

# Complexity Results for Infinite-Horizon Markov Decision Processes

Omid Madani

A dissertation submitted in partial fulfillment  
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2000

Program Authorized to Offer Degree: Computer Science & Engineering



University of Washington  
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Omid Madani

and have found that it is complete and satisfactory in all respects,  
and that any and all revisions required by the final  
examining committee have been made.

Chair of Supervisory Committee:

---

Richard Anderson

Reading Committee:

---

Steve Hanks

---

Richard Anderson

---

Anne Condon

Date: \_\_\_\_\_



In presenting this dissertation in partial fulfillment of the requirements for the Doctorial degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature\_\_\_\_\_

Date\_\_\_\_\_



University of Washington

Abstract

## Complexity Results for Infinite-Horizon Markov Decision Processes

by Omid Madani

Chair of Supervisory Committee

Professor Richard Anderson  
Computer Science and Engineering

Markov decision processes (MDPs) are models of dynamic decision making under uncertainty. These models arise in diverse applications and have been developed extensively in fields such as operations research, control engineering, and the decision sciences in general. Recent research, especially in artificial intelligence, has highlighted the significance of studying the computational properties of MDP problems.

We address computational complexity questions in solving MDP problems under infinite-horizon optimization criteria. In an infinite-horizon or an indefinite-horizon criterion, we are interested in making good decisions for the long run or an indefinite period of time. A distinction that partitions the thesis is the assumption on the observability of the state of the system with which the decision maker interacts.

In the first half of the thesis, we focus on partially observable or POMDP problems, wherein there is uncertainty about the current state of the system at times of decision. We show that many POMDP problems are undecidable. We make use of an undecidability result in research on probabilistic finite automata to establish our results.

In the second half of the thesis, we focus on fully observable MDP problems wherein the state of the system at each time point is known. These MDPs are solvable in polynomial time because they can be expressed as special linear programs. However, preferred techniques for solving them are simple dynamic programming value and policy iteration methods. We



analyze these algorithms on restricted classes of problems, including deterministic MDPs and MDPs whose linear programming formulations take two variables per inequality, and establish that several algorithms based on value or policy iteration are polynomial. Tools of analysis include studying parametric versions of the problems and viewing policy iteration as a Newton's method for finding the zero of a function. We expect that extending the analysis of policy iteration as a Newton's method will establish policy iteration to be polynomial on general fully observable MDPs.



## TABLE OF CONTENTS

<b>List of Figures</b>	<b>iv</b>
<b>Chapter 1: Introduction</b>	<b>1</b>
1.1 Markov Decision Processes . . . . .	1
1.2 The Scope of the Thesis . . . . .	4
1.3 Overview of the Thesis . . . . .	9
<b>Chapter 2: Unobservable Markov Decision Processes and Probabilistic           Finite Automata</b>	<b>11</b>
2.1 Introduction . . . . .	11
2.2 Preliminaries and Notation . . . . .	12
2.3 Partially Observable Markov Decision Processes . . . . .	12
2.4 Probabilistic Finite Automata . . . . .	19
2.5 Non-Periodicity of Optimal Action Sequences . . . . .	20
2.6 Several Structural Properties of UMDPs and Open Problems . . . . .	23
2.7 Discussion . . . . .	28
<b>Chapter 3: Undecidability in POMDPs and Probabilistic Planning</b>	<b>30</b>
3.1 Introduction . . . . .	30
3.2 Undecidability of the Emptiness Problem . . . . .	31
3.3 Undecidable Problems in Probabilistic Planning . . . . .	37
3.4 Undecidable Problems for POMDPs . . . . .	39
3.5 Summary and Discussion . . . . .	47

<b>Chapter 4:</b>	<b>Fully Observable Markov Decision Processes</b>	<b>49</b>
4.1	Preliminaries . . . . .	49
4.2	Fully Observable Markov Decision Processes . . . . .	50
4.3	MDPs as Monotone Linear Programs . . . . .	52
4.4	MDP(3) and MDP(2) . . . . .	55
4.5	Iterative Dynamic Programming Algorithms For MDPs . . . . .	59
4.6	An Analysis of Policy Iteration at a High Level . . . . .	63
4.7	Summary and Discussion . . . . .	69
<b>Chapter 5:</b>	<b>Algorithms for MDP(2) and TVPI Problems</b>	<b>71</b>
5.1	Introduction . . . . .	71
5.2	Algorithms Based on Bellman-Ford Style Value Propagation . . . . .	73
5.3	Analysis of Policy Iteration . . . . .	81
5.4	Generalization to Feasibility in TVPI . . . . .	102
5.5	Summary and Discussion . . . . .	112
<b>Chapter 6:</b>	<b>Polynomial Value Iteration Algorithms for Average Reward Deterministic MDPs</b>	<b>115</b>
6.1	Introduction . . . . .	115
6.2	Value Iteration on the Deterministic MDP Problem . . . . .	117
6.3	The Behavior of Value Iteration on Random Graphs . . . . .	128
6.4	Summary and Discussion . . . . .	130
<b>Chapter 7:</b>	<b>Conclusions</b>	<b>135</b>
7.1	Related Work . . . . .	135
7.2	Thesis Contributions and Future Work . . . . .	140
<b>Bibliography</b>		<b>142</b>

<b>Appendix A: Some Properties of Optimal Value Functions and Policies in UMDPs</b>	<b>151</b>
A.1 Contraction Mappings and the Fixed-Point Property . . . . .	151
A.2 Optimal Policies and Value Functions . . . . .	152
<b>Appendix B: The Weak Equality Test</b>	<b>156</b>
<b>Appendix C: Proof of Lemma 5.3.13 on Fast Convergence of the Newton's Method</b>	<b>158</b>

## LIST OF FIGURES

1.1	The basics of a Markov decision process. . . . .	2
1.2	Hierarchy of problems from simpler to more complex, all related to MDP problems. Direct contributions are made to the circled problems. . . . .	5
1.3	The flow of research on algorithm analysis and design for MDPs. Algorithm analysis and insight on problem structure go hand in hand, and insights may suggest new algorithms. Changes in problem definition allows for acquiring insight and successful analysis. . . . .	9
2.1	(a) The 3 dimensional unit simplex which represents possible distributions over 3 states, partitioned into two regions. Action $a$ is optimal in the top region. (b) The evolution of a point toward the stationary point. The pattern of the points of the sequence being above or below the $\frac{4}{11}$ line is irregular. . .	21
2.2	(a) A simple instance of an undiscounted objective where closedness of optimal regions does not hold. The numbers on the edges are the probabilities of the transitions. The only reward is from executing action $b$ in state 2, other rewards being zero. Thus the model can be viewed as a goal-oriented model (Section 2.3.3), where $s_4$ is the goal state. (b) The optimal policy shown as a mapping from probability of being in state 1 versus state 2 to choice of action. In any distribution where state 1 has nonzero probability, action $a$ is optimal. . . . .	25
2.3	(a) Several linear functions in $\mathcal{L}$ from $[0, 1]$ to $[0, 1]$ . Functions $l_1$ and $l_2$ correspond to the identity and permutation matrices respectively. (b) A function $f \in \mathcal{F}$ consisting of three intervals and two border points. . . . .	26

3.1	Outcomes of the weak equality test when $a^i b^j$ is read, (a) when $i = j$ , (b) when $i \neq j$ . While the probability of making a decision is minute, if a decision is made, the chances of Suspect and Correct is the same when $i = j$ , but Suspect is much more likely when $i \neq j$ . . . . .	34
3.2	(a) When the TM is accepting, on an accepting sequence $w$ , while the probability of making an all decisive decision is minute, the PFA will accept or reject with equal probability. By concatenating $w$ , the acceptance probability can be raised to as close to $1/2$ as desired. (b) When the TM is not accepting, on a false accepting sequence $w$ , if the PFA makes a decision, with very high probability it is rejection. . . . .	35
3.3	(a) A PFA, or the criterion of maximizing the probability of reaching a goal state ( $f$ ) in a probabilistic planning model. (b) The PFA emptiness problem modeled as a total reward criterion: The old accepting (goal) state $f$ , on any action, now has a transition to an extra absorbing state giving reward of 1.0. All other rewards are zero. (c) Similarly, a PFA emptiness problem modeled as an average reward problem	41
4.1	(a) An edge (action) in a general MDP model shown as a directed branching (hyper) edge. If the edge is chosen at state $u$ , a reward of 10 units is obtained, and with probability 0.6 the next state is state $v$ , and otherwise it is state $w$ . (b) A hypergraph representation of an MDP problem, where at each state there are one or more edges available. Rewards and transition probabilities are not shown, but understood to label the edges. (c) A policy, <i>i.e.</i> an assignment of single edge to each vertex. . . . .	51
4.2	(a) A single MDP(2) edge referred to as $u$ - $v$ edge, with reward 2 and slope (transition probability .7). The remainder of the transition probability is assumed to go to a zero reward vertex with no choice. (b) An MDP(2) problem graph (the side branches of edges not shown), and (c), an MDP(2) policy. . . . .	56

