise noted, experiments were performed on a 2.8 GHz machine with 4Gb of RAM. The code for HSVI2 and PERSEUS was used from the web sites of Trey Smith and Matthijs Spaan respectively and were run with a time limit of 90 minutes. For PERSEUS, 10,000 points were used and the average of 10 runs is provided. As experiments were conducted on different machines, results may vary slightly, but we expect the trends to remain the same.

Table 6.3 shows the results for three POMDPs benchmarks. The Aloha problem is a networking domain using the slotted Aloha scheme [25] and the tag problem involves a robot that must catch and tag an opponent [83]. Because the original tag problem stops after the opponent is successfully tagged and thus is not fully infinite-horizon, we also provide results for a version in which the problem repeats rather than stopping. A discount factor of 0.999 was used for the aloha domain and 0.95 was used for the tag problems.

In the first and third problems, Mealy machines produced by nonlinear programming provide the highest-valued solutions and generally use much less time than the other methods. In the second problem, the Mealy NLP formulation is competitive with the state-of-the-art in terms of quality and time. Linear programming (BPI) also showed improved performance by using a Mealy machine in two of the three problems. It is surprising that on the aloha problem, the Mealy formulation for BPI allows much larger controllers to be solved, but results in lower value. This is in contrast to the NLP results and is likely due to the problem structure which causes different local optima to appear in the Mealy formulation. That is, when linear optimization is used over each node and observation and different probability parameters are used the local optima will have a different structure. In the case of the aloha problem, the local optima cause BPI to "get stuck" at lower valued solutions. We suspect this is the exception rather than the rule (as shown by our experiments) and may not be the case with the full BPI algorithm which grows the controller when local optima are found.

Table 6.4 shows the results for three two agent DEC-POMDP benchmarks: the meeting in a grid problem [17], the box pushing domain [94] and the stochastic Mars rover problem [8]. On all of the DEC-POMDP domains a discount factor of 0.9 was used. To put the results from the Moore and Mealy machines into perspective, we also include results from heuristic policy iteration with nonlinear programming (HPI w/ NLP) from Section 7.1 and [15].

In all three problems, the Mealy machine solved by NLP obtains the highest values. This approach is able to do so with very concise controllers in a short amount

Algorithm	Value	Size	Time
Aloha $\|S\| = 90, \|A\| = 29, \|O\| = 3$			
NLP-Mealy	1221.72	7	312
HSVI2	1217.95	5434	5430
NLP-Moore	1211.67	6	1134
BPI-Moore[2]	1082.19	8	4592
BPI-Mealy[2]	1059.29	32	5294
PERSEUS	853.42	68	5401
Tag $\|S\| = 870, \|A\| = 5, \|O\| = 30$			
PBPI[1]	-5.87	818	1133
RTDP-BEL[1]	-6.16	2.5m	493
PERSEUS[1]	-6.17	280	1670
HSVI2[1]	-6.36	415	24
NLP-Mealy	-6.65	2	323
NLP-Moore	-13.94	2	5596
BPI-Mealy[2]	-17.50	25	5376
BPI-Moore[2]	-23.83	8	4583
Tag Repeat $\|S\| = 870, \|A\| = 5, \|O\| = 30$			
NLP-Mealy	-11.44	2	319
PERSEUS	-12.35	163	5656
HSVI2	-14.33	8433	5413
BPI-Mealy[2]	-19.02	9	5649
BPI-Moore[2]	-19.96	4	3137
NLP-Moore	-20.00	1	37

Table 6.3. Results for POMDP problems comparing Mealy and Moore machines and other algorithms. The size for the machines is the number of nodes in the controller. The size for other algorithms is the number of planes or belief points. The time is in seconds.

[1] These results are taken from PBPI: [56], RTDP-BEL: [19], PERSEUS: [108] and HSVI2: [105]

[2] These implementations of BPI use fixed controller sizes without start state bias in the Moore case.

Algorithm	Value	Size	Time
Meeting in a Grid $\lvert S\rvert = 16, \lvert A_i\rvert = 5, \lvert O_i\rvert = 2$			
NLP-Mealy	6.12	5	52
HPI w/ NLP	6.04	7	16763
NLP-Moore	5.66	5	117
BPI-Mealy[2]	3.49	123	5417
BPI-Moore[2]	3.22	19	5447
Box Pushing $\lvert S\rvert = 100, \lvert A_i\rvert = 4, \lvert O_i\rvert = 5$			
NLP-Mealy	138.76	4	2828
HPI w/ NLP	95.63	10	6545
NLP-Moore	50.64	4	5176
BPI-Mealy[2]	42.03	22	5169
BPI-Moore[2]	17.57	5	3181
Mars Rover $\lvert S\rvert = 256, \lvert A_i\rvert = 6, \lvert O_i\rvert = 8$			
NLP-Mealy	20.41	3	838
HPI w/ NLP	9.29	4	111
BPI-Mealy[2]	9.29	8	5253
NLP-Moore	8.16	2	43
BPI-Moore[2]	-3.30	2	2164

Table 6.4. Results for DEC-POMDP problems comparing Mealy and Moore machines and other algorithms. The size refers to the number of nodes in the controller and the time is in seconds.

[2] These implementations of BPI use fixed controller sizes without start state bias in the Moore case.

of time. Heuristic policy iteration with NLP (HPI w/ NLP) also performs well, but cannot outperform the NLP-Mealy results. The Mealy formulation using linear optimization (BPI-Mealy) is able to solve much larger controllers than the Moore formulation and always produces higher quality solutions (much higher in the second and third problems). Note that the HPI w/ NLP approach uses Moore machines as it's policy representation. We believe that using Mealy machines would improve the resulting value, but leave this for future work.

In a final experiment, we compare the quality of controllers obtained by utilizing fixed-size Mealy and Moore machines optimized by NLP on the DEC-POMDP benchmarks. Table 6.5 shows the results of the comparison. As can be seen, Mealy machines always achieve better quality for a fixed size. Similar results were also obtained in the POMDP benchmarks. By using approximate solvers, there may be problems for which this is not the case (as seen by the BPI results on the aloha problem), but we are encouraged by the results.

We can also see that in these problems much larger Moore controllers are needed to represent the same value as a given Mealy controller. In fact, the solver often exhausted resources before this was possible. In the box pushing and mars rover problems, the largest solvable Moore machines were not able to achieve the same quality as a one node Mealy machine. In the meeting in a grid problem, a four node Moore machine is required to approximate the one node Mealy machine.

It is also worth noting that similar value was often found using Mealy machines without state-observation reachability or action elimination. For instance, in the meeting in a grid problem three node controllers could be solved producing an average value of 5.94 (the same as the three node controller with the more efficient representation) in the somewhat longer time of 36s. The use of state-observation reachability and action elimination allows increased scalability, permitting increased value in problems that require larger controllers. Thus, even without using the controller structure to increase solution efficiency, the Mealy formulation can be beneficial.

6.7 Discussion

We presented a novel type of controller for centralized and decentralized POMDPs that is based on the Mealy machine. All existing controller-based algorithms can be adapted to use this type of machine instead of the currently used Moore machine. This requires minor alterations to current algorithms, but because Mealy machines are more powerful and possess additional structure, solution quality is likely to increase. We adapted two such algorithms and our experiments show that Mealy machines can

	Number of nodes				
Type	1	2	3	4	5
Meeting in a grid: $\lvert S\rvert = 16, \lvert A_i\rvert = 5, \lvert O_i\rvert = 2$					
NLP-Mealy	5.66	5.94	5.94	5.98	6.12
NLP-Moore	3.58	4.83	5.23	5.62	5.66
Box pushing: $\lvert S\rvert = 100, \lvert A_i\rvert = 4, \lvert O_i\rvert = 5$					
NLP-Mealy	120.90	133.85	138.69	138.76	x
NLP-Moore	-1.58	31.97	46.28	50.64	x
Mars rover: $\lvert S\rvert = 256,\ \lvert A_i\rvert = 6,\ \lvert O_i\rvert = 8$					
NLP-Mealy	19.84	20.41	20.41	x	x
NLP-Moore	0.80	8.16	x	x	x

Table 6.5. Results for Mealy and Moore machines of different sizes for DEC-POMDP benchmarks. A blank entry means that the controller of that size could not been computed given the resource restrictions of the NEOS server.

lead to higher-valued controllers. In fact, Mealy controllers optimized by nonlinear programming outperformed the state-of-the-art POMDP and DEC-POMDP case approaches in most instances. This shows the promise of this alternative representation for all controller-based approaches.

Mealy machines are beneficial for several reasons. First, they are more powerful than Moore machines, resulting in higher-valued solutions with the same representation size. Second, they possess special structure that can be straightforward to exploit. This includes start state information, knowledge from informative observations and a centralized first node for decentralized agents.

Some open questions remain. For instance, there may be some problems in which Moore controllers are found which outperform the Mealy controllers that are found. This is due to the fact that the linear and nonlinear programs for Mealy machines can be more difficult to solve. When all observation-state pairs are reachable and no actions can be eliminated, the Mealy formulations possess more variables and constraints. As a consequence, they may only be able to find solutions for small controllers, while Moore formulations may be able to solve sufficiently large controllers to produce higher values. Further examination is required to determine when this is the case, but as a first step, we can look at the success of the reachability analysis and action elimination. If a large number of observation-state pairs and actions can be removed, the Mealy formulations are likely to perform well. Even when this is not the case, the Mealy formulations may perform well due to their greater representational power, but further study is required for this to be determined.

CHAPTER 7

APPROXIMATE DYNAMIC PROGRAMMING APPROACHES FOR DEC-POMDPS

As mentioned in Section 3.3.1, another way to produce approximate solutions for DEC-POMDPs is to use approximate dynamic programming. These approaches build up finite-horizon trees without retaining all possibilities that are required to guarantee optimality. This results in more scalable, but no longer optimal algorithms. In this chapter, we discuss a method for approximate dynamic programming for infinite-horizon DEC-POMDPs using controllers as well as incorporating incremental policy generation (described in Chapter 4) into these approximate approaches. Lastly, we incorporate ideas from these two methods to create a heuristic incremental policy generation algorithm for infinite-horizon DEC-POMDPs.

7.1 Heuristic policy iteration

While the optimal infinite-horizon policy iteration method shows how a set of controllers with value arbitrarily close to optimal can be found, the resulting controllers may be very large and many unnecessary nodes may be generated along the way. This is exacerbated by the fact that the algorithm cannot take advantage of an initial state distribution and must attempt to improve the controller for any initial state. As a way to combat these disadvantages, we have developed a heuristic version of policy iteration that removes nodes based on their value only at a given set of centralized belief points. We call these centralized belief points because they are distributions over the system state that in general could only be known by full observability of the problem. As a result, the algorithm may no longer be optimal, but it can often produce more concise controllers with higher solution quality for a given initial state distribution.

7.1.1 Directed pruning

Our heuristic policy iteration algorithm uses sets of belief points to direct the pruning process of our algorithm. There are two main advantages of this approach: it allows simultaneous pruning for all agents and it focuses the controller on certain areas of the belief space. We first discuss the benefits of simultaneous pruning and then mention the advantages of focusing on small areas of the belief space.

The pruning method used by the optimal algorithm will not always remove all nodes that could be removed from all the agents' controllers without losing value. Because pruning requires each agent to consider the controllers of other agents, after nodes are removed for one agent, the other agents may be able to prune other nodes. Thus pruning must cycle through the agents and ceases when no agent can remove any further nodes. This is both time consuming and causes the controller to be much larger than it needs to be.

Like the game theoretic concept of incredible threats[1], a set of suboptimal policies for an agent may be useful only because other agents may employ similarly suboptimal policies. That is, because pruning is conducted for each agent while holding the other agents' policies fixed, polices that are useful for any set of other agent policies are retained, no matter the quality of these other agent policies. Some of an agent's policies may only be retained because they have the highest value when used in conjunction with other suboptimal policies of the other agents. In these cases, only by removing the set of suboptimal policies simultaneously can controller size be reduced

[1]An incredible threat is an irrational strategy that the agent knows it will receive a lower value by choosing it. While it is possible the agent will choose the incredible threat strategy, it is irrational to do so.

while at least maintaining value. This simultaneous pruning could further reduce controller sizes and thus increase scalability and solution quality.

The advantage of considering a smaller part of the state space has already been shown to produce drastic performance increases in POMDPs [56, 83] and finite-horizon DEC-POMDPs [94, 111]. For POMDPs, a problem with many states has a belief space with large dimensionality, but many parts may never be visited by an optimal policy. Focusing on a subset of belief states can allow a large part of the state space to be ignored without significant loss of solution quality.

The problem of having a large state space is compounded in the DEC-POMDP case. Not only is there uncertainty about the state, but also about the policies of the other agents. As a consequence, the generalized belief space which includes all possible distributions over states of the system and current policies of the other agents must be considered to guarantee optimality. This results in a huge space which contains many unlikely states and policies. The uncertainty about which policies other agents may utilize does not allow belief updates to normally be calculated for DEC-POMDPs, but as we will discuss, it can be done by assuming a probability distribution over actions of the other agents. This limits the number of policies that need to be considered by all agents and if the distributions are chosen well, may permit a high-valued solution to be found.

7.1.2 Belief set generation

As mentioned above, our heuristic policy iteration algorithm constructs sets of belief points for each agent which are later used to evaluate the joint controller and remove dominated nodes. To generate the belief point set, we start at the initial state and by making assumptions about the other agents, we can calculate the resulting belief state for each action and observation pair of an agent. By fixing the policies for the other agents, this belief state update can be calculated in a way very similar to that described for POMDPs in section 3.1.1. This procedure can be repeated from each resulting belief state until a desired number of points is generated or no new points are visited.