4.3	(a) Two $u$ - $u$ cycles, $c = e_1e_3$ , and $c' = e_2e_3$ , where $e_1$ and $e_2$ are $u$ - $v$ edges, and $e_3$ is a $v$ - $u$ edge. $e_1$ has reward ( $y$ -intercept) 1, and transition probability (slope) $1/2$ . (b) The value functions $f_c(x) = 5/4 + 1/2x$ , and $f_{c'}(x) = 1/4 + 3/4x$ , corresponding to $u$ - $u$ cycles $c$ and $c'$ shown with the fixed points. . . . .	57
4.4	Value iteration and policy iteration algorithms. . . . .	60
4.5	A legal sequence of policies visited by policy iteration in a three vertex MDP problem. . . . .	67
4.6	An MDP(2) “necklace” graph. . . . .	69
5.1	The Backward-Propagate Algorithm . . . . .	75
5.2	The Forward-Propagate Algorithm. . . . .	76
5.3	$F_{u,v}(\lambda)$ is convex, piecewise linear, and nondecreasing. The breakpoints (points of slope change) are circled. . . . .	82
5.4	(a) An example adag with the bottleneck vertex $s$ duplicated (one with only in-edges and the other with out-edges). (b) Progress of $x_s(t)$ on the adag. . .	86
5.5	The <i>Newton</i> process for finding $x^*$ . . . . .	89
5.6	(a) The progress of the Newton process in finding $x^*$ . (b) Among individual iterates, here between $x^{(1)}$ , $x^{(2)}$ , and $x^{(3)}$ , there may only be relatively small change in distance. . . . .	91
5.7	That $r^{(i)} \leq r^{(i-1)}$ can be seen geometrically, as in the triangle $ACE$ , lines segments $AE$ and $BD$ are parallel, so $\frac{ BG }{ AF } = \frac{ CG }{ CF } = \frac{ GD }{ FE }$ , thus $\frac{ BG }{ GD } = \frac{ AF }{ FE }$ ( $ BG $ denotes the length of segment $BG$ ). This corresponds to the case $r^{(i)} = r^{(i-1)}$ , and otherwise we have $\frac{ BG }{ GD } < \frac{ AF }{ FE }$ . . . . .	92
5.8	(a) Pivoting $l^{(i)}$ around $(x^{(i)}, F(x^{(i)}))$ to obtain $l'$ reduces $x^{(i+1)}$ to $x_{l'}$ . (b) Setting $m^{(0)}$ to zero by moving $h$ up if necessary yields a tight sequence. . . .	93
5.9	(a) Newton’s method. We have $\delta^{(i+1)} < \delta^{(i)}$ and $\alpha^{(i+1)} < \alpha^{(i)}$ , thus $(-m^{(i+1)}) < (-m^{(i)})$ . (b) Again $\delta^{(i+1)} < \delta^{(i)}$ and $\alpha^{(i+1)} < \alpha^{(i)}$ , thus $1 - m^{(i+1)} < 1 - m^{(i)}$ . . . .	97
5.10	The Freezing algorithm . . . . .	100

5.11	(a) The glb graph. (b) The lub graph. All nonzero slope edges in the glb graph are reversed in the lub graph. . . . .	103
5.12	A (generic) iterative-dual algorithm for computing the endpoints. . . . .	103
5.13	(a) When $l_i \leq r_i$ for a variable $x_i$ . (b) When $l_i > r_i$ , which leads to infeasibility.	106
5.14	Iterative algorithms for the glb and, in the parenthesis, the lub phases, which are an extension of policy iteration. . . . .	108
6.1	(a) The optimal cycle (with mean zero here) may not form if ties are broken arbitrarily or if each vertex uses a local ordering of edges. Here, the values labeling the vertices are the ones obtained at the end of the iteration ( <i>i.e.</i> $x_v^{(t)}$ for iteration $t$ ) and used for the next iteration. (b) An example where a vertex may switch forever. In this example, all edges have reward 0, and the vertices on the optimal cycle are initialized with (or somehow have obtained) values 1 and 0. . . . .	123
6.2	(a) An example graph where it takes $\Omega(n^2)$ iterations for the highest values to reach the optimal cycle. (b) An example where Gauss-Siedel value iteration converges to a cycle (here the two-edge cycle) other than the highest average reward cycle, irrespective of the order of processing the vertices. The tables show the value of the vertices in each iteration with respect to the two possible processing orders. . . . .	124
6.3	(a) Averages length of optimal cycles and the first iteration until an optimal cycle is found. (b) A comparison of the run times of Karp's algorithm versus value iteration using history walks and value iteration using visited policies. .	129

## ACKNOWLEDGMENTS

I thank all my advisors for their guidance, patience, encouragement, and collaboration. Steve Hanks was always available with ideas on specific problems and general advice on making hard decisions such as choosing the general directions of research. I am also very grateful to him for his very generous support of my work through the years and allowing me to explore and pursue my own choice of problems. Richard Anderson was very effective in giving ideas and encouraging me on several key fruitful directions on my research on analysis of algorithms for MDPs. Anne Condon has been very helpful and encouraging throughout my PhD research, and especially on the undecidability work. I am thankful to Steve Hanks, Michael Littman, Anne Condon and others whose work on planning and the complexity of MDPs motivated my work. I have also benefited from many discussions with and suggestions from my immediate advisors, and also Dick Karp, Jim Burke, Michael Littman, Paul Beame, Erik Hansen and many other colleagues.

Through my graduate career at the University of Washington, I have gained a lot from interaction with many students and faculty, in particular in the artificial intelligence and the theory groups. I am especially indebted to professors Oren Etzioni, Larry Ruzzo, and Steven Tanimoto for my early research experiences. Adam Carlson, my officemate for several years, and other fellow “Chateau” dwellers and in general all the friends in the department, have made my life in this period very warm and pleasurable. At the risk of forgetting several, these friends include: Brad Chamberlin, William Chan, AnHai Doan, James Fix, Mark Friedman, Brian Grant, Matthai Philipose, Brendan Mumey, E Christopher Lewis, Vass Litvinov, Sujay Parekh, Kurt Partridge, Frederick Pighin, Kari Pulli, Erik Selberg, Gun Sirer, Jeff Voelker, Peter Van Vleet, and Oren Zamir.

I have been very fortunate to have a great amount of family and extended-family support in my life and education. I owe it all to my parents for their love and continual guidance

and encouragement, and their sacrifices for all their children. My sister Navid and brother in-law Kirk have been a constant source of support and advice throughout the years. I am also indebted to many other family and friends, in Seattle and elsewhere, for their support and friendship through my education. Finally, I thank my fiancée Anila and her family for their love and care, and for adding another welcome dimension to my life in the last two years of graduate school.

**DEDICATION**

To my parents

## Chapter 1

## INTRODUCTION

*I've laid my plans*

*Now lay the chance*

(Whitney Houston, in the lyrics of “one moment in time”)

Control, decision making under uncertainty, and long term planning, coupled with adaptation, are the stuff of intelligent behavior. Not surprisingly, corresponding problems have drawn considerable attention and contribution from many fields including applied mathematics, management and operations research, engineering, computer science and artificial intelligence. Consequently, these problems are often rich in models, rich in mathematical theory and solution methodologies, and rich in applications. In this thesis, we explore some computational aspects of one such model of control—having a well developed theory of its own and wide applicability—called the Markov decision process. Our treatment will be from a computational complexity theory point of view.

### 1.1 *Markov Decision Processes*

The work in this thesis concerns a basic model of dynamic decision making under uncertainty called the *Markov decision process* (MDP). Broadly, it consists of a *system* composed of a set of *states* and a number of *actions* available at each state. *Time* is discretized and at each time point the system is in a single state. The MDP is a sequential decision model: at every time point, an action is executed by a *decision maker*, a *reward* is obtained (or a cost incurred), time increments, the system changes state, and an *observation* containing information on the new system state is made; then this basic sequence, shown in Fig. 1.1 (adapted from Puterman [103]), repeats. The general objective for controlling the process is

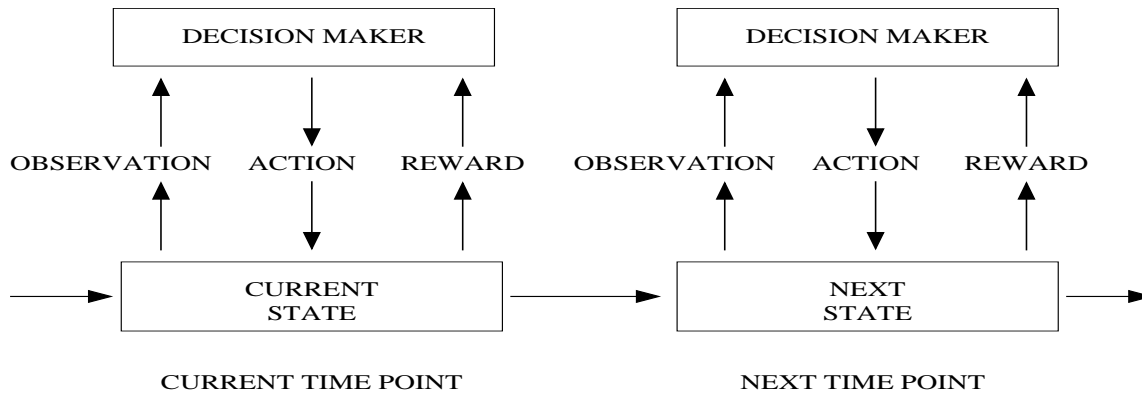


Figure 1.1: The basics of a Markov decision process.

to find a *policy*, *i.e.* a prescription of choice of actions based on the information available at each time point, so that a level of performance is achieved based on the rewards obtained, if actions are chosen according to the policy.