More formally, we assume the other agents have a fixed distribution for choosing actions at each system state. That is, if we know $P(\vec{a}_{-i}|s)$ then we can determine the probability any state results given a belief point and an agent's action and observation. The derivation of the likelihood of state s', given the belief state b, and agent i's action a_i and observation o_i is shown below.

$$\begin{aligned}
P(s'|a_i, o_i, b) &= \sum_{\vec{a}_{-i},\vec{o}_{-i},s} P(s', \vec{a}_{-i}, \vec{o}_{-i}, s|a_i, o_i, b) \\
&= \frac{\sum_{\vec{a}_{-i},\vec{o}_{-i},s} P(\vec{o}|s, b, \vec{a}, s')P(s', s, \vec{a}, b)}{P(o_i, a_i, b)} \\
&= \frac{\sum_{\vec{a}_{-i},\vec{o}_{-i},s} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a}, b)P(\vec{a}, s, b)}{P(o_i, a_i, b)} \\
&= \frac{\sum_{\vec{a}_{-i},\vec{o}_{-i},s} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a})P(\vec{a}_{-i}|a, s, b)P(\vec{a}, s, b)}{P(o_i, a_i, b)} \\
&= \frac{\sum_{\vec{a}_{-i},\vec{o}_{-i},s} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a})P(\vec{a}_{-i}|a_i, s, b)P(s|a_i, b)P(a_i, b)}{P(o_i, a_i, b)} \\
&= \frac{\sum_{\vec{a}_{-i},\vec{o}_{-i},s} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a})P(\vec{a}_{-i}|s)b(s)}{P(o_i|a_i, b)}
\end{aligned}$$

where

$$P(o_i|a_i, b) = \sum_{a_{-i},o_{-i},s,s'} P(\vec{o}|s, \vec{a}, s')P(s'|s, \vec{a})P(\vec{a}_{-i}|s)b(s)$$

Thus, given the action probabilities for the other agents, $-i$, and the transition and observation models of the system, a belief state update can be calculated.

7.1.3 Algorithmic approach

We provide a formal description of our approach in Table 7.1. Given the desired number of belief points, k, and random action and observation selection for each

Input: A joint controller, the desired number of centralized belief points k, initial state b_0 and fixed policy for each agent π_i.

1. Starting from b_0, sample a set of k belief points for each agent assuming the other agents use their fixed policy.
2. Evaluate the joint controller by solving a system of linear equations.
3. Perform an exhaustive backup to add deterministic nodes to the local controllers.
4. Retain nodes that contribute the highest value at each of the belief points.
5. For each agent, replace nodes that have lower value than some combination of other nodes at each belief point.
6. If controller sizes and parameters do not change then terminate. Else go to step 2.

Output: A new joint controller based on the sampled centralized belief points.

Table 7.1. Heuristic Policy Iteration for DEC-POMDPs.

agent, the sets of points are generated as described above. The search begins at the initial state of the problem and continues until the given number of points is obtained. If no new points are found, this process can be repeated to ensure a diverse set is produced. The arbitrary initial controller is evaluated and the value at each state and for each initial node of any agent's controller is determined. The exhaustive backup procedure is exactly the same as the one used in the optimal algorithm, but updating the controller takes place in two steps. First, for each of the k belief points, the highest-valued set of initial nodes is found. To accomplish this, the value of beginning at each combination of nodes for all agents is calculated for each of these k points and the best combination is kept. This allows nodes that do not contribute to any of these values to be simultaneously pruned. Next, pruning is conducted over each node of each agent's controller using the linear program shown in Table 7.2. If a distribution of nodes for the given agent has higher value at each of the belief points for any initial nodes of the other agents' controllers, it is pruned and replaced with

Variables: ϵ, $x(\hat{q}_i)$ and for each belief point b	
Objective: Maximize ϵ	
Improvement constraints:	$\forall b, q_{-i} \; \sum_s b(s) \left[\sum_{\hat{q}_i} x(\hat{q}_i) V(\hat{q}_i, q_{-i}, s) - V(\vec{q}, s) \right] \geq \epsilon$
Probability constraints:	$\sum_{\hat{q}_i} x(\hat{q}_i) = 1$ and $\forall \hat{q}_i \; x(q_i) \geq 0$

Table 7.2. The linear program used to determine if a node q for agent i is dominated at each point b and all initial nodes of the other agents' controllers. As node q may be dominated by a distribution of nodes, variable $x(\hat{q}_i)$ represents $P(\hat{q}_i)$, the probability of starting in node $\hat{q}$ for agent i.

that distribution. The new controllers are then evaluated and the value is compared with the value of the previous controller. This process of backing up and pruning continues while the controller parameters continue to change.

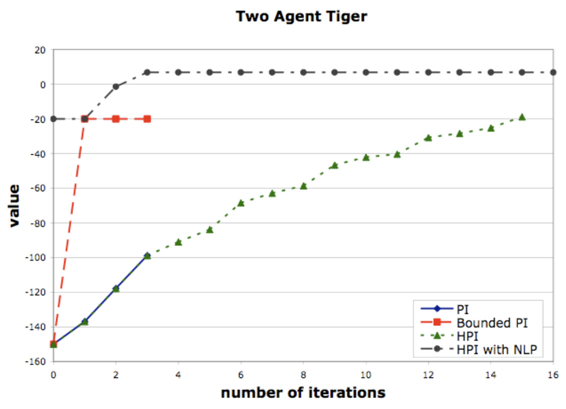
We can also improve the solution quality of the controllers after each step using a nonlinear program similar to the one discussed in Chapter 5 or 6. To accomplish this, instead of optimizing the controller for just the initial belief state of the problem, all the belief points being considered are used. A simple way to achieve this is to maximize over the sum of the values of the initial nodes of the controllers weighted by the probabilities given for each point. This approach can be used after each pruning step and may further improve value of the controllers.

7.1.4 Experiments

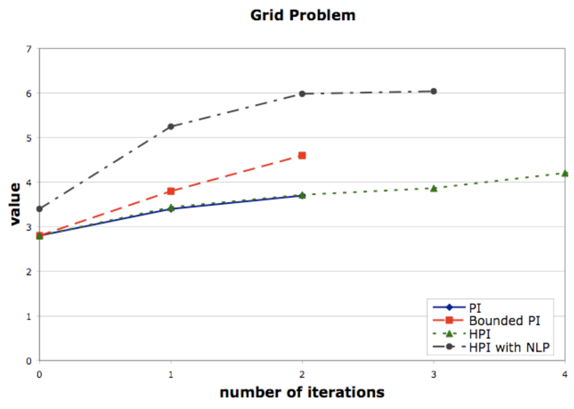
For these experiments, we initialized single node controllers for each agent with self loops. For each problem, the first action of the problem description was used. This resulted in the repeated actions of opening the left door in the two agent tiger problem, moving up in the meeting in a grid problem and turning left in the box pushing problem (all of which are described in the appendix). The reason for starting with the smallest possible controllers was to see how many iterations we could complete before running out of memory. The set of belief points for each problem was generated given the initial state distribution and a distribution of actions for the other agents. For the meeting in a grid and box pushing problems, it was assumed that all agents chose any action with equal probability regardless of state. For the two agent tiger problem, it was assumed that for any state agents listen with probability 0.8 and open each door with probability 0.1. This simple heuristic policy was chosen to allow more of the state space to be sampled by our search. The number of belief points used for the two agent tiger and meeting in a grid problems was ten and twenty points were used for the box pushing problem.

For each iteration, we performed an exhaustive backup and then pruned controllers as described in steps four and five of Table 7.1. All the nodes that contributed to the highest value for each belief point were retained and then each node was examined using the linear program in Table 7.2. For results with the NLP approach, we also improved the set of Moore controllers after heuristic pruning by optimizing a nonlinear program whose objective was the sum of the values of the initial nodes weighted by the belief point probabilities. We report the value produced by the optimal and heuristic approaches for each iteration that could be completed in under four hours and with the memory limits of the machine used. The optimal methods consisted of policy iteration and a version of policy iteration (PI) that improves the values of the controllers at each step using bounded policy iteration, which we call Bounded PI. The nonlinear optimization was performed on the NEOS server, which provides a set of machines with varying CPU speeds and memory limitations.

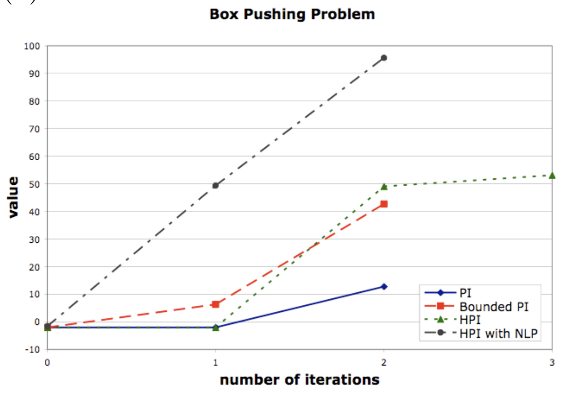
The values for each iteration of each problem are given in Figure 7.1. We see the heuristic policy iteration (HPI) methods are able to complete more iterations than the optimal methods and as a consequence produce higher values. In fact, the results from HPI are almost always exactly the same as those for the optimal policy iteration algorithm without bounded updates for all iterations that can be completed by the



(a)



(b)



(c)

Figure 7.1. Comparison of the dynamic programming algorithms on (a) the two agent tiger problem, (b) the meeting in a grid problem and (c) the box pushing problem. The value produced by policy iteration with and without bounded backups as well as our heuristic policy iteration with and without optimizing the NLP were compared on each iteration until the time or memory limit was reached.

optimal approach. Thus, improvement occurs primarily due to the larger number of backups that can be performed.

We also see that while incorporating bounded updates improves value for the optimal algorithm, incorporating the NLP approach into the heuristic approach produces even higher value. Optimizing the NLP requires a small time overhead, but substantially increases value on each iteration. This results in the highest controller value in each problem. Using the NLP also allows our heuristic policy iteration to converge to a six node controller for each agent in the two agent tiger problem. Unfortunately, this solution is known to be suboptimal. As an heuristic algorithm, this is not unexpected, and it should be noted that even suboptimal solutions by the heuristic approach outperform all other methods in all our test problems.

7.2 Incremental policy generation (IPG) with approximate dynamic programming

Incremental policy generation fits nicely with approximate dynamic programming algorithms because it increases efficiency of the backup step. To demonstrate this, we incorporate it into the state-of-the-art point-based incremental pruning (PBIP) algorithm for finite-horizon DEC-POMDPs and show significant performance gains. The results suggest that the performance of other approximate dynamic programming algorithms for DEC-POMDPs could be similarly improved by integrating the incremental policy generation approach.

7.2.1 Integrating with approximate dynamic programming

To improve the worst-case complexity of incremental policy generation, it can be incorporated into a memory bounded (MBDP-based) algorithm. MBDP-based approaches [95] make use of additional assumptions: (1) the joint state information, *i.e.* joint belief state, is accessible at planning time, although this information is not available to the agents at execution time; (2) only a small number of joint belief states are considered at each decision step. Because all current memory bounded algorithms use a form of exhaustive backup, incremental policy generation can be used to make the backup step more efficient. This can be accomplished by using assumption (1) to provide a set of known states for the incremental policy generation algorithm. That is, given a joint belief state, one can find all the possible next states that are reachable. This can be viewed as a special case of the incorporating start state information version described in Chapter 4, with the simplifying assumption that the system state is revealed after $T - t$ steps. Thereafter, one can select the subtrees after step t for agent i, $Q_i^{a,o}$, to generate set Q_i for each agent.

While incremental policy generation could be incorporated into any approximate dynamic programming approach, we chose to use the point-based incremental pruning (PBIP) algorithm [32]. This approach replaces the exhaustive backup with a branch-and-bound search in the space of joint policy trees. Like all MBDP-based approaches, the search technique builds $(t + 1)$-step joint policy trees for a small set of selected belief states, one joint policy tree for each belief state. Unlike the previous approaches, it computes upper and lower bounds of the partial $(t+1)$-step joint policy trees. With these bounds, it prunes dominated $(t + 1)$-step trees at earlier construction stages. Finally, the best T-step joint policy tree with respect to the initial belief state b_0 is returned. Because each subtree generated after taking a joint action and perceiving a joint observation can be any of the t-step joint policy trees from the last iteration, PBIP does not exploit the state reachability.

To incorporate incremental policy generation, we provide PBIP with the set $Q^{\vec{a},\vec{o}} \leftarrow \otimes_i Q_i^{a,o}$ of t-step joint policies that can be executed if we took joint action $\vec{a}$ one step before and any possible joint observation $\vec{o}$ is seen. With this, PBIP is now able to distinguish between subtrees executed after taking a given action and perceiving a given observation. Indeed, it is likely that subtrees that are good candidates for joint action joint observation pairs are useless for another pair.

7.2.2 Experiments

We now examine the performance increase achieved by incorporating the incremental policy generation approach into the leading approximate algorithm, PBIP. Only PBIP is used because it always produces values at least as high as IMBDP [94] and MBDP-OC [23] and is more scalable than MBDP [95]. It is worth noting that IPG can also be incorporated into each of these algorithms to improve their efficiency.

Horizon	PBIP	PBIP-IPG	Value
Meeting in a 3x3 Grid, $\|S\| = 81$, $\|A_i\| = 5$, $\|\Omega_i\| = 9$			
10	x	352s	3.85
100	x	3084s	92.12
200	x	13875s	193.39
Box Pushing, $\|S\| = 100$, $\|A_i\| = 4$, $\|\Omega_i\| = 5$			
10	46s	11s	103.22
100	536s	181s	598.40
500	2541s	1141s	2870.13
1000	5068s	2147s	5707.59
2000	10107s	4437s	11392.03
Stochastic Mars Rover, $\|S\| = 256$, $\|A_i\| = 6$, $\|\Omega_i\| = 8$			
2	106s	19s	5.80
3	x	71s	9.38
5	x	301s	12.66
10	x	976s	21.18
20	x	14947s	37.81

Table 7.3. Running time and value produced for each horizon using PBIP with and without incremental policy generation (IPG).

Common benchmarks which are described in the appendix are used and the choice for MaxTrees was fixed at 3 for each algorithm. Due to the stochastic nature of PBIP, each method was run 10 times and the mean values and running times are reported.