An important characteristic of the model is the so-called *Markov property*, from which MDPs inherit their name. The Markov property states that the next state of the system, and the reward attained and the observation made, depend only on the current state of the system and the action executed, and not, for example, on the history of previous system states and executed actions. The change in state, however, can be probabilistic, adding an element of uncertainty to the model. The Markov property is a very simplifying and powerful assumption about the system, as it makes model analysis and development of solutions tractable.

In this work, we study problems where the state and action sets are finite. The control objectives of interest to us concern measures of reward obtained over the long run: over an indefinite number of time points or action executions. These measures are referred to as *infinite-horizon* criteria in the MDP literature, and common ones we will be using are *expected total reward* and *expected average reward per time step*. Our focus will be on *stationary* Markov processes, meaning that the effect of actions (the rewards obtained and the state transitions) do not change with time.

An important model variation, over which our work is partitioned, is the observability

of the system state. In the *fully observable* MDP model, the assumption is that the state of the system is known to the decision maker at each time point. In Fig. 1.1, the fully observable MDP would correspond to the observation revealing the identity of the system state. Thus, in this case, the only source of uncertainty is the stochastic effect of actions in state transitions. In the *partially observable* (POMDP) generalization, there is an additional source of uncertainty: the observation may provide partial or incorrect information on system state. Not surprisingly, partial observability not only reduces the performance of policies but also makes the task of computing good policies harder than the fully observable variant. In this thesis, when we refer to an MDP problem, we mean the fully observable variant, but we also use the term to refer to the broad model.

We assume all system characteristics, such as the number of states and the effects of actions, are given in the problem formulation. Precise definitions of the problems we study are given in subsequent chapters.

### 1.1.1 Background and Applications of MDPs

The MDP is a general model with a rich mathematical theory and encompasses a variety of problems with numerous applications. Chapter 1 of Puterman [103] describes many examples of how fully observable MDPs are used as modeling tools. These include inventory management and machine maintenance, applications in behavioral ecology, and games. The book contains an extensive bibliography on the history of MDPs. White [122] also gives many examples where fully observable MDPs have been applied in real world problems.

The POMDP generalization was first formulated for the problem of decoding signals received across a noisy channel by Drake [36]. Perhaps most problems modeled as fully observable MDPs are more realistically modeled as partially observable MDPs. Thus POMDPs have been considered in areas such as system inspection, maintenance, and management, with a large body of literature, from which we refer the reader to a very few [101, 75, 117, 61]. General references on POMDPs include surveys by Monhan [90] and Lovejoy [79], and books by Bertsekas [9] and White [123].

In AI research, MDPs have been recently adopted as a framework for acting and planning

under uncertainty. For instance, probabilistic planning problems [74, 12] are a significant example of POMDPs, which motivated our analysis of infinite-horizon POMDPs. Some machine learning models—for example reinforcement learning—view the underlying world model as a POMDP [85, 63]. The interest in MDPs has motivated considerable research in this area with more emphasis on the computational aspects (algorithms and computational complexity) [18, 49, 76] than previously. Recent applications of POMDPs in AI include stochastic control and navigation problems [71, 72], medical diagnosis and therapy [53, 120], and control of attention in a machine vision system [28].

## 1.2 *The Scope of the Thesis*

We investigate the computational aspects of infinite-horizon fully observable and partially observable MDP problems using (computational) complexity theory. Complexity theory (see, for example, the introductory book by Papadimitriou [96]), especially in its common form of asymptotic worst case complexity, has proven to be an invaluable methodology for understanding the resource requirements of computational problems, formalizing intuition on the hardness of problems, in pointing to ways for best approaching, formulating, and solving many of them, and in generating novel algorithmic ideas.

While work on MDPs and POMDPs dates back to as early as the 40’s and 60’s respectively, the study of their computational complexity is relatively recent. Research in theoretical computer science on the power of randomized computation, in settings such as algorithm design, interactive proof systems, and games against nature, motivated the first work on the analysis of the complexity of MDP problem variants [98, 22, 23]. More recently, as research in AI has incorporated uncertainty as an indispensable component of many problems, researchers have further investigated the computational complexity of various MDP problems and algorithms [76, 13, 93, 84]. A paper by Goldsmith and Mundhenk [45] surveys complexity results for many MDP problems and points to new directions.

This thesis investigates questions raised in the first paper by Papadimitriou and Tsitsiklis on complexity of MDPs [98], and which later work (*e.g.*, [23, 89, 76, 84]) further motivated. In the first part of the thesis, we address the computability of infinite-horizon POMDPs.

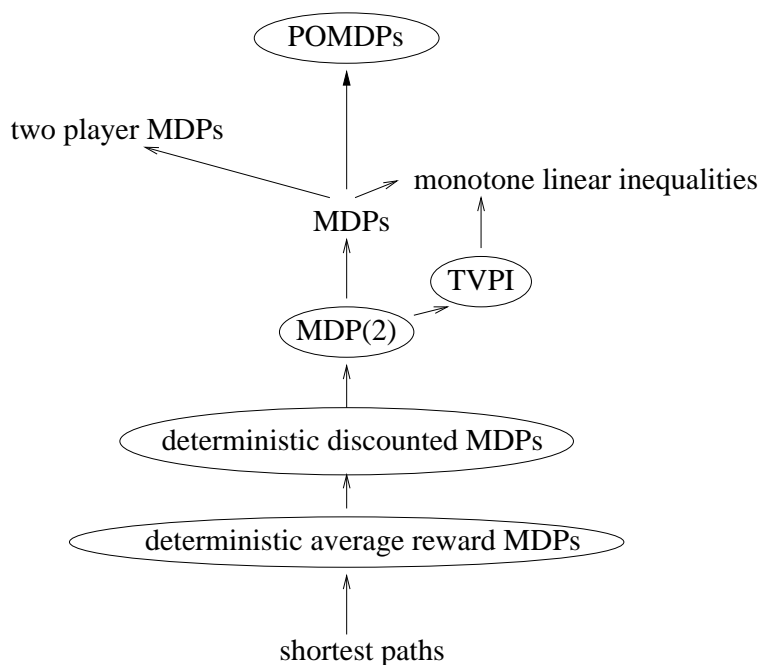


Figure 1.2: Hierarchy of problems from simpler to more complex, all related to MDP problems. Direct contributions are made to the circled problems.

In the second part, we investigate polynomial time algorithms for infinite-horizon fully observable MDPs and related problems.

Fig. 1.2 shows the relative position of problems we have investigated within context, roughly from easier to harder problems. Definitions for the problems are given in subsequent chapters. Our direct contributions are with respect to the circled problems. We have shown the undecidability of problems in infinite-horizon POMDPs. For the rest of the circled problems, we have established that several algorithms, all within the family of dynamic programming<sup>1</sup> value iteration and policy improvement methods, are polynomial time. We note that the deterministic MDP under the average reward criterion, or equivalently the

---

<sup>1</sup>The term dynamic programming, coined by Richard Bellman (see his autobiography [8], page 159 for an interesting account of its origin) has slightly different semantics in computer science and general optimization. In computer science, it usually refers to the technique of solving problems by recording and combining solutions to subproblems (see, for example, [25]). We are using the original and more general semantics used in optimization, and common in MDP literature, as a stage-wise or time-varying optimization technique.

minimum or maximum mean cycle problem, and the feasibility problem in linear systems with two variables per inequality (TVPI) find many applications, for example, in combinatorial network flow algorithms and system performance analysis [44, 43, 21, 95, 121, 30]. The algorithms analyzed and developed are simple to describe and implement, and we expect that they will be among the most efficient algorithms in practice as well. A more detailed description of the work and some of the main contributions of the thesis follows.

### 1.2.1 Undecidability in POMDPs

Our work on computability of POMDPs was motivated by the question of whether (finite-time) algorithms exist for answering some basic questions that naturally arise in solving POMDPs. These problems appeared to be undecidable (mentioned first in [98]), but undecidability remained a matter of conjecture. Recent work on probabilistic planning, which is a special POMDP problem, further motivated investigating such problems. The probabilistic planning problem gives a good example of the basic problem in the area. In this problem, truth values to a set of propositions define system states, and actions change values of the propositions, and thus change states, stochastically. A basic problem here is, given the current state, and a desired goal state, is there a plan, *i.e.* a sequence of actions, whose execution causes the state of the system after the last action execution to be the goal state, with probability exceeding a given threshold. Such a *plan existence* problem arises in any methodology that attempts to find plans with high success probability in probabilistic planning. This plan existence problem and a collection of other problems, including questions of approximability under several optimization objectives, are shown undecidable in our work. Along the way, a few other properties of POMDPs, pointing to ease or hardness of solving these problems, especially under the important discounted criterion (defined in Chapter 2) are explored. The focus of the chapter will be on the extreme case of unobservable MDPs, which simplifies the complexity analysis.

In establishing undecidability, we make connections to undecidability of the language emptiness problem for probabilistic finite automata (defined in Chapter 2), first shown undecidable by Paz [99]. We make use of the undecidability proof given by Lipton and Condon

[24]. This latter work is developed in the context of research on the power of randomization in various resource bounded computational models, and in particular it is based on a technique developed by Freivalds [42], who shows that finite automata with randomized computations—unlike their deterministic counterparts—have a semi-counting ability. This ability, further developed and adapted, is at the heart of the reduction of Lipton and Condon [24]. A contribution of the thesis in this part, reflected in the many problems shown undecidable, is in exhibiting that the reduction is very applicable and extensible. We explain the reduction and explore its properties in order to use it in our undecidability results.

### 1.2.2 *Efficient Algorithms for MDPs*

Finding an optimal policy for a fully observable MDP is a classical problem in optimization theory. It is known that the problem can be solved as a linear program (LP) in polynomial time [58, 69], but no other polynomial time algorithm is known, which leaves our state of understanding of polynomial time algorithms for these problems at an inadequate level. There are other motivating factors for studying efficient algorithms for the problem. For example, it is shown by Condon [22] that a two player MDP problem is complete for the class of languages accepted by log space bounded randomized alternating Turing machines and that it is in the class  $NP \cap coNP$ , but it is open whether it is in the class  $P$  (*i.e.* whether it can be solved in polynomial time). The problem has a policy improvement algorithm that exhibits similar good behavior as its counterpart for (single player) MDPs, and it appears that analyses of efficiency of algorithms on one problem are likely to shed light on another. As we will see in this thesis and shown in Fig. 1.2, the study of efficient algorithms for MDPs has positive consequences for related problems as well.

Our work on analysis and design of algorithms focuses on a subclass of MDPs we refer to as MDP(2). We identify MDP(2) as a special case of the feasibility problem in linear programs with two variables per inequality (TVPI) and show that existing combinatorial algorithms for TVPI apply to MDP(2). Our main contribution, however, is in analyzing iterative dynamic programming methods on MDP(2) and its simpler deterministic MDP variants. Our contributions, in terms of new algorithms, are:

- Polynomial policy improvement algorithms for MDP(2).
- Strongly polynomial algorithms—variants of value iteration and policy improvement—for deterministic average reward and discounted MDPs.
- New polynomial time algorithms—generalizations of the policy improvement variants—for TVPI.

Parametric analysis and viewing policy improvement methods as Newton’s method for finding the zero of a convex function—a function implicitly defined in the problem—have provided the framework for establishing the positive results on policy improvement on MDP(2). Details on these analysis techniques are given in subsequent chapters. There exist many open problems in improving the analyses, in the design of new algorithms, and in extending the analysis techniques to harder problems, which we list in later chapters.

It is instructive to consider the flow of the research. The challenging aspect of this work has been mostly in the analysis of existing algorithms rather than developing new ones. The difficulty has been identifying aspects of the interaction between an algorithm under study and the problem structure, and formulating a measure of progress of the algorithms that would lead to proof of polynomial running time (assuming this were the case). The algorithm analysis and design portion of this thesis has followed the diagram in Fig.1.3 at several times. Thus the motivation to analyze an existing algorithm has led to problem simplifications, which has allowed for discerning successful approaches to analysis and insight on problem structure. Insights gained, in turn, have led to new analyses, new algorithms, and problem variations for further exploration.

The work flow confirms a statement made by Gusfield and Irving, in the preface of their book on algorithms for the Stable Marriage Problem [48], that there is a “productive and almost inseparable relationship between mathematical insight [into problem structure] and the design of algorithms.” In our case, an existing problem (the MDP) and an existing algorithmic technique (policy improvement) motivated the research, and many of the structural properties of the problems are only meaningful or useful in the presence of the algorithms that lead to their discovery. In turn, however, these discoveries of structure can

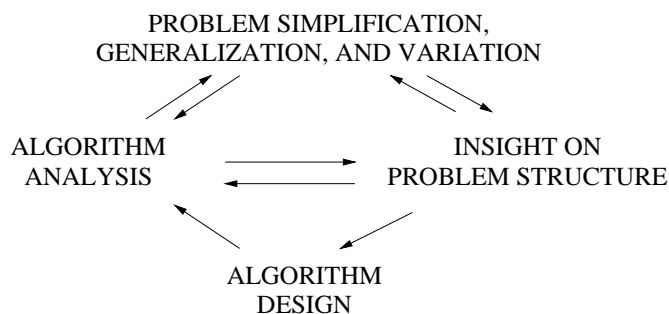


Figure 1.3: The flow of research on algorithm analysis and design for MDPs. Algorithm analysis and insight on problem structure go hand in hand, and insights may suggest new algorithms. Changes in problem definition allows for acquiring insight and successful analysis.

lead to superior algorithms and possibly the discovery of further structure, as will be seen for example in our analysis of the deterministic average reward MDP problem. We expect that the results in this work will extend and lead to further exploration and discovery of efficient algorithms in the dynamic programming family applicable to MDPs and similar problems.

### 1.3 Overview of the Thesis

The remainder of the thesis is organized as follows.

- Chapter 2: We give basic definitions for POMDPs, with the focus on the unobservable case (UMDP), and probabilistic finite automata. We also explore a few properties of UMDPs—mainly under the discounted objective—and in particular, we describe a counterexample showing that a plausible and desirable *periodicity* property, defined in the chapter, does not hold.
- Chapter 3: We show the undecidability of various POMDP problems, including in-approximability under several optimization objectives. We make use of the reduction in Lipton and Condon [24] throughout the chapter, and the reduction is described extensively.

- Chapter 4: We define fully observable MDPs, the MDP(2) subclass and TVPI, and describe some of their properties and existing dynamic programming algorithms for solving MDPs. This chapter also includes a first attempt, albeit an unsuccessful one in showing a subexponential upper bound on the run-time, at analysis of *policy iteration* (a common policy improvement algorithm).
- Chapter 5: Algorithms for MDP(2) and the related TVPI problems are explored. We present our analysis of the policy iteration algorithm, using parametric analysis and viewing policy iteration as a Newton's method. This analysis leads to polynomial time algorithms for MDP(2) and TVPI problems.
- Chapter 6: We explore the behavior of value iteration on the deterministic average reward MDP (DMDP) problem defined in the chapter. This work is motivated by a promising approach to an open problem in Chapter 5. The analysis leads to discovery of structural properties and new polynomial time algorithms for DMDPs based on value iteration.
- Chapter 7: We close with a summary of the thesis, and a discussion of related work and directions for future research.

## Chapter 2

**UNOBSERVABLE MARKOV DECISION PROCESSES AND  
PROBABILISTIC FINITE AUTOMATA**

In this chapter, we describe the partially observable and the unobservable Markov decision process (UMDP) and probabilistic finite automata models and computational problems. In addition to providing the preliminaries for the undecidability results in the next chapter, we explore several properties of UMDPs as part of our work toward obtaining insight into the computability of unobservable problems.

**2.1 Introduction**

In this chapter, we begin by describing the partially observable Markov decision process (POMDP) model and several infinite-horizon optimality criteria. Our focus will be on a subclass of the POMDP, the unobservable MDP (UMDP) model. We describe the notions of action sequences and their quality and optimality under several optimality criteria. Each optimality criterion yield its own optimization objectives and problems of interest. We give a few specific problem definitions in Section 2.3.4 for the discounted model. Problems under other criteria are similarly posed. We also define the closely related probabilistic finite automaton model in Section 2.4.