Experimental results are shown in Table 7.3 with PBIP and PBIP with the incremental policy generation approach (termed PBIP-IPG). It can be seen that PBIP is unable to solve the Meeting in a 3 by 3 Grid or Mars Rover problems for many horizons in the allotted time (12 hours). Incorporating IPG allows PBIP to solve these problems for much larger horizons. On the Box Pushing domain, PBIP is able to solve the problem on each horizon tested, but PBIP-IPG can do so in at most half the running time. These results show that while the branch and bound search used by PBIP allows it to be more scalable than MBDP, it still cannot solve problems with a larger number of observations. Incorporating IPG allows these problems to be solved because it uses action and observation information to reduce the number of trees

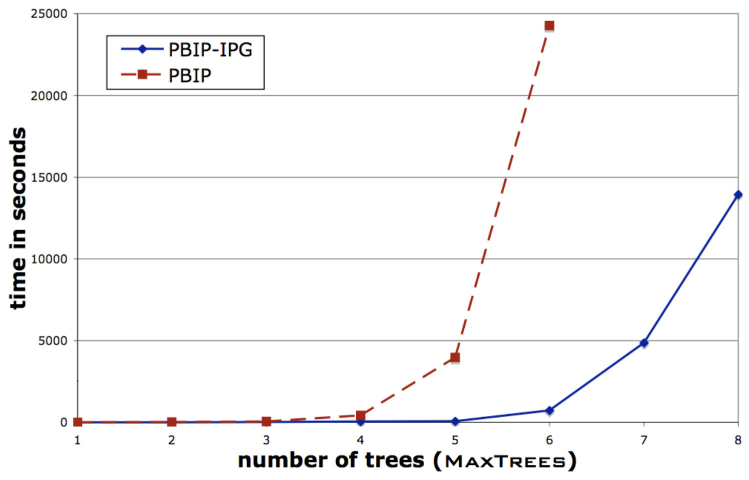


Figure 7.2. Running times for different values of MaxTrees on the Box Pushing problem with horizon 10.

considered at each step. Thus, the exponential affect of the number of observations is reduced by the IPG approach.

Figure 7.2 shows the running time for different choices of MaxTrees (the number of trees retained at each step of dynamic programming) on the Box Pushing domain with horizon 10. While the running time increases with the number of MaxTrees for both approaches, the time increases more slowly with the IPG approach. As a result, a larger number of MaxTrees can be used by PBIP-IPG. This is due to more efficient backups, which produce fewer horizon $t + 1$ trees for each horizon t tree. These results, together with those from Table 7.3 show that incorporating the incremental policy generation approach allows improved scalability to larger horizons and a larger number of MaxTrees in approximate dynamic programming approaches for DEC-POMDPs.

7.3 Approximate infinite-horizon incremental policy generation: Mixing IPG with heuristic policy iteration

We can mix the idea of sampling the state space from heuristic policy iteration and the state space reachability of incremental policy generation to construct a more

Algorithm 3: Heuristic incremental policy generation

```
input  : A set controllers for all agents, $C$, and a desired number of points, $m$
output: A set of backed up controllers, $\hat{C}'$
begin
    $P \leftarrow pointSet(C, m)$
    for each agent, $i$ do
        $C'_i \leftarrow incrPolGen(C, i)$
    $C' \leftarrow \cup_i \hat{C}'_i$
    for each belief point, $p$ do
        $\hat{C}'_p \leftarrow bestSet(C', p)$
    $\hat{C}' \leftarrow \cup_p \hat{C}'_p$
    $\hat{C}' \leftarrow prune(\hat{C}')$
    return $\hat{C}'$
end
```

scalable algorithm for infinite-horizon DEC-POMDPs called heuristic incremental policy generation. While this algorithm can no longer guarantee that ϵ-optimality is achieved, in general many more backups can be completed. This can result in higher-valued solutions being produced.

7.3.1 Algorithmic approach

An overview of this approach is provided in Algorithm 3. Our heuristic algorithm proceeds the same way as the optimal approach by using dynamic programming to improve the value of an initial solution (described in Section 4.3.2). On each step, we generate a set of belief states (state probability distributions) using action probabilities from the agents' current controllers. We then backup the controllers using incremental policy generation. Finally, we retain only those nodes that are part of the best set of controllers for each point.

Due to uncertainty in local observations and about the other agent's polices, the agents will not know what the actual state of the problem, but during execution, only a small part of the state space may be visited. By sampling in this space, many more nodes can be eliminated that are not required for a high valued solution. This allows the policy to be focused on states that are reachable given the current policies of the agents.

To generate a set of belief points for the agents, we first estimate the probability of each agent's action given the states of the problem, $\hat{P}(a_i|s)$. This is accomplished by estimating the probability that the node of an agent's controller is q given that the current state is s, $\hat{P}(q_i|s)$. This is found with an occupancy distribution by examining the probability the agent begins in the given node and state or the likelihood it transitions to the node and state later in the problem as determined by

$$\hat{P}(a_i|s) = \sum_{q_i} P(a_i|q_i)\hat{P}(q_i|s)$$

where

$$\hat{P}(q_i, s) = p_0(q_i', s') + \gamma \sum_{s,a_i} P(s'|s, a_i) \sum_{o_i} P(o_i|s', a_i) \sum_{q} P(a_i|q_i)P(q_i'|q_i, a_i, o_i)\hat{P}(q_i, s)$$

with $p_0(q_i, s)$ defined as the initial probably of node q_i and state s. The transition and observation models are estimated for each agent by choosing random actions for the other agents in the decentralized models. This quantity can then be normalized to produce $\hat{P}(q_i|s)$.

Once we have an estimate of the probabilities of other agent's actions at each state, we can determine the resulting state distributions for each action taken and observation seen. The probability of state s', given the belief state b, and agent i's action a_i and observation o_i, $P(s'|a_i, o_i, b)$, is then given by Equation 4.1 with a normalizing factor of $P(o_i|a_i, b)$. Thus, given the action probabilities for the other agents, and the transition and observation models of the system, a belief state update can be calculated.

Starting with the initial problem state, a set of belief points can then be generated. Once this occurs, the current controller can be backed up using incremental policy generation. After this, instead of retaining all nodes that are produced, only those that contribute to the solution with the highest value for each point are retained. This allows many more nodes to be removed at each step while focusing the solution on the states that are reached by the agents' policies. Finally, pruning dominated nodes is conducted as described for policy iteration. The process of generating a new set of points, backing up the controller and retaining the best nodes continues until the agents' controllers no longer change.

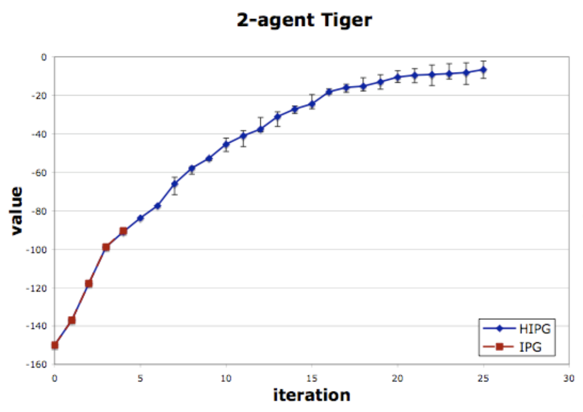
7.3.2 Experiments

The comparison between the optimal incremental policy generation approach (IPG) with the heuristic approach (HIPG) is shown in Figure 7.3. Due to randomness in point generation, the heuristic algorithm was run 10 times and mean values are reported. The maximum and minimum values for each iteration are shown with error bars. Also, the number of points used for each problem was 20.

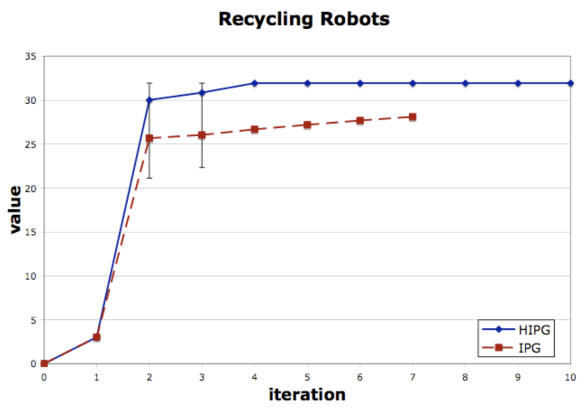
Using the heuristic approach, value is improved in each problem. In the tiger problem, Figure 7.3(a), the same value as optimal approach is found for the first few steps, but increased value can be found due to better scalability. In the recycling robot problem, (b), a higher valued solution is more quickly found and converged to, while the optimal approach cannot reach that value before resources are exhausted. Lastly, in the box pushing problem, (c), the two algorithms alternate between producing the highest value, but due to increased scalability, the heuristic approach eventually outperforms the optimal approach.

7.4 Discussion

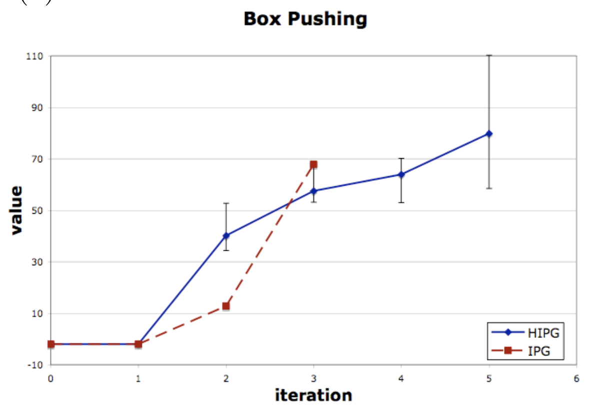
In this chapter, we presented an approximate dynamic programming approach for infinite-horizon DEC-POMDPs and methods for incorporating incremental policy generation into approximate dynamic programming approaches for both finite and



(a)



(b)



(c)

Figure 7.3. Comparison of the heuristic and optimal incremental policy generation algorithms on (a) the two agent tiger, (b) recycling robot and (c) the box pushing problems.

infinite horizons. These methods increase scalability by sampling the state space and other agent policies. Incremental policy generation can improve the efficiency of these methods as well as any other methods that utilize dynamic programming. Together these approaches can provide the scalability required to solve large DEC-POMDPs.

Thus, sampling of the state space and other agent policies can increase scalability and solution quality, but requires accurate estimates of policies. This can be given by other less sophisticated solution methods (as is done in the MBDP-based approaches) or by the current solutions (as in our heuristic incremental policy general algorithm). Both approaches has been shown to work well in many cases, but are domain dependent. This is a promising area of research that will continue to improve as other methods are found to reduce the large search space that results from solving decentralized POMDPs.

CHAPTER 8

ACHIEVING GOALS IN DEC-POMDPS

Besides DEC-POMDP problems that terminate after a known set of steps (finite-horizon) and those that continue for an infinite number of steps (infinite-horizon), there are those that terminate when certain conditions are met. We may not know how many steps are required, but the agents are attempting to achieve some goal or set of goals. We first give an overview of current goal based algorithms and the relevant previous work. We then present a model for indefinite-horizon DEC-POMDPs with action-based termination and a dynamic programming algorithm to determine optimal solutions. We then describe the more general goal-directed model and an approximate algorithm that is able to take advantage of goals in DEC-POMDPs. We provide a bound on the likelihood that an ϵ-optimal solution is found and show experimentally that this approach can produce high quality solutions on a range of domains.

8.1 Overview

Many DEC-POMDP domains involve a set of agents completing some task or achieving a goal. The number of steps needed to achieve the goal is unknown, but the problem ends once it has been completed. Examples of these domains include agents targeting and catching an object, meeting in an environment, cooperatively exploring and experimenting in an environment, or moving an object or sets of objects. In each case, agents have to cooperate while using only local information to finish the task in an efficient manner. For instance, meeting in an environment requires the agents to choose a central location and each determine a set of paths that will allow the location to be reached in the quickest manner while also considering what paths the other agents may take. This is a general class of problems that has many natural real world applications.

In goal based problems, the number of steps until completion depends on the actions that are taken and not a fixed number of steps as in finite-horizon problems. Solving these problems as infinite-horizon essentially involves solving the same problem over and over again. Also, infinite-horizon solutions are susceptible to changes in the discount factor, which maintains a bounded value. We can instead model these problems as indefinite-horizon, which terminate after some unknown number of steps. We base our indefinite-horizon representation on the action-based termination model used in POMDPs [46]. Our model assumes that terminal actions can be taken at any step and cause the problem to stop. In order to guarantee that problems terminate after a finite number of steps and optimal solutions can be found, we must also assume that all rewards are negative except for those generated from the terminal actions. Under these assumptions, we provide an optimal dynamic programming algorithm for indefinite-horizon DEC-POMDPs with action-based termination.

We can relax some of the assumptions made for action-based termination to allow us to model a wider range of problems. While we can no longer guarantee that these problems will terminate after a finite number of steps, the presence of goals allows us to implement a sampling approach that can take advantage of the added structure to produce high quality infinite-horizon solutions. We present an approximate algorithm for solving a class of *goal-directed* DEC-POMDPs, which have a well defined termination condition and any type of reward function. We show that this sampling approach can outperform the state-of-the-art approximate infinite-horizon DEC-POMDP algorithms on a number of goal-directed problems.

8.2 Related work

Goldman and Zilberstein [42] modeled goal oriented DEC-POMDPs as finite-horizon problems with a set of goal states and negative rewards in all non-goal states. Adding these goal states to the finite-horizon problem was shown to not change the complexity of general DEC-POMDPs (NEXP-complete [16]) and independent transition and observation (IT-IO) DEC-MDPs (NP-complete [14]) when no additional assumptions were made. In IT-IO DEC-MDPs, Goldman and Zilberstein also assume that each agent also has access to a no-op action in goal states, where the agent essentially waits for the other agents to reach the goal. When a unique goal state was assumed or when a single goal is always known to have higher value than the others, the IT-IO case was shown to be P-complete.

Problems that repeat or continue in some way after goals have been reached can be solved using infinite-horizon algorithms. The infinite-horizon DEC-POMDP approaches described in these thesis can be used to solve this problems. Approximate approaches include linear programming, best-first search and the controller-based approaches described in Chapters 5, 6 and 7.

8.3 Indefinite-horizon DEC-POMDPs

In this section, we first discuss the indefinite-horizon model as well as indefinite-horizon approaches for POMDPs. We then provide the DEC-POMDP model and an algorithm to solve it, including a proof that this approach will provide an optimal indefinite-horizon solution.