In the rest of the chapter, we explore structural properties of the UMDP problem, especially under the discounted criterion. In particular, we describe a certain property of *periodicity* of optimal action sequences, such that if a UMDP problem has that property, it simplifies finding optimal solutions. We give a three-state UMDP example for which the periodicity property does not hold. The example is a candidate for a simple POMDP problem that has irrational optimal value (even if all input parameters are rational), thus pointing out the difficulty in computing optimal solutions. The question of whether POMDPs with

irrational optimal values exist was also raised by Littman [76]. We also show in Section 2.6.1 that under similar conditions, the two-state unobservable problem enjoys the periodicity property. Several open problems are presented in Section 2.5, including a conjecture on the decidability of the two-state discounted infinite-horizon UMDP problem. We conclude with a discussion of the open problems.

## 2.2 Preliminaries and Notation

We assume the input parameters in all problems are rational numbers. We will be using matrices and vectors at several points, but only familiarity with elementary linear algebra is needed. We use boldface font to denote vectors. For a vector  $\mathbf{x}$ ,  $\mathbf{x}^T$  denotes its transpose, and  $\mathbf{x}[i]$  denotes its  $i$ th component.  $\mathbf{M}[i, j]$  denotes the entry in row  $i$  and column  $j$  of matrix  $\mathbf{M}$ , and  $\mathbf{M}[i, \cdot]$  denotes the vector corresponding to row  $i$  of the matrix.

Sequences of different objects such as scalars and vectors are used frequently in the thesis, and we use the parenthesized superscript notation to refer to them. For example,  $o^{(j)}$  denotes the  $j$ th object of the sequence  $o^{(1)}, o^{(2)}, \dots$ , and  $\{o^{(i)}\}$  refers to the whole sequence. Sequences may be finite or infinite.

We will use  $\Sigma$  to denote a finite set of symbols (an alphabet). In this case, a sequence is treated as a juxtaposition (or concatenation) of the symbols of  $\Sigma$ , for example,  $w = \sigma_1\sigma_2\cdots$ , where  $\sigma_i \in \Sigma$ . A string is a finite sequence, possibly empty. If  $w$  and  $w'$  are two sequences, where  $w$  is finite, then  $ww'$  is the sequence corresponding to their concatenation,  $w^k$  denotes  $w$  concatenated with itself  $k$  times, and  $w^\infty$  denotes the infinite concatenation of  $w$  with itself, or  $w^\infty = wwww\cdots$ . With sequence  $w = \sigma_1\sigma_2\cdots$ , where  $\sigma_i \in \Sigma$ ,  $w_i$  denotes the  $i$ th element of  $w$ , or  $w_i = \sigma_i$ ,  $|w|$  denotes the length of sequence  $w$ , *i.e.* the number of symbols in  $w$ , and  $pre(w, i)$  denotes the length  $i$  prefix of  $w$ , thus  $pre(w, i) = \sigma_1\sigma_2\cdots\sigma_i$ .  $\Sigma^*$  denotes the set of all finite sequences (strings) of elements in  $\Sigma$ .

## 2.3 Partially Observable Markov Decision Processes

A Markov decision process (MDP) model consists of a set  $Q$  of  $n$  states, and a finite set of actions,  $\Sigma$ . Time is discretized and at each time point  $t$ ,  $t = 0, 1, \dots$ , the system occupies

a single state in  $Q$ , which is called the (current) state of the system at time  $t$ . The means of change in system state is action execution, and there is uncertainty about the outcome of actions. Consider the states indexed:  $s_i \in Q, 1 \leq i \leq n$ . Associated with each action  $a \in \Sigma$  is an  $n \times n$  stochastic matrix  $\mathbf{M}_a$  which specifies the state transition probabilities for action  $a$ .  $\mathbf{M}_a[i, j]$  has the semantics that if the state of the system is  $s_i$  at a given time point  $t$  and action  $a$  is executed, with probability  $\mathbf{M}_a[i, j]$  the state of the system at time point  $t+1$  is  $s_j$ . We refer to  $\mathbf{M}_a$  as the transition matrix or the dynamics (matrix) of action  $a$ . Also associated with each action  $a$  is an  $n \times 1$  vector of rewards  $\mathbf{R}_a$ , with the semantics that the decision maker gains  $\mathbf{R}_a[i]$  (or incurs cost  $\mathbf{R}_a[i]$  when  $\mathbf{R}_a[i] < 0$ ), if the state of the system is  $s_i$  and action  $a$  is executed. MDP models satisfy the *Markov property*: the Markov property, from which MDPs get their name, says that the next state of the system, and other aspects such as the reward attained and the observation made (see below), depend only on the current state of the system and the action executed, and not, for example, on the history of previous system states and executed actions.

The states and actions (basically the transition matrices and the reward vectors) are completely specified as part of the problem instance. Informally, the problem of a decision maker, confronted with such a system, is to execute actions that maximize some measure of accumulated reward—for example, the expected total reward—over some fixed number of time steps or over an indefinite number of time steps. Observe that in making a choice of action, not only the immediate reward of an action is important, but also the state the action leads to may be significant in obtaining a high reward in the future.

The reader is referred to books on MDPs and POMDPs, such as Puterman [103], White [123], and Bertsekas [9], for a more comprehensive introduction to the models and motivational examples. We will next describe two important aspects, state observability and the time horizon, over which model specifications vary, and define a few computational problems.

### 2.3.1 Observability

In the traditional *fully observable* MDP model, the system state at each time point is known to the decision maker. In the partially observable (POMDP) generalization, the decision maker may have only partial information on the system state, for example a probability distribution over possible states. Consequently, in the POMDP model, there are two sources of uncertainty: uncertainty about the outcome of actions and, at each time point, uncertainty over the current system state. In general, partial observability makes the life of the decision maker, in terms of attaining high rewards, difficult, as the current system state may not be known, and, for example, the decision maker may need to act conservatively. Furthermore, as states and action specifications are given, and only the state at any given time point may not be known with certainty, an algorithm for finding optimal actions may need to take contingencies (different outcomes for different possible current states) into account, and in general the task of selecting good actions becomes harder as well.

Generally in POMDPs, after each action execution, the decision maker obtains some feedback, albeit imperfect, on the current state of the system. In common POMDP models, a state  $s_i$  emits an *observation* (signal)  $O_j \in \mathcal{O}$ , where  $\mathcal{O}$  is a finite observable set. The source of partial observability is the possibility that multiple states may emit the same observation. Traditionally, the observation is modeled as a random variable of the state or state and action. The distributions of the random variables are given as part of the problem instance. The fully observable MDP model corresponds to an extreme case where each state maps to a unique observable, and it will be the focus of Chapters 4 and later.

In this chapter and the next, we focus on another extreme of partial observability, the unobservable MDP (UMDP) model, where all states map to the same observation. The UMDP model simplifies our analyses of computability and is closely related to the probabilistic finite automaton model defined in Section 2.4. Of course, the hardness results that we establish in the next chapter apply to the more general POMDP variations as well.

We always assume that the decision maker is given a probability distribution over the initial state. For the undecidability results, the decision maker may even know the initial state. Let a state probability distribution at time  $t$  be denoted by the  $1 \times n$  vector  $\mathbf{x}^{(t)}$ ,

$\mathbf{x}^{(t)}[i]$  is the probability that the system is in state  $s_i$ . Given probability distribution  $\mathbf{x}^{(t)}$  at time  $t$ , and execution of action  $a$  at time  $t$ , the probability distribution at time  $t + 1$  is  $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}\mathbf{M}_a$ . Thus by knowing current probability distribution and action, the next probability distribution is easily computed. Therefore in UMDPs, the distribution  $\mathbf{x}^{(t)}$ , computed from the initial probability distribution  $\mathbf{x}^{(0)}$  and the sequence of actions executed till time  $t$ , is the only information on system state at time  $t$  available to the decision maker.

### 2.3.2 Infinite-Horizon Criteria

In this thesis, we will be interested in *infinite-horizon* optimization objectives or criteria, that is, we assume the decision maker executes actions in the system over an indefinite or an unbounded number of time steps, and in case of UMDPs we will consider both finite and infinite sequence of actions. This contrasts with *finite-horizon* objective, where the decision maker has a specified time limit in executing actions. The complexity of finite horizon POMDPs has been extensively studied; see for example [98, 76, 92, 93].

### 2.3.3 Measures of Value of Action Sequences

In solving UMDPs, we need a measure of value of an action sequence to formulate optimization objectives and computational problems. We will sometimes refer to a choice of value measure as an *optimality criterion*. Once a value measure is defined, an optimization objective becomes selecting the action sequence with the highest value in the collection of possible action sequences. We will now discuss several value measures for action sequences and introduce notation for expressing them.

Firstly, execution of an action  $a$ , given distribution  $\mathbf{x}$  over system state, gives an expected reward of  $\sum_{1 \leq i \leq n} \mathbf{x}[i]\mathbf{R}_a[i]$ , which can be expressed compactly using vector product notation as  $\mathbf{x}\mathbf{R}_a$ .

Let  $w$  denote a nonempty, finite or infinite, sequence of actions. By execution of the sequence  $w$  we mean that the  $i$ th action,  $w_i$ , is executed at time  $t = i - 1, i \geq 1$ , so  $w_1$  is executed at time 0,  $w_2$  at time 1, and so on. Let us first express the total reward value of  $w$ , which is the total expected reward obtained when initial system distribution is  $\mathbf{x}^{(0)}$  and

$w$  is executed. Let  $\mathbf{R}_{w_i}$  and  $\mathbf{M}_{w_i}$  denote the reward vector and the dynamics matrix of the  $i$ th action in  $w$  respectively. Then  $\mathbf{x}^{(i-1)}\mathbf{R}_{w_i}$  is the expected reward from executing the  $i$ th action  $w_i$  at time  $t = i - 1$ . Note that  $\mathbf{x}^{(i)}$  is determined by  $w$  and  $\mathbf{x}^{(0)}$ :  $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)}\mathbf{M}_{w_i}$ . We use the function  $\mathcal{V}$  to denote the expected reward of  $w$  when initial distribution is  $\mathbf{x}^{(0)}$ . We have:

$$\mathcal{V}(w, \mathbf{x}^{(0)}) = \sum_{i=1}^{|w|} \mathbf{x}^{(i-1)}\mathbf{R}_{w_i}, \quad (2.1)$$

where if  $|w| = \infty$ , it is understood that the limit is taken, if well defined. This expected total reward measure is called the *total reward criterion*. It is always well defined for the special case where a reward is obtained at most once upon entering a specially designated “goal” state. We call this special criterion the *goal-oriented* model or criterion. AI probabilistic planning problems, for example [74], with traditional goal-oriented objectives, can be modeled this way. Also, our undecidability results of the next chapter are based on reductions from goal-oriented problems. An example goal-oriented model is given in Fig. 2.2 on page 25.

Another common variation on the optimality criterion, called the *total discounted reward* value, or the *discounted criterion* for short, is always well defined in the presence of arbitrary reward and state transition structure. Under this criterion, a discount factor  $\beta$ ,  $0 \leq \beta < 1$ , is used to discount future rewards. The value of a sequence  $w$ , at an initial distribution  $\mathbf{x}^{(0)}$  is then expressed as:

$$\mathcal{V}_\beta(w, \mathbf{x}^{(0)}) = \sum_{i=1}^{|w|} \beta^{i-1} \mathbf{x}^{(i-1)}\mathbf{R}_{w_i}, \quad (2.2)$$

The *average reward* criterion is another common and natural measure. We will define the average reward criterion in the next chapter. In this chapter, our focus will be on properties of UMDPs under the discounted criterion.

#### 2.3.4 Computational Problems

Under any optimality criterion a number of computational problems of interest suggest themselves. Let us formulate several such problems under the infinite-horizon discounted criterion. A discounted UMDP model is specified by the sextuple  $\mathcal{M} = \{Q, \Sigma, \{\mathbf{M}_a\}, \{\mathbf{R}_a\}, \mathbf{x}_0, \beta\}$ ,

where  $Q$  and  $\Sigma$  are the states and actions of the models,  $\{\mathbf{M}_a\}$  and  $\{\mathbf{R}_a\}$  denote the set of transition dynamics and rewards corresponding to the actions,  $\mathbf{x}_0$  denotes the initial distribution, and  $\beta$  is the discount factor. A simple decision problem, which we call the *sequence existence problem* is then:

**Problem 2.3.1** *Given a discounted UMDP  $\mathcal{M}$ , and a threshold  $\tau$ , is there a finite action sequence  $w$ , such that  $\mathcal{V}_\beta(w, \mathbf{x}^{(0)}) > \tau$ ?*

This sequence existence problem is the main problem shown undecidable in chapter 3. Similar problems can be specified under other criteria. Let us next define optimal values and optimal action sequences under the discounted model, in order to describe optimization problems of interest. Let  $\mathcal{X}$  be the set of  $n \times 1$  vectors of probability distribution over system state:  $\mathcal{X} = \{\mathbf{x} | \mathbf{x}[i] \geq 0, \sum_{i=1}^n \mathbf{x}[i] = 1\}$ .  $\mathcal{X}$  is known as the  $n$  dimensional unit simplex in  $\mathcal{R}^n$ . The three dimensional unit simplex is shown in Fig. 2.1 on page 21. Let the *optimal value function*  $\mathcal{V}^* : \mathcal{X} \rightarrow R$ , be defined as:

$$\mathcal{V}^*(\mathbf{x}) = \sup_{w \in \Sigma^*} \mathcal{V}_\beta(w, \mathbf{x}) \quad (2.3)$$

It is not hard to see that the function  $\mathcal{V}^*$  is well defined over any  $\mathbf{x} \in \mathcal{X}$  as  $\mathcal{V}^*(\mathbf{x})$  is bounded from above and below at any  $\mathbf{x}$ : the execution of any action at any time point falls in  $[-R, +R]$ , where  $R$  denotes the highest reward in absolute value over components of action reward vectors, thus for any  $w$ , its value  $\mathcal{V}_\beta(w, \mathbf{x})$  is bounded above and below by  $\pm R \sum_{i=0}^{\infty} \beta^i = \frac{\pm R}{1-\beta}$ . Put in words,  $\mathcal{V}^*(\mathbf{x})$  is the highest value in expectation attainable if the initial distribution is  $\mathbf{x}$ . It is also the case that there is some action sequence  $w$ , which we call an *optimal action sequence*, that has value  $\mathcal{V}^*(\mathbf{x})$  when executed with the initial distribution being  $\mathbf{x}$ . Also, it is not hard to give examples where no finite action sequence achieves the optimal value, *i.e.* all optimal sequences have infinite length. Theorem 2.3.2 summarizes some of the properties of the optimal value function under the discounted criterion. The results are standard in POMDP literature, see for example books by White or Bertsekas [123, 9]. In appendix A, some of these properties are proved for purposes of illustration. Similar results hold under other optimality criteria.

**Theorem 2.3.2** *Consider a UMDP problem under the discounted criterion. The optimal value function  $\mathcal{V}^*$  is bounded, continuous and convex over the unit simplex  $\mathcal{X}$  of probability distributions over the states. At each probability distribution  $\mathbf{x} \in \mathcal{X}$ , there is an action sequence  $w$ , called an optimal action sequence, such that  $\mathcal{V}^*(\mathbf{x}) = \mathcal{V}_\beta(w, \mathbf{x})$ .*

### *Policies*

In many POMDP problems we are interesting in computing a *policy*. A policy, in general, is a prescription of action as a function of the information available to the decision maker. In the UMDP case, we may think of action sequences as policies that specify the action to take as a function of the time point. Another common representation of a policy in a general POMDP is a mapping from the simplex of probability distributions  $\mathcal{X}$  to the set of actions. Such a policy is called a *stationary policy* because whenever the state-distribution is at a certain point  $\mathbf{x}$  the same action is prescribed. It is known that considering only the space of such policies is sufficient for achieving optimal behavior (obtaining optimal values) for infinite-horizon POMDPs. See for example [116]. A computational POMDP problem would then be, given a POMDP model, find an optimal policy in this representation. An initial distribution need not be specified, as the optimal mapping is found for all possible initial distributions, so this is a more general and demanding problem than the problems we considered previously. In Section 2.5, and in appendix A, we study several properties of UMDPs with this representation of policies in mind, but in the next chapter we basically view action sequences as the desired output for solving UMDPs.

Other representation of policies exist, such as finite controllers (see for example [49]). We discuss finite controllers in the Section 2.7 and in the next Chapter.

In summarily, UMDP optimization problems include, given a UMDP and an initial distribution  $\mathbf{x}^{(0)}$ , computing  $\mathcal{V}^*(\mathbf{x})$ , computing a finite representation of an optimal action sequence, or computing a finite prefix, for example the first action of the sequence. Throughout this chapter and next we always assume that a UMDP problem includes the distribution of initial state in the input.

## 2.4 Probabilistic Finite Automata

A probabilistic finite-state automaton (PFA)  $\mathcal{M}$  is defined by a quintuple  $\mathcal{M} = (Q, \Sigma, T, s_1, s_n)$  where  $Q$  is a set of  $n$  states,  $\Sigma$  is the input alphabet,  $T$  is a set of  $n \times n$  row-stochastic transition matrices, one for each symbol in  $\Sigma$ ,  $s_1 \in Q$  is the initial state of the PFA, and  $s_n \in Q$  is an *accepting* state. A PFA has the dynamics of a UMDP. The automaton occupies one state from  $Q$  at any point in time, and at each stage transitions stochastically from one state to another. The state transition is determined as follows:

1. The current input symbol  $a$  determines a transition matrix  $\mathbf{M}_a$ .
2. The current state  $s_i$  determines the row  $\mathbf{M}_a[i, \cdot]$ , a probability distribution over the possible next states.
3. The state changes according to the probability distribution  $\mathbf{M}_a[i, \cdot]$ .