8.3.1 Overview

In many benchmark and real world problems, a set of agents must achieve some goal in order to receive maximum value. Benchmark problems include the two agent tiger problem [72], the box pushing problem [94] and the meeting in a grid problem

[17] (which are described in the appendix). After this goal has been reached, the problem either resets (tiger and box problems) or the problem effectively stops (grid problem). These problems can be modeled as a type of indefinite-horizon problem in which the problem stops after an unknown number of steps after some goal condition has been reached. This is a natural and expressive class of problems in which a set of agents must complete some task such as navigate to a goal location or simultaneously or independently choose a goal action.

Goldman and Zilberstein discussed goal-oriented problems, but modeled them as finite-horizon and required the presence of no-op actions when agents' transitions are independent. Other approaches have ignored the presence of a goal and have used general DEC-POMDP algorithms to find solutions. This causes the problem to be solved over and over again and the solution to be dependent on the horizon or discount chosen. The alternative is to model these problems as indefinite-horizon, which removes the need to either know the proper horizon beforehand or choose an often meaningless discount factor.

Indefinite-horizon models for POMDPs have been proposed by Patek [79] and Hansen [46]. Patek describes *partially observed stochastic shortest path problems*, which make the following assumptions: (1) existence of a policy that terminates with probability one, (2) any policy that does not guarantee termination has infinite negative reward and (3) termination is fully observable. He proves that there is a stationary optimal policy, and in the limit value iteration and policy iteration will converge to such a policy. Hansen proposes the *action-based termination* model which, in addition to Patek's assumptions also assumes (4) a set of terminal actions, (5) all non-terminal actions have negative reward, and (6) terminal actions can have a positive or negative reward that depends on the state. The action-based termination model allows simpler algorithms as proper policies, which guarantee termination, are easily found. This greatly simplifies the use of value and policy iteration with the model.

8.3.2 Indefinite-horizon framework and solution

Hansen's action-based termination model can be extended to DEC-POMDPs with a minor change. We will require that each agent has a set of terminal actions and the problem stops when one of these is taken by each agent simultaneously. We also require negative rewards for non-terminal actions which implies the other assumptions made by Patek and Hansen. We could have assumed the problem ends when any agent chooses a terminal action without adding much complication to the algorithm and analysis. We could have also used a set of fully observable goal states that the agents must reach, but this complicates matters as an agent's actions depend on only local information, but reaching the goal may depend on certain information being observed by all agents. Thus, even if the goal state is completely observable, estimation of the intermediate states may not possible with local information, making transitions to the goal unlikely or impossible. Therefore, it is an open question whether it is decidable to find an ϵ-optimal indefinite-horizon DEC-POMDP policy without assumptions that are more restrictive than those made by Patek.

We can define an indefinite-horizon DEC-POMDP with action-based termination in the same way as a finite-horizon DEC-POMDP in that a discount factor is not used, but with the added difference that there is no specified horizon. Instead, the problem stops when a terminal action is chosen by each agent. Assumptions (1)-(6) then allow us to describe a dynamic programming algorithm that returns an optimal solution for indefinite-horizon DEC-POMDPs in a finite number of steps. This algorithm is an extension of the indefinite-horizon POMDP algorithm presented by Hansen [46] and the optimal finite-horizon DEC-POMDP algorithm presented by Hansen et al. [47]. We can build policy trees for each agent by first considering only terminal actions on the last step and building up the tree by adding only non-terminal actions on each successive step. This process of generating next step trees and pruning continues until we can be sure that the optimal set of trees has been produced.

Unfortunately, the decentralized case does not allow an error bound to be calculated in the form of a Bellman residual as is used by the POMDP method. The Bellman residual is found from the maximum difference in value for all belief states, but in DEC-POMDPs, the value of an agent's policy depends not only on the state of the system, but also on the policies chosen by the other agents. Thus, constructing a Bellman residual in the DEC-POMDP case is an open question. Instead, we can calculate an upper bound on the horizon of any optimal policy and then generate all trees up to that bound using dynamic programming. A set of policy trees that provides the highest value for the given initial state distribution, regardless of horizon, is an optimal solution for the problem.

We first show how to calculate the upper bound on the horizon of an optimal policy.

Lemma 8.3.1. *An optimal set of indefinite-horizon policy trees must have horizon less than* $k_{max} = (R_{now} - R^{T}_{max})/R^{NT}_{max}$.

where R^{T}_{max} is the value of the best combination of terminal actions, R^{NT}_{max} the value of best combination of non-terminal actions and R_{now} is the maximum value attained by choosing a set of terminal actions on the first step given the initial state distribution.

Proof. This proof centers on the fact that the reward for all non-terminal actions is negative. After k steps of dynamic programming, the maximum value attained by any set of policies is $(k-1)R^{NT}_{max} + R^{T}_{max}$. If this value is less than the best value of immediately choosing terminal actions at the initial state distribution, R_{now}, then the agents are better off ending the problem on the first step. Thus, when

$(k-1)R^{NT}_{max} + R^{T}_{max} \leq R_{now}$ we know that it is more beneficial to choose some combination of terminal actions at an earlier step. Any set of trees with horizon higher than k will have even lower value, so $k_{max} = (R_{now} - R^{T}_{max})/R^{NT}_{max}$ □

This allows us to bound the horizon, and then we can just choose the set of trees for any horizon up to k_{max} that provides the highest value for the initial state distribution.

Theorem 8.3.2. *Our dynamic programming algorithm for indefinite-horizon POMDPs returns an optimal set of policy trees for the given initial state distribution.*

Proof. This follows almost directly from Lemma 8.3.1. Our dynamic programming algorithm solves the indefinite-horizon problem up to horizon k_{max}, which we have shown represents an upper bound on the horizon of optimal policies. Because our algorithm represents an exhaustive search (except for pruning) in policy space up to horizon k_{max}, an optimal optimal set of policy trees must be contained in the resulting sets of trees. Pruning does not remove any policy trees that will be useful in any optimal policy (because it is only done when there is another tree that is always better for all states and trees or subtrees of the other agents, as shown in [47]). A set of trees that provides the highest value for the given initial state then represents an optimal policy for the indefinite-horizon DEC-POMDP. □

If we choose R_{now} to be the maximum value attained by choosing a set of terminal actions for any possible initial state distribution, the resulting sets of policy trees will include an optimal set for any initial distribution. Also, if we allow any agent to terminate the problem, we must include both terminal and non-terminal actions on the first step of the algorithm and then ensure that at least one agent chooses a terminal action on the last step of the problem.

Many DEC-POMDP domains can be naturally modeled as indefinite-horizon with action-based termination. One example is the search for a moving target problems discussed by Hansen. When adapted to the decentralized case, either both agents must both simultaneously "catch" the target or one *is* the target and the goal is to meet by using local information. The later case is exactly modeled by the meeting in a grid benchmark DEC-POMDP problem [17]. The terminal actions for this grid problem consist of the agents deciding to stay in the same place when they believe they are in the same grid square as the other agents. The multiagent tiger benchmark problem [72] in which the problem resets when any agent chooses a door that it believes the tiger is behind is also naturally captured by this model. The terminal actions for the tiger problem are choosing to open a door. Many other robot navigation and task oriented problems can be similarly represented using action-based termination.

8.4 Goal-directed DEC-POMDPs

If we relax assumptions (1)-(6) given above, but retain the notion of a goal, we may no longer be able to provide an optimal solution that ends after a bounded number of steps. Instead, we have a class of general infinite-horizon DEC-POMDPs that has some structure which can allow us to produce high quality approximate solutions by focusing on policies that achieve the goal. In this section, we first give an overview of this class of DEC-POMDPs and then provide a sample-based algorithm for generating solutions.

8.4.1 Overview

We will discuss a class of infinite-horizon DEC-POMDPs with specified goal criteria. We call these problems *goal-directed* and require that the problem terminates or resets if any of the following hold:

- the set of agents reach a global goal state
- a single agent or set of agents reach local goal states

- any chosen combination of actions and observations is taken or seen by the set of agents

These assumptions permit a very general class of problems that is common in benchmark domains and real world applications. Because the two agent tiger [72] and meeting in a grid problems [17] fit in the action-based termination model, they also fit here, but the box pushing problem [94] is also included in this class. The box pushing problem resets when at least one agent pushes a small box or when both agents push a large box into a goal row. Thus, there are multiple goal states that can be achieved by the agents, but no terminal actions.

Real world problems include those discussed for indefinite-horizon DEC-POMDPs as well as those with more general goals and reward structure. For instance, consider a set of agents that must perform different experiments at certain research sites. Some of these sites may require multiple agents performing some experiment together in order to get the most scientific value, while other sites may require a specific tool be used by a single agent. Positive rewards are given for successfully performing experiments at each site and the task is completed when all sites have been experimented on. This is a version of the Mars rover problem discussed in [14]. Many other problems have these characteristics in which the agents must complete some task with clear termination conditions.

8.4.2 Algorithmic approach

In order to solve these problems, we develop a sampling approach for DEC-POMDPs. Sampling has been effective in reducing the necessary search space in many planning problems. For instance, in POMDPs, work has primarily dealt with sampling belief states and generating policies for these sampled states. Methods have been developed which can bound the sample size necessary for determining ϵ-optimal policies [83] and empirically, very high quality policies can be found with a much smaller set of belief states [83, 104]. Unfortunately, it is not straightforward to extend these methods to DEC-POMDPs. This is due to the fact that a shared belief state can no longer be calculated and policies for each agent must be evaluated in the context of policies for all other agents and executed in a decentralized manner.

In our approach, we use the presence of goals to construct an algorithm which samples the policies of the agents and produces an infinite-horizon solution. The policies for all agents are sampled simultaneously by using trajectories which begin at the initial state and terminate when the goal conditions are met. Because planning is conducted in a centralized fashion, it is always known when the goal is reached. If the goal can always be achieved, the goal conditions focus and limit the possible trajectories and with sufficient samples we could construct all possible policies for the set of agents. This would result in an exhaustive policy search and an optimal joint policy could then be chosen from the set of possibilities.

When a subset of the possible trajectories is generated, a partial policy search is conducted and we must decide what is done when observations are seen that are not observed in the trajectories. Also, due to decentralized execution of policies, even if the goal is reached during the trajectories, if an unseen observation occurs it could cause the agents to fail to reach the goal. Because it may not be possible to ensure the goal is reached, we must ensure a valid policy is generated for any number of steps. For these reasons, we choose to generate finite-state controllers from the sampled trajectories. The controllers are optimized to provide policy choices for observations that are and are not part of the trajectories and action choices for histories of any possible length. The controllers will also allow the best decentralized policies to be chosen while considering the possible choices of the other agents.

Algorithm 4: Goal-directed controller generation

```
input  : The total number of samples desired, $n_{total}$, and the number of those
         to retain, $n_{best}$
output: A set of controllers for the agents, $\hat{Q}$
begin
    $trajectories \leftarrow getTrajectory(b_0, goal, n_{total})$
    $bestTraj \leftarrow chooseBest(n_{best}, trajectories)$
    for each agent, $i \in I$ do
        $Q_i \leftarrow buildController(bestTraj_i)$
        $\hat{Q}_i \leftarrow reduceController(Q_i)$
    $change \leftarrow true$
    while $change$ do
        $change \leftarrow findBestActions(\hat{Q})$
        $change \leftarrow findTransitions(\hat{Q})$
    return $\hat{Q}$
end
```

8.4.3 Algorithm description

Our approach is given in Algorithm 4. First, we generate a set of *trajectories* which consist of action and observation sequences for each agent that begin at the initial state distribution and end when the goal is achieved. Actions are chosen randomly by each agent and observations are chosen based on their likelihood after the set of actions has been taken. Trajectories that have not reached the goal after a given cutoff are discarded.

The trajectories generate rewards for the set of agents at each step until the goal is reached. These can be summed by $R(b_0, \vec{a}_0) + \ldots + \gamma^T R(b_{goal}, \vec{a}_{goal})$ where the reward is weighted by the state probabilities $R(b, \vec{a}) = \sum_s b(s)R(s, \vec{a})$ and T is the number of steps used to reach the goal. A heuristic that considers the value of trajectories can be used to focus and reduce the number considered. This is done by choosing some number of trajectories that produce the best value and retaining only those. While some low value trajectories may be valuable to determine what actions should be taken in certain cases (such as when actions fail to have the desired outcome), in general, the highest valued trajectories are the most useful for generating high-valued policies.

A controller for each agent can then be generated from these trajectories. This is accomplished by first creating a tree whose root node represents the initial state and leaves represent the goal being achieved. Starting from the root, new nodes are created for each action that was taken at the first step of the trajectories. From these nodes, new nodes are then created for each observation that was seen after the given action was taken. This process continues until the goal has been reached, resulting in a tree which represents any action and observation sequence that is present in any of that agent's trajectories. An example of a set of trajectories and an initial controller is shown in Figure 8.1. It is for the possible policies that are generated at this step for which we derive a bound in our analysis.

To make better use of limited space and trajectories, this tree can then transformed into a controller which is then optimized. A controller is constructed by adding transitions after the goal has been reached into an absorbing node or back to the root node depending on the problem dynamics. To ensure that an action is defined for any possible action and observation history we can arbitrarily assign controller transitions for observations that are not a part of any trajectory. For instance, the controller can transition back to the beginning or stay in place for these unseen observations. These transitions can be improved during the optimization phase.

Because these controllers are generated from trees, as soon as two trajectories differ, the controller branches out and a separate path is created until the goal is reached. This is true even if the trajectories become the same for later sequences. For this reason, we can reduce the size of the controllers by starting from the last node of each trajectory and determining if the same actions are possible, the same observations are seen and the same node is transitioned to. If this is the case, the nodes can be merged. This can greatly reduce the size of the controllers without reducing their expressiveness.

We then need to choose the best of the possible actions at each node of the controller. This can be done by any search method such as branch and bound. The different combinations of action choices are evaluated for the set of agents and the one with the highest value is chosen. In many cases, we can stop after this search and have a high quality infinite-horizon policy for each agent, but it may also be beneficial to adjust the controller transitions.