An example PFA is described in Section 2.5.1. In the next chapter, we restrict attention to PFA in which the accepting state  $s_n$  is absorbing:  $\mathbf{M}_a[n, n] = 1.0, \forall a \in \Sigma$ . This is equivalent to the assumption of the automaton halting upon transition to the accepting state  $s_n$ .

We say a PFA  $\mathcal{M}$  *accepts* the string  $w \in \Sigma^*$  if the automaton ends in the accepting state upon reading the string  $w$ , otherwise we say it *rejects* the string. We denote by  $p^{\mathcal{M}}(w)$  the acceptance probability of string  $w$  by PFA  $\mathcal{M}$ . Note that for any infinite sequence  $w$ , the acceptance probabilities of the prefixes  $p^{\mathcal{M}}(\text{pre}(w, i))$  do not decrease with  $i$  as the accepting state is assumed to be absorbing, and consequently the limit  $\lim_{i \rightarrow \infty} p^{\mathcal{M}}(\text{pre}(w, i))$  is well defined, and is defined naturally to be the acceptance probability  $p^{\mathcal{M}}(w)$  of  $w$ .

The language accepted by a PFA  $\mathcal{M}$  given a threshold  $\tau$ , denoted by  $L(\mathcal{M}, \tau)$ , is the set of all strings that take the PFA to the accepting state with probability greater than  $\tau$ :

$$L(\mathcal{M}, \tau) = \{w \in \Sigma^* : p^{\mathcal{M}}(w) > \tau\}.$$

An important question, which is surprisingly undecidable [99, 24], is whether, given a PFA  $\mathcal{M}$  and threshold  $\tau$ , the accepted language is empty. The undecidability of this *emptiness* problem is used in the next chapter in establishing the undecidability of many problems under UMDPs.

## 2.5 Non-Periodicity of Optimal Action Sequences

We can view an MDP model with discounting as an instance of models in which attainment of any nonzero reward terminates with probability one, or equivalently the system eventually transitions into a no-reward absorbing state. If  $\beta$  is the discount factor, then  $1 - \beta$  is the probability of termination at each stage. See, for example, the proof of Theorem 3.4.3 in chapter three or Chapter 5.3 in [103] for this way of modeling the discount factor. As such, discounted infinite-horizon models are closer to finite-horizon models than undiscounted models. Problems in finite-horizon models are computable because the set of action sequences under consideration is finite due to the finite bound on their length. Also, as we will see in the next chapter, undiscounted problems are not approximable in general, but approximation of the optimal value, to within any additive factor  $\epsilon > 0$ , is computable for POMDPs under the discounted criterion. This follows directly from the presence of the discounting factor. These facts point to the relative ease of solving problems in discounted UMDPs. Still, as we show below, there remain difficulties with solving discounted POMDPs optimally.

It is plausible that in an unobservable model, optimal sequences of actions, if not finite, always become repetitive ending in a regular structure. This can be verified, for example, in the case of deterministic unobservable models. A quote from Rodney Brooks on sensorless robots [39] expresses a similar intuition: “Without sensors, robots would be nothing more than fixed automation, going through the same repetitive task again and again in a carefully controlled environment.” However, while this repetitive property might hold in most cases in random environments too, we will next see that it does not hold all the time, even in the presence of discounting. First we define the notion of a periodic action sequence.

**Definition 1** *Given an alphabet  $\Sigma$ , a sequence  $w$  is periodic if  $w = uv^\infty$ , where  $u, v \in \Sigma^*$ .*

Note that finite sequences are always periodic as  $v$  can be an empty string in the definition.

Existence of optimal sequences that are periodic would be evidence for decidability of the decision problems because the size of the period and the action sequence in it could be guessed and then verified. While this property might be the case for two-state UMDPs (see the next section), we adapt a PFA example from Paz [99] to show that this is not true for

UMDPs with three states, even in the presence of discounting. Thus the lack of periodicity points to difficulties in computing optimal action sequences in discounted models.

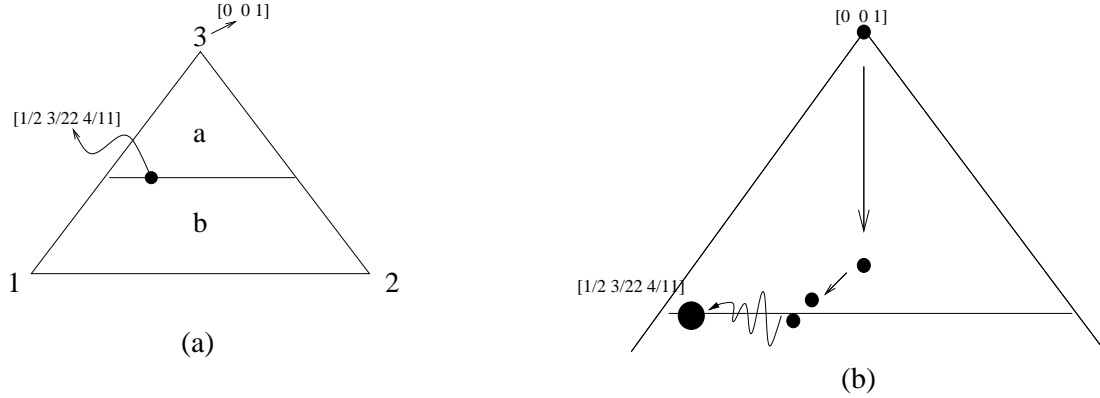


Figure 2.1: (a) The 3 dimensional unit simplex which represents possible distributions over 3 states, partitioned into two regions. Action  $a$  is optimal in the top region. (b) The evolution of a point toward the stationary point. The pattern of the points of the sequence being above or below the  $\frac{4}{11}$  line is irregular.

### 2.5.1 An Example of Non-Periodicity

Consider the following 3 state, 2 action UMDP. The two actions  $a$  and  $b$  have identical dynamics, denoted by the matrix  $\mathbf{M}$  below, but their reward vectors  $\mathbf{R}_a$  and  $\mathbf{R}_b$  are different:

$$\mathbf{M} = \begin{bmatrix} \frac{2}{3} & 0 & \frac{1}{3} \\ \frac{5}{9} & \frac{1}{3} & \frac{1}{9} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{bmatrix}, \quad \mathbf{R}_a = \begin{bmatrix} 0 & 0 & \frac{7}{4} \end{bmatrix}, \quad \mathbf{R}_b = [1 \ 1 \ 0] \quad (2.4)$$

The unique stationary point of the matrix is at  $[\frac{1}{2} \ \frac{3}{22} \ \frac{4}{11}]$  (*i.e.*,  $[\frac{1}{2} \ \frac{3}{22} \ \frac{4}{11}]\mathbf{M} = [\frac{1}{2} \ \frac{3}{22} \ \frac{4}{11}]$ ), and the matrix is ergodic, that is, all points of the simplex converge to the stationary point with repeated applications of the matrix:  $\forall \mathbf{x} \in \mathcal{X}, \lim_{i \rightarrow \infty} \mathbf{x} \mathbf{M}^i = [\frac{1}{2} \ \frac{3}{22} \ \frac{4}{11}]$ .

Due to identical dynamics, the action that yields the higher reward is the best (optimal) action at any given distribution. Hence, we note that the reward vectors are designed so that if the third component of a point is greater than  $\frac{4}{11}$  then action  $a$  is to be executed,

and if below  $\frac{4}{11}$  then action  $b$  is to be applied, while either action is optimal at  $\frac{4}{11}$ . Notice that this is also irrespective of the discount factor  $\beta$ .

Figure 2.1(a) shows the three dimensional unit simplex with regions of the simplex where actions  $a$  and  $b$  are optimal, and the stationary point, with a line drawn corresponding the points with third component equal to  $\frac{4}{11}$ . The line separates the regions of optimality for  $a$  and  $b$ . In order to show the non-periodicity of optimal action sequences, we need only show that convergence of the images of a point as optimal actions are executed, to the stationary point, visits the  $a$  and  $b$  regions, *i.e.* goes above and below the line, in an irregular manner (see Fig. 2.1(b)).