A more compact or higher valued controller may be able to be constructed if the tree-like structure is adjusted and transitions between different nodes are used. Because it is often intractable to search through all possible controller transitions for all agents, we use a heuristic method that allows transitions to be changed within a controller. This approach tests each node and observation to determine if a higher value can be attained by transitioning to a different resulting node while keeping other transitions in the agent's controller and all other agents' controllers fixed. If an improvement is found, the transition is updated for that observation. We can continue optimizing the action choices and transition choices for the agents until no further improvements can be made.

8.4.4 Example

An example of policy generation for an agent is shown in Figure 8.1. This example is motivated by a version of the meeting in a grid problem in which the agents must navigate to a certain grid location while movements can slip, resulting in the agent staying in place. We begin with a set of action and observation sequences which represent trajectories starting at an initial state and end when the goal condition is satisfied (labeled g throughout). These trajectories represent different paths that the agent took to achieve the goal. While it was possible that the agent reached the goal

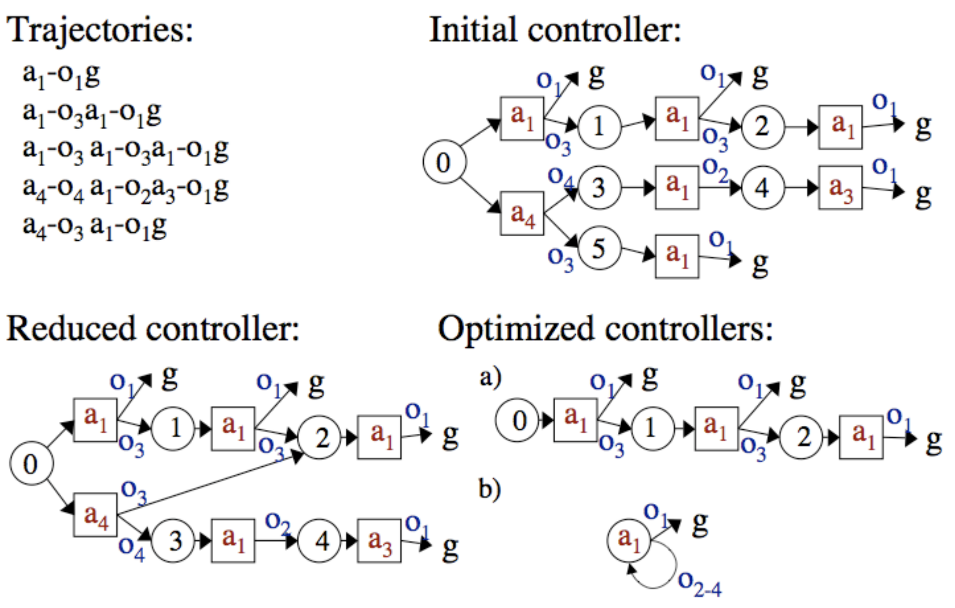


Figure 8.1. An example of a controller generated from a set of trajectories. The trajectories all begin at an initial state and end when a goal condition has been satisfied. The initial controller is then created from these trajectories and redundant nodes are combined to produce the reduced controller. This controller is then optimized by evaluating it in conjunction with the controllers of the other agents and optimizing the action choices and transition choices as shown in a) and b) respectively.

location after taking one action, because movements are noisy and multiple paths exist, other instances required multiple iterations of that action or a different set of actions.

The trajectories are then used to generate an initial controller. For each trajectory, there is a path in the controller that reaches the goal. Circles are used to represent nodes, squares represent action choices and observations are labeled above the given transition arrow. This initial controller is a tree, except we can consider transitioning to any node once the goal has been reached. We can then reduce the controller by noticing that node 5 and node 2 are equivalent. That is, each takes only action a_1 and after observing o_1 transitions to the goal. Thus, the two nodes can be merged, reducing the controller size from 6 to 5.

The optimization phase further reduces the controller size by choosing actions and adjusting transitions. The action choices that provide the highest value are chosen in part a, producing a 3 node controller with fixed actions. The transitions after each observation, whether part of the trajectories or not, are then optimized, resulting in a single node that chooses action a_1 until the goal is reached. The optimized controller is very simple and provides a high valued policy for this version of the meeting in a grid problem.

8.5 Analysis

The initial policy for the controller is built from sampled trajectories of the domain. With an unlimited number of samples and a bound on the horizon, by choosing the best actions, the initial controllers would be optimal. More interestingly, it is possible to bound the error of the controllers obtained, based on the number of samples used to generate them. We show that bounds developed in the context of POMDPs [60] may be adapted to our setting. The goal is to show that with probability at least $1 - \delta$ we can construct controllers that have value within ϵ of the optimal set of controllers.

These bounds are based on an analysis similar to that of learning-theoretic bounds, with a few modifications. In machine learning, typically all samples may be used in evaluating every hypothesis. The bounds then rely on the large number of samples available for testing each hypothesis. This is not the case in our setting. When evaluating a deterministic joint policy π, only trajectories with the same actions as π for the observations that are seen may be used. Also, as mentioned above, because policies are produced from the trajectories, actions will not be specified for all observation sequences. This does not present a problem in our analysis, since the set of samples used to define the trajectories is identical to the set of samples used to define the policies.

We adopt the analysis from [60], section 5.1, to derive the bounds. While this analysis produces somewhat loose bounds, it lays the foundation for more sophisticated bounds, which remains the subject of future work. We use $\pi \in \Pi$ to denote a joint deterministic policy from the set of all possible policies. We also use A_i to denote the set of available actions for agent i, and $A = \prod_i^n A_i$ with $|A| \geq 2$. We assume an upper bound on the number of steps required to reach the goal, denoted by T. The set of available trajectories is $\mathcal{H}$, assuming that the actions taken are chosen randomly with a uniform probability, and independently in each step. Finally, let $V_{\max}$ be the maximal difference between the values of the policies in Π. Using this notation, we show that if the optimal policy is π^*, the policy obtained from the samples is $\tilde{\pi}(\mathcal{H})$ and given a sufficient number of samples, then for any ϵ we can bound:

$$\mathrm{P}\left[V^{\pi^*}(s_0) - V^{\tilde{\pi}(\mathcal{H})}\right] \geq \epsilon$$

We highlight the lemmas that are different from [60] and provide the corresponding lemma or theorem numbers. The proofs are straightforward and thus omitted. The following lemma describes the probability of generating a trajectory, h, that is consistent with any joint policy π. We use $acc_\pi(h)$ to denote the presence or absence of this consistency.

Lemma 8.5.1 (An adaptation of Lemma 5.1)**.** *For any joint deterministic policy:*

$$\mathrm{P}\left[acc_\pi(h) = 1\right] = \frac{1}{|\mathcal{A}|^T}$$

Lemma 5.2 states that the probability that a (joint) policy π produces a trajectory h is equal to the probability that a random policy produces h given that h is consistent with π. We can also bound the probability that there is a joint policy that is consistent with less than $|\mathcal{H}|/|\mathcal{A}|^{T+1}$ samples.

Lemma 8.5.2 (An adaptation of Lemma 5.3). *Assuming* $|\mathcal{H}| > |\mathcal{A}|^{T+3} \log(2|\Pi|/\delta)$, *the probability that there exists a policy* $\pi \in \Pi$ *for which* $|\{h|acc_\pi(h) = 1\}| < |\mathcal{H}|/|\mathcal{A}|^{T+1}$ *is at most* $\delta/2$.

Lemma 5.4 shows that if the set of randomly generated trajectories that are consistent with π, called $S_\pi(\mathcal{H})$, is large enough for all π, then $\tilde{V}^\pi(s_0)$ is a close approximation of $V^\pi(s_0)$. That is,

$$\text{if} \quad |S_\pi(\mathcal{H})| > \left(\frac{V_{max}}{\epsilon}\right)^2 \log(2|\Pi|/\delta) \quad \forall \pi \in \Pi$$

$$\text{then with probability } 1-\delta/2, \quad \left|V^\pi(s_0) - \tilde{V}^\pi(s_0)\right| \leq \epsilon \quad \forall \pi \in \Pi$$

where $\tilde{V}$ is the value obtained from the samples.

Together with Lemma 5.4, the theorem below follows.

Theorem 8.5.3 (An adaptation of Theorem 5.5).
Assuming $|\mathcal{H}| > |\mathcal{A}|^{T+3}(V_{\max}/\epsilon)^2 \log(2|\Pi|/\delta)$, *we have with probability at least* $1-\delta$ *simultaneously for all* $\pi \in \Pi$ *that:*

$$\left|V^\pi(s_0) - \tilde{V}^\pi(s_0)\right| \leq \epsilon$$

The theorem above shows that the value of the generated policy is with probability $1-\delta$ at most ϵ from the true value of the policy. To bound the difference between the generated policy $\tilde{\pi}$ and the optimal policy π^*, we can use the triangle inequality and the optimality of $\tilde{\pi}$ on the samples to prove the following.

Corollary 8.5.4. *From Theorem 8.5.3 we have:*

$$\mathrm{P}\left[V^{\pi^*}(s_0) - V^{\tilde{\pi}(\mathcal{H})} \geq \epsilon\right] \leq \delta.$$

8.6 Experiments

In this section, we evaluate the performance of our goal-directed approach as well as four state-of-the-art infinite-horizon DEC-POMDP approximation algorithms. We compare our approach to the NLP methods from Chapters 5 and 6, Bernstein et al.'s DEC-BPI method [17] and Szer and Charpillet's BFS algorithm [110]. Note that the Moore formulation given in [17] is used for DEC-BPI. Results are given for three common benchmark domains as well as the larger mars rover problem.

8.6.1 Experimental Setup

For the NLP and DEC-BPI approaches, each algorithm was run until convergence was achieved with ten different random deterministic initial controllers, and the mean values and running times were found for a range of controller sizes. The nonlinear program solver snopt on the NEOS server was used to determine solutions for the NLP approaches. The algorithms were run on increasingly larger controllers until memory was exhausted (2GB) or time expired (4 hours). We then report the results for the controller size with the highest value for each method.

For our goal-directed approach, our algorithm was also run until it converged ten times and the mean values and running times are reported. We also provide the sizes of typical controllers that are generated. The total number of trajectories generated and retained were 5000000 and 25 for the two agent tiger problem, 1000000 and 10 for the meeting in a grid and box pushing problems and 500000 and 5 for the two rover problems. It is worth noting that due to the small number of samples retained, our bound on the optimality of the solution does not hold in the results below. Sampling time ranged from less than 30 seconds for the tiger problem to approximately 5 minutes for the stochastic version of the rover problem.

The BFS algorithm was run until on increasing controller sizes until memory or time was exhausted. The solution reported represents the optimal deterministic controller for the largest solvable controller size. Because different machines were required for the algorithms, computation times are not directly comparable. Nevertheless, we expect that they would only vary by a small constant.

We test the algorithms on three benchmark domains and two versions of a very large rover coordination problem. The benchmark domains that were used are the two agent tiger problem [72], the meeting in a grid problem [17] and the box pushing domain [94]. The new problem that we developed is referred to as the mars rover problem. All problems are described in the appendix and a discount factor of 0.9 was used in our experiments.

8.6.2 Results

The experimental results are given in Table 8.1. In all domains except the meeting in a grid problem, higher values are found with our goal-directed approach. The BFS approach is limited to very small controllers so even though an optimal deterministic controller is generated for the given controller sizes, these controllers were not large enough to produce a high value. In the rover problems, it could not find a solution for a one node controller before time was exhausted. DEC-BPI can solve larger controllers, but has a high time requirement and generally performs poorly in these domains. The NLP methods performs better, producing the highest values in the grid problem, while providing the second and third highest value in the others before exhausting the given resources. The Mealy formulation performs especially well with solution quality often approaching that of the goal-directed approach. Nevertheless, the goal-directed approach provides very high value results often with much less time than the other approaches. It requires the least time in all domains except for the rover problems, but in this case, the increase in value is likely worth the small increase in time. In fact, the goal-directed method can sometimes find the known optimal solution in the deterministic rover problem (31.3809). The goal-directed approach

Algorithm	Value	Size	Time
Two Agent Tiger $\|S\| = 2$, $\|A_i\| = 3$, $\|\Omega_i\| = 2$			
Goal-directed	5.04	11,12	75
NLP-Moore[1]	-1.09	19	6173
NLP-Mealy	-1.49	6	226
BFS	-14.12	3	12007
DEC-BPI (Moore)	-52.63	11	102
Meeting in a Grid $\|S\| = 16$, $\|A\| = 5$, $\|\Omega\| = 2$			
NLP-Mealy	6.12	5	52
NLP-Moore	5.66	5	117
Goal-directed	5.64	4	4
BFS	4.21	2	17
DEC-BPI (Moore)	3.60	7	2227
Box Pushing $\|S\| = 100$, $\|A\| = 4$, $\|\Omega\| = 5$			
Goal-directed	149.85	5	199
NLP-Mealy	138.76	4	2828
NLP-Moore[1]	54.23	4	1824
DEC-BPI (Moore)	9.44	3	4094
BFS	-2.00	1	1696
Mars Rover Problems: $\|S\| = 256$, $\|A\| = 6$, $\|\Omega\| = 8$			
Deterministic			
Goal-directed	26.93	5	491
NLP-Mealy	20.89	4	770
NLP-Moore	9.64	2	379
DEC-BPI (Moore)	-1.11	3	11262
BFS	x	x	x
Stochastic			
Goal-directed	21.48	6	956
NLP-Mealy	20.41	3	838
NLP-Moore	8.16	2	43
DEC-BPI (Moore)	-1.18	3	14069
BFS	x	x	x

Table 8.1. The values produced by each method along with the number of nodes in the controllers and the running time in seconds.

[1] These results utilize controllers with deterministic action selection.

is able to produce concise, high-quality solutions because it is can exploit the goal structure of the problem rather to producing a solution for an arbitrary fixed size.

8.7 Discussion

In this chapter, we discussed a natural class of problems in which agents have a set of joint goals. Under assumptions that each agent has a set of terminal actions and rewards for non-terminal actions are negative, we presented an indefinite-horizon model. To solve this model, we described a dynamic programming algorithm that is guaranteed to produce an optimal solution. For problems with a more general reward structure or when other goal conditions are utilized rather than terminal actions, we modeled these problems as goal-directed DEC-POMDPs. We provided an approximate infinite-horizon approach for this class of problems and developed a bound on the number of samples needed to approach optimality. Unlike previous algorithms, our method uses the goal structure to produce finite-state controllers rather than optimizing controllers of an arbitrarily chosen size. Experimental results show that our approach can significantly outperform current state-of-the-art algorithms by quickly producing concise, high quality results even for large DEC-POMDPs.

A major benefit of the approximate goal-directed approach is that sampling of trajectories is used to discover domain structure. This allows the correct controller size to be determined based on the samples and only actions that are present in the samples (i.e. taken given the agent's history) are included. This greatly simplifies the search space and focuses it on solutions that reach the problem's goal. The action selection process is similar to that used in action elimination for Mealy machines, but in the Mealy case, only the previous observation is considered. These methods could be combined so that the trajectories produce a Mealy controller rather than a Moore controller. This would allow a more powerful controller to be constructed in the optimization phase. That is, because actions depend on nodes and observations in a

Mealy representation, a greater range of choices are possible rather than just mapping nodes to actions. This could improve solution quality for the learned controller.

Another topic of interest is extending the trajectory sampling method to non goal-directed problems. This is straightforward to do in finite-horizon problems as we can end trajectories when the horizon is reached. This would result in a forward search method that reduces the search space by providing only feasible solutions and can weight these solutions by value or likelihood. This approach could also be used for infinite-horizon problems. For instance, a researcher could define subgoals that may be useful for agents to achieve. Solving these smaller problems and combining the resulting controllers could allow much larger problems to be solved.

CHAPTER 9
CONCLUSION

Decision-making under uncertainty is an active area of research. As agents are built for ever more complex environments, methods that consider the uncertainty in the system have strong advantages. In this thesis, we explored key questions in planning under uncertainty for both single agent (centralized) and multiagent (decentralized) settings. The approaches that we developed improve the state-of-the-art by increasing scalability while also producing high-quality solutions. We first discuss each of these contributions and then describe some of the remaining open questions that are of future interest.

9.1 Summary of contributions

We began by describing a method called incremental policy generation that improves the performance of the optimal DEC-POMDP dynamic programming algorithms by drastically reducing the necessary search space. This is accomplished by discovering domain structure through reachability analysis. This is the first algorithm to perform incremental backups in DEC-POMDPs and reduces the effect of one of the main bottlenecks for solving large DEC-POMDPs, generating a next step policy. This approach also allows start state information to be incorporated to further reduce the search space and increase scalability. This method allows larger horizons to be used in finite-horizon problems and larger finite- and infinite-horizon problems to be solved optimally.

We also described a method to represent optimal memory-bounded solutions for centralized and decentralized POMDPs as nonlinear programs. While solving this representation optimally may be intractable, we demonstrated that an approximate solution of this NLP provides concise, high quality results. It is worth noting that a generic nonlinear solver was used in this thesis and as specialized solution methods are developed solution quality should increase further. This representation provides a framework to apply a wide range of powerful optimization techniques to solve POMDPs and DEC-POMDPs.

A further reduction in solution size and an increase in solution quality can be attained by utilizing Mealy machines instead of the currently used Moore machines in controller-based methods for POMDPs and DEC-POMDPs. This results in an alternative controller representation that is more powerful and possesses additional structure that can be exploited by solution methods. Mealy machines can be used in any controller-based approach for POMDPs or DEC-POMDPs with minor modifications, often resulting in increased efficiency and solution quality. When compared to the state-of-the-art approximate solution methods for both models, Mealy machines solved by nonlinear programming often outperformed the other approaches in both run time and solution quality.

We also discussed approximate dynamic programming methods for DEC-POMDPs. We developed a sample-based approximate dynamic programming method for infinite-horizon DEC-POMDPs which no longer guarantees optimality, but is much more scalable than the optimal approach. We showed how current finite-horizon approximate DP methods can be improved by the incorporation of incremental policy generation. This allows the algorithms to become more scalable without sacrificing solution quality. We then combined these methods to create a heuristic incremental policy generation algorithm for infinite-horizon problems. These approaches are able to scale to very large problems and horizons, while also producing high quality results.

Lastly, we have incorporated the ideas of sampling, domain structure discovery and memory-based techniques to solve DEC-POMDPs with goals. With this approach, we demonstrated how certain goal conditions can be taken advantage of to produce more efficient solutions. We proved that given certain assumptions, dynamic programming could be adapted to solve the indefinite-horizon problem which terminates when the goal has been reached. We also developed a sample-based algorithm which is able to solve problems with more relaxed goal conditions. We provided a bound on the number of samples required to ensure that the optimal solution is found with a high probability. In general, this algorithm was able to outperform other DEC-POMDP approximate algorithms on a range of goal-directed problems. The approach also provides the framework for sample-based methods to be extended to other classes of decentralized POMDPs.

These approaches provide more scalable approaches for solving POMDPs and DEC-POMDPs. This is accomplished through exploiting domain structure, sampling and the use of memory-bounded representations. These methods demonstrate that we can solve larger problems with larger horizons and provide higher quality solutions. This increased scalability is vital for solving a wide range of applications. Therefore, these contributions supply a valuable set of tools for systems to solve real-world problems modeled as centralized and decentralized POMDPs.

9.2 Future work

There are many engaging open questions in the area of decision-making under uncertainty and a range of appropriate planning and learning applications. We mention some of these issues and applications below.

9.2.1 Theory and Algorithms

Structured representations Structured problem or solution representations (such as factorization or hierarchy) have become very popular and have been able to increase the scalability of solution methods in a wide range of problems. For example, researchers have assumed that the state of the problem is fully observed by each agent, resulting in a form of factored MDP [21] called a multiagent MDP (MMDP). Efficient planning solutions have been developed for these problems [43, 44]. Structure has also been studied in POMDPs, including models with state spaces factored with Bayesian networks [22] and hierarchical controllers [28, 49].

Some of the work presented in this thesis can be applied to structured representations. For instance, [28] extended our nonlinear programming formulation to allow hierarchy to be discovered during the optimization. Also, the subproblems in the hierarchical controllers of [49] can be represented as Moore or Mealy machines and solved with the nonlinear program formulation. Similarly, modifications of the Moore and Mealy formulations would allow factored POMDPs to be represented (including breaking up state variables into factored pieces and similarly breaking up the value constraints based on this factorization).

In the DEC-POMDP case, exploiting factored representations is more difficult. Here, when factored states are considered, but multiple agents may affect these states, the factorization may not hold across multiple steps of the problem. That is, while state variables for different agents may be independent for one step, dependences may exist after several steps allowing for only minor efficiency gains [76]. A more limited version of the DEC-MDP model is required to achieve a more fully factored representation [116]. It is a topic of future work to extend the DEC-POMDP algorithms in this thesis to similarly factored representations.

Another approach to extend factored models to DEC-POMDPs could incorporate Algebraic Decision Diagrams (ADDs). ADDs have been used in factored POMDPs to more compactly represent problems and allow them to be solved more efficiently [48, 99]. As it becomes difficult to even represent DEC-POMDPs with a large number of states, actions and observations, the model and value functions may similarly benefit from ADD representations.

Hierarchical models are also difficult to represent in DEC-POMDPs. This is because decentralized agents that are completing subtasks of different lengths would no longer be synchronized. That is, because agents cannot observe when other agents have completed their subtasks, they may choose to move to a different part of the hierarchy while other agents are still completing previous tasks. Thus, the agents' decisions are no longer synchronized, which may result in poor performance. Therefore, it remains an open problem as to how to represent hierarchical DEC-POMDP models.

Structure discovery To achieve better scalability and efficiency in planning and learning algorithms for both single and multiagent domains, structured policy representations could also be discovered. Goal-based methods can determine controller structure by sampling and Mealy machines provide structure in the form the previous observation, but many real-world problems have additional structure that could benefit solution methods. In order to make better use of limited memory, we could identify key attributes of an agent's action and observation history and use them to model potential solutions. These attributes may consist of such information as whether another agent has been seen or an estimate of the system state. An agent can then remember these attributes rather than the observation histories themselves. This approach could improve scalability by reducing memory requirements. Techniques such as search, constraint satisfaction and optimization as well as machine learning methods like feature extraction can be used to determine this structure, which can then be exploited by the solution methods. If structure is already known (as in the case of factored models or those that consider independence and locality of interaction in DEC-POMDPs), the knowledge could be directly incorporated into solution representations. These models could be solved by methods similar to those described in this thesis. In many problems, this combination of approaches is likely to improve both scalability and solution quality.

Limiting assumptions Current models for sequential decision-making under uncertainty can represent many problems, but several common simplifying assumptions limit their applicability. Some of these limitations arise from assuming discrete parameters and static environments. Many real-world scenarios such as robots with continuous actuators and sensors or software agents on the internet where the environment frequently changes and other competitive agents may be present do not fit these assumptions. There is currently limited work on these topics, leading to questions of how to better: extend models and algorithms to continuous state, action and observation spaces, develop online planning and learning algorithms to solve problems with dynamic environments and develop multiagent models to include possibly adversarial agents in addition to a team. Also, many of the cooperative techniques developed in this thesis could be applied to competitive settings in an effort to improve algorithms for determining equilibria.

9.2.2 Applications

There are many real-world applications for decision-making under uncertainty. These applications can inform research by identifying major barriers to solving realistic scenarios, while making the work more grounded and focused on improving scalability in a practical way. Recent POMDP applications have included assistive systems for the elderly [53], server maintenance [98] and robot navigation [59].

Other centralized and decentralized applications include: medical diagnosis and treatment, green computing, sensor networks and e-commerce. In *medical diagnosis and treatment*, there is inherent uncertainty due to partial patient information. Models of uncertainty could be used to decide which tests and treatments should be used to optimize the health of the patient and reduce inefficiencies. This type of problem has already been modeled as a POMDP [51]. In *green computing*, decision-theoretic methods such as POMDPs or DEC-POMDPs could be used to save energy by deciding which computers to run jobs on or when to power machines on or off based on partial and local system information. This is similar to the server maintenance problem above. DEC-POMDPs are a natural model for *sensor network* applications where agents have limited system knowledge such as weather monitoring, target tracking, etc. Multiagent models can also be used in *e-commerce* systems where a team of agents makes purchases in stock markets or internet vendors. These models seek to maximize the profit for the user in fast-paced uncertain environments without the use of global information. In the future, applications such as these that require complex models will become even more common as computing power increases and computers become more ubiquitous. More sophisticated models for decision-making will be required and more sophisticated solution methods will need to be developed.

APPENDIX A
DEC-POMDP DOMAIN DESCRIPTIONS

A.1 Recycling robot problem

We have extended the recycling robot problem [109] to the multiagent case. The robots have the task of picking up cans in an office building. They have sensors to find a can and motors to move around the office in order to look for cans. The robots are able to control a gripper arm to grasp each can and then place it in an on-board receptacle. Each robot has three high level actions: (1) search for a small can, (2) search for a large can or (3) recharge the battery. In our two agent version, the larger can is only retrievable if both robots pick it up at the same time. Each agent can decide to independently search for a small can or to attempt to cooperate in order to receive a larger reward. If only one agent chooses to retrieve the large can, no reward is given. For each agent that picks up a small can, a reward of 2 is given and if both agents cooperate to pick the large can, a reward of 5 is given. The robots have the same battery states of high and low, with an increased likelihood of transitioning to a low state or exhausting the battery after attempting to pick up the large can. Each robot's battery power depends only on its own actions and each agent can fully observe its own level, but not that of the other agent. If the robot exhausts the battery, it is picked up and plugged into the charger then continues to act on the next step with a high battery level. The two robot version used in this thesis has 4 states, 3 actions and 2 observations.

A.2 Two agent tiger problem

Another domain with 2 states, 3 actions and 2 observations called the multiagent tiger problem was introduced by Nair et al. [72]. In this problem, there are two doors. Behind one door is a tiger and behind the other is a large treasure. Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action (i.e., both opening the same door) a larger positive reward or a smaller penalty is given to reward this cooperation. If an agent listens, a small penalty is given and an observation is seen that is a noisy indication of which door the tiger is behind. While listening does not change the location of the tiger, opening a door causes the tiger to be placed behind one of the door with equal probability.

A.3 Meeting in a grid problems

A larger domain with 16 states, 5 actions and 2 observations was also introduced by Bernstein et al. In this problem, two agents must meet in a 2x2 grid with no obstacles. The agents start diagonally across from each other and available actions are move left, right, up, or down and stay in place. Only walls to the left and right can be observed, resulting in each agent knowing only if it is on the left or right half. The agents cannot observe each other and do not interfere with other. Action transitions are noisy with the agent possibly moving in another direction or staying in place and a reward of 1 is given for each step the agents share the same square.

We extended this problem to a version which has a grid that is 3x3 rather than 2x2. The agents can observe walls in four directions rather than two and they must meet in the top left or bottom right corner rather than in any square. The resulting problem has 81 states, 5 actions and 9 observations.

A.4 Box pushing problem

A much larger domain was introduced by Seuken and Zilberstein [94]. This problem, with 100 states, 4 actions and 5 observations consists of two agents that can gain reward by pushing different boxes. The agents begin facing each other in the bottom corners of a 4x3 grid with the available actions of turning right, turning left, moving forward or staying in place. There is a 0.9 probability that the agent will succeed in moving and otherwise will stay in place, but the two agents can never occupy the same square. The middle row of the grid contains two small boxes and one large box. This large box can only be moved by both agents pushing at the same time. The upper row of the grid is considered the goal row. The possible deterministic observations consist of seeing an empty space, a wall, the other agent, a small box or the large box. A reward of 100 is given if both agents push the large box to the goal row and 10 is given for each small box that is moved to the goal row. A penalty of -5 is given for each agent that cannot move and -0.1 is given for each time step. Once a box is moved to the goal row, the environment resets to the original start state.

A.5 Rover problems

This problem has more than twice the number of states of the previous largest domain. It has 256 states, 6 actions and 8 observations and involves two rovers performing certain scientific experiments at a set of sites. The agents can conduct scientific experiments at four possible sites by choosing to drill or sample. These sites are arranged in a two-by-two grid in which the rovers can independently move north, south, east and west. Two of the sites require only one agent to sample them while two of the sites require both agents to drill at the same time in order to get the maximum reward. If a site only needs to be sampled, but it is drilled instead, the site is considered ruined and thus a large negative reward is given. If a sites requires drilling, but is sampled instead, a small positive reward is given. Once a site is sampled or drilled any further experiment is considered redundant and incurs a small negative reward. The rovers can fully observe their own local location as well as whether an experiment has already been performed at that site. When at least one experiment is performed at each site, the problem is reset. We provide results for a version of this problems in which the rovers can deterministically move in the intended direction and a version in which there is a small probability that movement fails and the rover stays in place.

APPENDIX B

AMPL FORMULATIONS FOR NLP

In this thesis, we used the nonlinear program solver snopt [38] on the NEOS server (www-neos.mcs.anl.gov) to solve POMDPs and DEC-POMDPs. In this appendix, we provide the AMPL formulation [36] for the Moore NLP representation as well as a description of the problem data files. These files as well as the Mealy formulations and several other files can be found on the DEC-POMDP webpage[1].

B.1 The POMDP formulation

The AMPL formulation for solving POMDPs is provided in this section. We first describe our model for the NLP and then give an example of a problem data file for this model.

B.1.1 NLP model

In the model below, `param` represents a fixed value while `var` represents a variable. Parameters defined with `param` are set in the data file described in the next section. Variables defined with `var` can also be initialized in a similar manner. Arrays of parameters or variables are defined with curly braces ({}). For instance,

```
param reward {s in 1..numSTATES, a in 1..numACTIONS};
```

defines a two-dimensional array of size `numSTATES*numACTIONS` and is accessed by `reward[s,a]` as seen in the nonlinear bellman constraints below.

The objective function and constraints are defined in a similar manner. The objective is set with `maximize` followed by a name (here termed `Belief`) and a colon.

[1]Currently at http://www.cs.umass.edu/∼camato/decpomdp

The keyword `sum` is used to add the values. The constraints are defined starting with keywords `subject to` followed by a name and then the constraint. A set of constraints can be formed by adding the curly braces before the colon.

```
# the general POMDP problem

param numNODES;
param numSTATES;
param numACTIONS;
param numOBSERVATIONS;
param initial_belief {s in 1..numSTATES};
param reward {s in 1..numSTATES, a in 1..numACTIONS};
param trans {s in 1..numSTATES, a in 1..numACTIONS, s_ in 1..numSTATES};
param obs {o in 1..numOBSERVATIONS, s in 1..numSTATES, a in 1..numACTIONS};
param discount;

var Value {q in 1..numNODES, s in 1..numSTATES};
var Prob {q_ in 1..numNODES, a in 1..numACTIONS, q in 1..numNODES, o in 1..numOBSERVATIONS} >= 0;

# objective
maximize Belief: sum {s in 1..numSTATES} initial_belief[s] * Value[1,s];

# Nonlinear Bellman constraints
subject to Value_Equality {q in 1..numNODES, s in 1..numSTATES}:
sum {a in 1..numACTIONS} ((sum {q_ in 1..numNODES} Prob[q_, a, q, 1]) * reward[s, a] +
discount * sum {s_ in 1..numSTATES} trans[s,a,s_] * sum {o in 1..numOBSERVATIONS} obs[o, s_, a] *
sum {q_ in 1..numNODES} Prob[q_, a, q, o] * Value[q_, s_]) = Value[q, s];

# probs sum to 1
subject to Sum_to_One {q in 1..numNODES, o in 1..numOBSERVATIONS}:
sum {q_ in 1..numNODES, a in 1..numACTIONS} Prob[q_, a, q, o] = 1;

# sum out action probs
subject to Action_Probs {a in 1..numACTIONS, q in 1..numNODES, o in 1..numOBSERVATIONS}:
sum {q_ in 1..numNODES} Prob[q_, a, q, o] = sum {q_ in 1..numNODES} Prob[q_, a, q, 1];
```

B.1.2 Problem data

Here, we provide an example of the data file for the Hallway problem [64] that can be found on Tony's POMDP page[2] as `hallway.POMDP`. Ellipses (...) are used to represent data omitted for the sake of brevity. The full file is available on the DEC-POMDP webpage.

```
param numNODES:=5;
param numSTATES:=60;
param numACTIONS:=5;
param numOBSERVATIONS:=21;
param discount:=0.95;
param initial_belief:= 1 0.017865
2 0.017857
```

[2]http://www.cs.brown.edu/research/ai/pomdp/examples/index.html

```
3 0.017857
4 0.017857
5 0.017857
6 0.017857
7 0.017857
8 0.017857
9 0.017857
10 0.017857
11 0.017857
12 0.017857
13 0.017857
14 0.017857
15 0.017857
16 0.017857
17 0.017857
18 0.017857
19 0.017857
20 0.017857
21 0.017857
22 0.017857
23 0.017857
24 0.017857
25 0.017857
26 0.017857
27 0.017857
28 0.017857
29 0.017857
30 0.017857
31 0.017857
32 0.017857
33 0.017857
34 0.017857
35 0.017857
36 0.017857
37 0.017857
38 0.017857
39 0.017857
40 0.017857
41 0.017857
42 0.017857
43 0.017857
44 0.017857
45 0.017857
46 0.017857
47 0.017857
48 0.017857
49 0.017857
50 0.017857
51 0.017857
52 0.017857
53 0.017857
54 0.017857
55 0.017857
56 0.017857
57 0.0
58 0.0
59 0.0
60 0.0
;
param trans default 0.0
[1, 1, 1] 1.0
[1, 2, 1] 0.95
[1, 2, 6] 0.05
...
;
param obs default 0.0
[1, 1, 1] 9.49E-4
[1, 1, 2] 9.49E-4
```

```
[1, 1, 3] 9.49E-4
[1, 1, 4] 9.49E-4
[1, 1, 5] 9.49E-4
[1, 2, 1] 9.49E-4
[1, 2, 2] 9.49E-4
[1, 2, 3] 9.49E-4
[1, 2, 4] 9.49E-4
[1, 2, 5] 9.49E-4
...
;
param reward [1, 1] 0.0
[1, 2] 0.0

...
;
```

B.2 The DEC-POMDP formulation

The AMPL formulation for solving DEC-POMDPs is provided in this section. We begin with the model for the NLP and then give an example of a problem data file that can be used with this model.

B.2.1 NLP model

The DEC-POMDP model is constructed in a manner very similar to the POMDP model. The parameters, variables, objective function and constraints are given below.

```
# the general DEC-POMDP problem with separate trans and action parameters for each agent

param numNODES;  #number of nodes for each agent
param numSTATES;
param numACTIONS;
param numOBSERVATIONS;
param initial_belief {s in 1..numSTATES};
param reward {s in 1..numSTATES, a1 in 1..numACTIONS, a2 in 1..numACTIONS};
param trans {s in 1..numSTATES, a1 in 1..numACTIONS, a2 in 1..numACTIONS, s_ in 1..numSTATES};
param obs {o1 in 1..numOBSERVATIONS, o2 in 1..numOBSERVATIONS, s_ in 1..numSTATES, a1 in 1..numACTIONS,
a2 in 1..numACTIONS};
param discount;

var Value {q1 in 1..numNODES, q2 in 1..numNODES, s in 1..numSTATES};
# A(q1,a1) and A(q2,a2)
var ProbAct1 {q1 in 1..numNODES, a1 in 1..numACTIONS} >= 0;
var ProbAct2 {q2 in 1..numNODES, a2 in 1..numACTIONS} >= 0;
# T(q1, q2, a1, a2, o1, o2, q1', q2')
var ProbTrans1 {q1 in 1..numNODES, a1 in 1..numACTIONS, o1 in 1..numOBSERVATIONS, q_1 in 1..numNODES} >= 0;
var ProbTrans2 {q2 in 1..numNODES, a2 in 1..numACTIONS, o2 in 1..numOBSERVATIONS, q_2 in 1..numNODES} >= 0;

maximize Belief: sum {s in 1..numSTATES} initial_belief[s] * Value[1,1,s];

# Nonlinear Bellman constraints
subject to Value_Equality {q1 in 1..numNODES, q2 in 1..numNODES, s in 1..numSTATES}:
sum {a1 in 1..numACTIONS, a2 in 1..numACTIONS} ProbAct1[q1, a1] * ProbAct2[q2, a2] * (reward[s, a1, a2] +
discount * sum {s_ in 1..numSTATES} trans[s,a1,a2,s_] *
sum {o1 in 1..numOBSERVATIONS, o2 in 1..numOBSERVATIONS} obs[o1,o2,s_,a1,a2] *
sum {q_1 in 1..numNODES, q_2 in 1..numNODES} ProbTrans1[q1, a1, o1, q_1] * ProbTrans2[q2, a2, o2, q_2] *
Value[q_1, q_2, s_]) = Value[q1, q2, s];


# trans probs sum to 1
subject to Sum_to_One_Trans1 {q1 in 1..numNODES, a1 in 1..numACTIONS, o1 in 1..numOBSERVATIONS}:
sum {q_1 in 1..numNODES} ProbTrans1[q1, a1, o1, q_1] = 1;

subject to Sum_to_One_Trans2 {q2 in 1..numNODES, a2 in 1..numACTIONS, o2 in 1..numOBSERVATIONS}:
sum {q_2 in 1..numNODES} ProbTrans2[q2, a2, o2, q_2] = 1;


# act probs sum to 1
subject to Sum_to_One_Act1 {q1 in 1..numNODES}:
sum {a1 in 1..numACTIONS} ProbAct1[q1, a1] = 1;

subject to Sum_to_One_Act2 {q2 in 1..numNODES}:
sum {a2 in 1..numACTIONS} ProbAct2[q2, a2] = 1;
```

B.2.2 Problem data

The data file for the box pushing problem [94] (also described in Appendix A) is given below. Again, ellipses (. . .) are used to represent data omitted for the sake of brevity and the full file is available on the DEC-POMDP webpage.

```
param numNODES:=4;
param numSTATES:=100;
param numACTIONS:=4;
param numOBSERVATIONS:=5;
param discount:=0.9;
param initial_belief default 0.0
28 1.0
;
param trans default 0.0
[1, 1, 1, 5] 0.02
[1, 1, 1, 6] 0.02
[1, 1, 1, 7] 0.02
[1, 1, 1, 8] 0.02
[1, 1, 1, 9] 0.02
...
;
param obs default 0.0
[1, 1, 1, 1, 1, 1] 1.0
[1, 1, 1, 1, 1, 2] 1.0
[1, 1, 1, 1, 1, 3] 1.0
[1, 1, 1, 1, 1, 4] 1.0
...
;
param reward default 0.0
[5, 1, 1] -0.2
[5, 1, 2] -0.2
[5, 1, 3] -5.199999999999999
[5, 1, 4] -0.2
[5, 2, 1] -0.2
[5, 2, 2] -0.2
[5, 2, 3] -5.199999999999999
[5, 2, 4] -0.2
[5, 3, 1] 9.8
[5, 3, 2] 9.8
...
;
```

BIBLIOGRAPHY

[1] Allen, Martin, Petrik, Marek, and Zilberstein, Shlomo. Interaction structure and dimensionality reduction in decentralized MDPs. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence* (Chicago, Illinois, 2008), pp. 1440–1441.

[2] Allen, Martin, and Zilberstein, Shlomo. Complexity of decentralized control: Special cases. In *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 22. 2009, pp. 19–27.

[3] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence* (Vancouver, Canada, 2007).

[4] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (Hyderabad, India, 2007), pp. 2418–2424.

[5] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Journal of Autonomous Agents and Multi-Agent Systems DOI: 10.1007/s10458-009-9103-z* (2009).

[6] Amato, Christopher, Bonet, Blai, and Zilberstein, Shlomo. Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs. In *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence* (Atlanta, GA, 2010).

[7] Amato, Christopher, Dibangoye, Jilles Steeve, and Zilberstein, Shlomo. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* (Thessaloniki, Greece, 2009), pp. 2–9.

[8] Amato, Christopher, and Zilberstein, Shlomo. Achieving goals in decentralized POMDPs. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems* (Budapest, Hungary, 2009), pp. 593–600.

[9] Aras, Raghav, Dutech, Alain, and Charpillet, François. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling* (Providence, RI, 2007), pp. 18–25.

[10] Åström, Karl J. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications 10* (1965), 174–205.

[11] Baxter, Jonathan, and Bartlett, Peter L. Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning* (2000), pp. 41–48.

[12] Becker, Raphen, Carlin, Alan, Lesser, Victor, and Zilberstein, Shlomo. Analyzing Myopic Approaches for Multi-Agent Communication. *Computational Intelligence 25*, 1 (2009), 31–50.

[13] Becker, Raphen, Lesser, Victor, and Zilberstein, Shlomo. Decentralized Markov Decision Processes with Event-Driven Interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (New York, NY, 2004), pp. 302–309.

[14] Becker, Raphen, Zilberstein, Shlomo, Lesser, Victor, and Goldman, Claudia V. Solving transition-independent decentralized Markov decision processes. *Journal of AI Research 22* (2004), 423–455.

[15] Bernstein, Daniel S., Amato, Christopher, Hansen, Eric A., and Zilberstein, Shlomo. Policy iteration for decentralized control of Markov decision processes. *Journal of AI Research 34* (2009), 89–132.

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

[17] Bernstein, Daniel S., Hansen, Eric A., and Zilberstein, Shlomo. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* (Edinburgh, Scotland, 2005), pp. 1287–1292.

[18] Bertsekas, Dimitri P. *Nonlinear Programming.* Athena Scientific, 2004.

[19] Bonet, Blai, and Geffner, Hèctor. Solving POMDPs: RTDP-Bel bs. point-based algorithms. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* (Pasadena, California, 2009), pp. 1641–1646.

[20] Boularias, Abdeslam, and Chaib-draa, Brahim. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling* (Sydney, Australia, 2008).

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

[22] Boutilier, Craig, and Poole, David. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (Portland, OR, 1996), pp. 1168–1175.

[23] Carlin, Alan, and Zilberstein, Shlomo. Value-based observation compression for DEC-POMDPs. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (Estoril, Portugal, 2008).

[24] Cassandra, Anthony, Littman, Michael L., and Zhang, Nevin L. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, 1997).

[25] Cassandra, Anthony R. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Providence, RI, 1998.

[26] Cassandra, Anthony R. A survey of POMDP applications. In *AAAI Fall Symposium: Planning with POMDPs* (Orlando, FL, 1998).

[27] Chalkiadakis, Georgios, and Boutilier, Craig. Coordination in multiagent reinforcement learning: A Bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems* (Melbourne, 2003), pp. 709–716.

[28] Charlin, Laurent, Poupart, Pascal, and Shioda, Romy. Automated hierarchy discovery for planning in partially observable environments. In *Advances in Neural Information Processing Systems*, 19. 2007, pp. 225–232.

[29] Claus, Caroline, and Boutilier, Craig. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (1998).

[30] Cogill, Randy, Rotkowitz, Michael, Van Roy, Benjamin, and Lall, Sanjay. An approximate dynamic programming approach to decentralized control of stochastic systems. In *Proceedings of the Forty-Second Allerton Conference on Communication, Control, and Computing* (2004).

[31] de Waal, Peter R., and van Schuppen, Jan H. A class of team problems with discrete action spaces: Optimality conditions based on multimodularity. *SIAM Journal on Control and Optimization 38* (2000), 875–892.

[32] Dibangoye, Jilles Steeve, Mouaddib, Abdel-Illah, and Chaib-draa, Brahim. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems* (Budapest, Hungary, 2009).

[33] Dutech, Alain, Buffet, Olivier, and Charpillet, François. Multi-agent systems by incremental gradient reinforcement learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (2001), pp. 833–838.

[34] Eckles, James E. Optimum maintenance with incomplete information. *Operations Research 16* (1968), 1058–1067.

[35] Emery-Montemerlo, Rosemary, Gordon, Geoff, Schneider, Jeff, and Thrun, Sebastian. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (New York, NY, 2004), pp. 136–143.

[36] Fourer, Robert, Gay, David M., and Kernighan, Brian W. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, 2002.

[37] Geffner, Hector. Classical, probabilistic and contingent planning: Three models, one algorithm. In *Proceedings AIPS'98 Workshop on Planning as Combinatorial Search* (1998).

[38] Gill, Philip E., Murray, Walter, and Saunders, Michael. Snopt: An SQP algorithm for large-scale constrained optimization. *SIAM Review 47* (2005), 99–131.

[39] Gmytrasiewicz, Piotr J., and Doshi, Prashant. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research 24* (2005), 24–49.

[40] Goldman, Claudia V., Allen, Martin, and Zilberstein, Shlomo. Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems 15*, 1 (2007), 47–90.

[41] Goldman, Claudia V., and Zilberstein, Shlomo. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems* (Melbourne, Australia, 2003).

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

[43] Guestrin, Carlos, Koller, Daphne, and Parr, Ronald. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, 15. 2001, pp. 1523–1530.

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

[45] Hansen, Eric A. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (Madison, WI, 1998), pp. 211–219.

[46] Hansen, Eric A. Indefinite-horizon POMDPs with action-based termination. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence* (Vancouver, Canada, 2007), pp. 291–296.

[47] Hansen, Eric A., Bernstein, Daniel S., and Zilberstein, Shlomo. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (San Jose, CA, 2004), pp. 709–715.

[48] Hansen, Eric A., and Feng, Zhengzhu. Dynamic programming for POMDPs using a factored state represenation. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems* (Breckenridge, CO, 2000).

[49] Hansen, Eric A., and Zhou, Rong. Synthesis of hierarchical finite-state controllers for POMDPs. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling* (Trento, Italy, 2003), pp. 113–122.

[50] Hauskrecht, Milos. Value-function approximations for partially observable Markov decision processes. *Journal of AI Research 13* (2000), 33–94.

[51] Hauskrecht, Milos, and Fraser, Hamish. Modeling treatment of ischemic heart disease with partially observable Markov decision processes. In *Proceedings of American Medical Informatics Association annual symposium on Computer Applications in Health Care* (Orlando, Florida, 1998), pp. 538–542.

[52] Ho, Yu-Chi. Team decision theory and information structures. *Proceedings of the IEEE 68*, 6 (1980), 644–654.

[53] Hoey, Jesse, von Bertoldi, Axel, Poupart, Pascal, and Mihailidis, Alex. Assisting persons with dementia during handwashing using a partially observable Markov decision process. In *Proceedings of the International Conference on Vision Systems* (Biefeld, Germany, 2007).

[54] Hopcroft, John E., and Ullman, Jeffrey D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[55] Horst, Reiner, and Tuy, Hoang. *Global Optimization: Deterministic Approaches.* Springer, 1996.

[56] Ji, Shihao, Parr, Ronald, Li, Hui, Liao, Xuejun, and Carin, Lawrence. Point-based policy iteration. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence* (Vancouver, Canada, 2007), pp. 1243–1249.

[57] Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence 101* (1998), 1–45.

[58] Kapetanakis, Spiros, and Kudenko, Daniel. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (2002), pp. 326–331.

[59] Kaplow, Robert, Atrash, Amin, and Pineau, Joelle. Variable resolution decomposition for robotic navigation under a POMDP framework. In *Proceedings of the International Conference on Robotics and Automation* (2010).

[60] Kearns, Michael, Mansour, Yishay, and Ng, Andrew Y. Approximate planning in large POMDPs via reusable trajectories. http://robotics.stanford.edu/ ang/papers/pomdp-long.pdf, 1999.

[61] Kumar, Akshat, and Zilberstein, Shlomo. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-agent Systems* (Budapest, Hungary, 2009), pp. 561–568.

[62] Li, Hui, Liao, Xuejun, and Carin, Lawrence. Incremental least squares policy iteration for POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (Boston, MA, 2006), pp. 1167–1172.

[63] Littman, Michael, Cassandra, Anthony R., and Kaelbling, Leslie Pack. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning* (1995), pp. 362–370.

[64] Littman, Michael L., Cassandra, Anthony R., and Kaelbling, Leslie Pack. Learning policies for partially observable environments: Scaling up. Tech. Rep. CS-95-11, Brown University, Department of Computer Science, Providence, RI, 1995.

[65] Littman, Michael L., Sutton, Richard S., and Singh, Satinder. Predictive representations of state. In *Advances in Neural Information Processing Systems*, 14. 2001, pp. 1555–1561.

[66] Lovejoy, William S. Computationally feasible bounds for partially observed Markov decision processes. *Operations research 39*, 1 (1991), 162–175.

[67] Madani, Omid, Hanks, Steve, and Condon, Anne. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence 147* (2003), 5–34.

[68] Marecki, Janusz, Gupta, Tapana, Varakantham, Pradeep, Tambe, Milind, and Yokoo, Makoto. Not all agents are equal: Scaling up distributed POMDPs for agent networks. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (Estoril, Portugal, 2008).

[69] Marschak, Jacob. Elements for a theory of teams. *Management Science 1* (1955), 127–137.

[70] Meuleau, Nicolas, Kim, Kee-Eung, Kaelbling, Leslie Pack, and Cassandra, Anthony R. Solving POMDPs by searching the space of finite policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (Stockholm, Sweden, 1999), pp. 417–426.

[71] Meuleau, Nicolas, Peshkin, Leonid, Kim, Kee-eung, and Kaelbling, Leslie Pack. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (1999), pp. 427–436.

[72] Nair, Ranjit, Pynadath, David, Yokoo, Makoto, Tambe, Milind, and Marsella, Stacy. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* (Acapulco, Mexico, 2003), pp. 705–711.

[73] Nair, Ranjit, Varakantham, Pradeep, Tambe, Milind, and Yokoo, Makoto. Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs. In *AAAI'05: Proceedings of the 20th national conference on Artificial intelligence* (2005).

[74] Oliehoek, Frans A., Spaan, Matthijs, Dibangoye, Jilles, and Amato, Christopher. Solving identical payoff bayesian games with heuristic search. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems* (Toronto, Canada, 2010).

[75] Oliehoek, Frans A., Spaan, Matthijs T J, and Vlassis, Nikos. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of AI Research 32* (2008), 289–353.

[76] Oliehoek, Frans A., Spaan, Matthijs T J, Whiteson, Shimon, and Vlassis, Nikos. Exploiting locality of interaction in factored Dec-POMDPs. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (Estoril, Portugal, 2008).

[77] Oliehoek, Frans A., Whiteson, Shimon, and Spaan, Matthijs T J. Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems* (Budapest, Hungary, 2009).

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

[79] Patek, Stephen D. On partially observed stochastic shortest path problems. In *Proceedings of the Fortieth IEEE Conference on Decision and Control* (Orlando, FL, 2001), pp. 5050–5055.

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

[81] Petrik, Marek, and Zilberstein, Shlomo. Average-reward decentralized Markov decision processes. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (Hyderabad, India, 2007), pp. 1997–2002.

[82] Petrik, Marek, and Zilberstein, Shlomo. A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research 35* (2009), 235–274.

[83] Pineau, Jolle, Gordon, Geoffrey, and Thrun, Sebastian. Point-based value iteration: an anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (Acapulco, Mexico, 2003), pp. 1025–1032.

[84] Platzman, Loren K. A feasible computational approach to infinite-horizon partially-observed Markov decision processes. Tech. rep., Georgia Institute of Technology, 1980. Reprinted in *Working Notes of the 1998 AAAI Fall Symposium on Planning Using Partially Observable Markov Decision Processes.*

[85] Poupart, Pascal. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes.* PhD thesis, University of Toronto, 2005.

[86] Poupart, Pascal, and Boutilier, Craig. Bounded finite state controllers. In *Advances in Neural Information Processing Systems*, 16. Vancouver, Canada, 2003, pp. 823–830.

[87] Poupart, Pascal, and Vlassis, Nikos. Model-based Bayesian reinforcement learning in partially observable domains. In *Proceedings of the Eleventh International Symposium on Artificial Intelligence and Mathematics* (Fort Lauderdale, FL, 2008).

[88] Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 1994.

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

[90] Radner, Roy. Team decision problems. *Annals of Mathematical Statistics 33* (1962), 857–881.

[91] Ross, Stéphane, Pineau, Joelle, Paquet, Sébastien, and Chaib-draa, Brahim. Online planning algorithms for pomdps. *Journal of AI Research 32*, 1 (2008), 663–704.

[92] Roth, Maayan, Simmons, Reid, and Veloso, Maria Manuela. Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems* (Utrecht University, 2005).

[93] Roy, Nicholas. Finding approximate POMDP solutions through belief compression. *Journal of AI Research 23* (2000), 2005.

[94] Seuken, Sven, and Zilberstein, Shlomo. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence* (Vancouver, Canada, 2007), pp. 344–351.

[95] Seuken, Sven, and Zilberstein, Shlomo. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (Hyderabad, India, 2007), pp. 2009–2015.

[96] Seuken, Sven, and Zilberstein, Shlomo. Formal models and algorithms for decentralized control of multiple agents. *Journal of Autonomous Agents and Multi-Agent Systems 17*, 2 (2008), 190–250.

[97] Shani, Guy, Brafman, Ronen I., and Shimony, Solomon E. Model-based online learning of POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning* (2005).

[98] Shani, Guy, and Meek, Christopher. Improving existing fault recovery policies. In *Advances in Neural Information Processing Systems*, 22. 2009.

[99] Sim, Hyeong Seop, Kim, Kee-Eung, Kim, Jim Hyung, Chang, Du-Seong, and Koo, Myoung-Wan. Symbolic heuristic search value iteration for factored POMDPs. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence* (Chicago, IL, 2008).

[100] Simmons, Reid, and Koenig, Sven. Probabilistic navigation in partially observable environments. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (Montral, Canada, 1995), pp. 1080–1087.

[101] Singh, Satinder, Jaakkola, Tommi, and Jordan, Michael. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning* (New Brunswick, NJ, 1994), pp. 284–292.

[102] Singh, Satinder, Littman, Michael L., Jong, Nicholas K., Pardoe, David, and Stone, Peter. Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning* (2003), pp. 712–719.

[103] Smallwood, Richard D., and Sondik, Edward J. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research 21* (1973), 1071–1088.

[104] Smith, Trey, and Simmons, Reid. Heuristic search value iteration for POMDPs. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence* (Banff, Canada, 2004).

[105] Smith, Trey, and Simmons, Reid. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence* (Edinburgh, Scotland, 2005), pp. 542–549.

[106] Sondik, Edward J. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971.

[107] Sondik, Edward J. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research 26* (1978), 282–304.

[108] Spaan, Matthijs T. J., and Vlassis, Nikos. Perseus: Randomized point-based value iteration for POMDPs. *Journal of AI Research 24* (2005), 195–220.

[109] Sutton, Richard S., and Barto, Andrew G. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[110] Szer, Daniel, and Charpillet, François. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning* (Porto, Portugal, 2005), pp. 389–399.

[111] Szer, Daniel, and Charpillet, François. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (Boston, MA, 2006).

[112] Szer, Daniel, Charpillet, François, and Zilberstein, Shlomo. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence* (Edinburgh, Scotland, 2005).

[113] Wah, Benjamin W., and Chen, Yixin. Solving large-scale nonlinear programming problems by constraint partitioning. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming* (2005).

[114] Wang, Xiao Feng, and Sandholm, Tuomas. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Advances in Neural Information Processing Systems*, 10. 2002, pp. 1010–1016.

[115] Weiss, Gerhard. *Multiagent Systems*. The MIT Press, Cambridge, MA, 1999.

[116] Witwicki, Stefan J., and Durfee, Edmund H. Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling* (Toronto, Canada, 2010).

[117] Wolfe, Britton, James, Michael R., and Singh, Satinder. Learning predictive state representations in dynamical systems without reset. In *Proceedings of the Twenty-Second International Conference on Machine learning* (New York, NY, USA, 2005), pp. 980–987.

[118] Wu, Feng, Zilberstein, Shlomo, and Chen, Xiaoping. Multi-agent online planning with communication. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* (Thessaloniki, Greece, 2009).

[119] Xuan, Ping, and Lesser, Victor. Multi-agent policies: From centralized ones to decentralized ones. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems* (2002).

[120] Xuan, Ping, Lesser, Victor, and Zilberstein, Shlomo. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents* (2001).