This non-regularity is shown by Paz [99]. He shows that the language accepted by the single letter, three state PFA  $\mathcal{M} = (\{s_1, s_2, s_3\}, \{a\}, s_1, s_3)$  with dynamics  $\mathbf{M}_a$  given as above, and given threshold  $\frac{4}{11}$ , is not regular and in fact not context-free as all single-letter context-free languages are regular. The argument, briefly, is to formulate the third component of  $[0 \ 0 \ 1]\mathbf{M}^k$ ,  $m_{3,3}^k$ , in terms of an eigenvalue  $\lambda = \frac{1}{4} + \frac{\sqrt{7}}{12}i$  of  $\mathbf{M}$  :  $m_{3,3}^k = \frac{4}{11} + u\lambda^k + \bar{u}\bar{\lambda}^k$ , where  $u = \frac{7}{22} + \frac{3\sqrt{7}}{154}i$  and  $\bar{u}$  and  $\bar{\lambda}$  denote the complex conjugates, and show that the difference  $m_{3,3}^k - \frac{4}{11}$  is such that for any two distinct powers  $k_1$  and  $k_2$ , where  $m_{3,3}^{k_1} - \frac{4}{11} > 0$  and  $m_{3,3}^{k_2} - \frac{4}{11} > 0$ , so that  $a^{k_1}$  and  $a^{k_2}$  are in the language, there is a  $k_3$ , such that  $m_{3,3}^{k_1+k_3} - \frac{4}{11} > 0$ , or  $a^{k_1+k_3}$  is in the language, while  $m_{3,3}^{k_2+k_3} - \frac{4}{11} < 0$ , or  $a^{k_2+k_3}$  is not. This violates Myhill-Nerode's characterization that regular languages define an equivalence relation with a finite number of classes on strings [57, 99].

From the non-regularity of this language, it immediately follows that the patterns of  $a$ 's and  $b$ 's do not form a periodic sequence in the optimal action sequence for initial distribution  $[0 \ 0 \ 1]$ .

This example brings up two noteworthy points and conjectures:

1. Small perturbations in, say, one of the reward vectors, changes the optimal regions so that the stationary point would fall inside one region. When the stationary point is inside a single region, it is not hard to see that at any point periodic optimal sequences exist. Thus it may be the case that non-periodic sequences may be very rare. Of course, in this example the two transition matrices were the same. It would

be interesting to show whether or not a conjecture such as the following holds: given a natural distribution over the inputs of the problems, with probability one, at every point an optimal periodic sequence exists.

2. The irrationality of optimal values points to further difficulties in computing optimal values and gives additional insight into sources of undecidability of the problem. We conjecture irrationality is the case for the example UMDP given above:

**Conjecture 2.5.1** *For some rational values of the discount factor  $\beta$ , the maximum expected total discounted reward obtained for the UMDP given by the parameters in 2.4 is irrational.*

## 2.6 Several Structural Properties of UMDPs and Open Problems

In this section, we explore a few additional properties of discounted UMDPs. The goal is to find the type of difficulties that exist in computing optimal action sequences and values. The previous section showed one source of difficulty, that of non-periodicity. Here we establish a few properties and conjecture on a few more with regards to representation of optimal action regions (see below) on the simplex of probability distributions. If these hold, they narrow down the difficulties of computing optimal action sequences, and otherwise provide further insight into the hardness behind the undecidability.

Consider the  $n$ -dimensional simplex  $\mathcal{X}$ . An action  $a$  is optimal at a point  $\mathbf{x}$  of the simplex if some optimal sequence of actions for point  $\mathbf{x}$  begins with  $a$ . An important property, which stems from the Markov property, is that in infinite-horizon problems, this optimality does not depend on the time: if an action  $a$  is optimal at  $\mathbf{x}$  it is optimal whenever the distribution over system state is  $\mathbf{x}$ . This allows for the representation of a policy to be a mapping from  $\mathcal{X}$  to a choice of an optimal action. In the presence of discounting, the containment  $a \in \arg \max_{\sigma \in \Sigma} [\mathbf{x}\mathbf{R}_\sigma + \beta\mathcal{V}^*(\mathbf{x}\mathbf{M}_\sigma)]$  implies  $a$  is optimal at  $\mathbf{x}$  (and vice versa), but when there is no discounting, the implication does not hold (see appendix).

Define the optimal region for action  $a$ , denoted by  $region^*(a)$ , to be all the points where action  $a$  is an optimal action. Closure of optimal regions is a desirable property, as it may

allow for computing optimal regions (optimal policies) by determining the boundary points where optimal actions tie. Closure is the case for finite-horizon optimal partitions.

**Lemma 2.6.1 (closedness)** *In the discounted criterion,  $\forall a \in \Sigma, region^*(a)$  is closed.*

**Proof.** The lemma follows from the continuity of  $\mathcal{V}^*$  (from Theorem 2.3.2), and the fact that if  $a \in \arg \max_{a \in \Sigma} [\mathbf{x}\mathbf{R}_a + \beta\mathcal{V}^*(\mathbf{x}\mathbf{M}_a)]$  then  $\mathbf{x} \in region^*(a)$  ( $a$  is optimal at  $\mathbf{x}$ ).

Let  $\mathbf{x}$  be such that there are points arbitrarily close to it at which action  $a$  is optimal. We show that action  $a$  must be optimal at point  $\mathbf{x}$  as well. Take a sequence  $\{\mathbf{x}^{(k)}\}$  of points converging to  $\mathbf{x}$  where

$$\forall \mathbf{x}^{(k)}, \forall b \in \Sigma, \mathbf{x}^{(k)}\mathbf{R}_a + \beta\mathcal{V}^*(\mathbf{x}^{(k)}\mathbf{M}_a) \geq \mathbf{x}^{(k)}\mathbf{R}_b + \beta\mathcal{V}^*(\mathbf{x}^{(k)}\mathbf{M}_b).$$

From continuity of  $\mathcal{V}^*$ ,  $\lim_{k \rightarrow \infty} \mathbf{x}^{(k)}\mathbf{R}_a + \beta\mathcal{V}^*(\mathbf{x}^{(k)}\mathbf{M}_a) = \mathbf{x}\mathbf{R}_a + \beta\mathcal{V}^*(\mathbf{x}\mathbf{M}_a)$ , which shows that action  $a$  is at least as good as any other action at  $\mathbf{x}$ :

$$\forall b \in \Sigma, \mathbf{x}\mathbf{R}_a + \beta\mathcal{V}^*(\mathbf{x}\mathbf{M}_a) \geq \mathbf{x}\mathbf{R}_b + \beta\mathcal{V}^*(\mathbf{x}\mathbf{M}_b),$$

hence action  $a$  is also optimal at  $\mathbf{x}$ . □

Fig. 2.2 shows that optimal regions in the absence of discounting may not be closed. The example also shows that optimal sequences in an undiscounted (goal-oriented) models may not be periodic in the sense of Definition 1, as there is no optimal sequence of actions at state 1: for any  $k \geq 1$ ,  $a^k b$  has a higher value than  $a^{k-1} b$ , but  $a^\infty$  has value 0.

A maximal connected subset of  $region^*(a)$  is a subset of  $region^*(a)$  that is connected and is not a proper subset of any other connected subset of  $region^*(a)$ . Important open problems on the structural properties are:

**Open Problem 2.6.2** *Is it the case that  $\forall a \in \Sigma, region^*(a)$  contains finitely many maximally connected subregions? If so, do these regions have piecewise linear boundaries?*

A piecewise linear boundary is defined by a finite number of linear functions. Such properties hold for finite-horizon POMDP problems [113]. It is not clear that in the limit to the infinite horizon, the properties still hold. The optimal value function has a finite number of pieces for the finite-horizon problem, but examples can be constructed where it has an infinite number of pieces in the infinite-horizon problem.

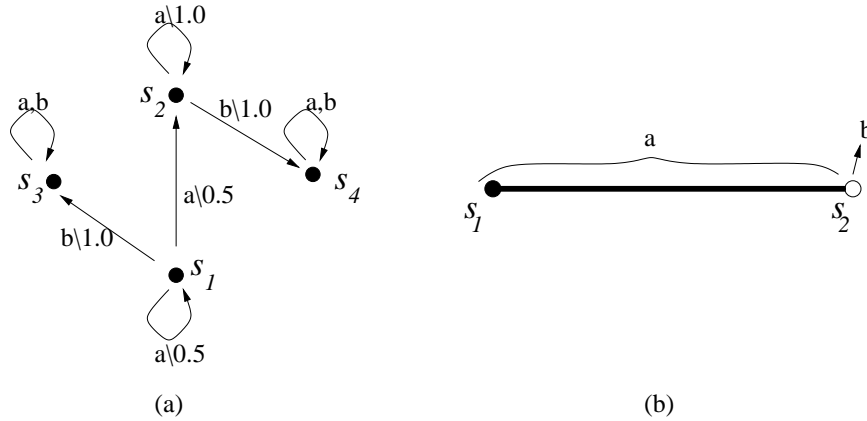


Figure 2.2: (a) A simple instance of an undiscounted objective where closedness of optimal regions does not hold. The numbers on the edges are the probabilities of the transitions. The only reward is from executing action  $b$  in state 2, other rewards being zero. Thus the model can be viewed as a goal-oriented model (Section 2.3.3), where  $s_4$  is the goal state. (b) The optimal policy shown as a mapping from probability of being in state 1 versus state 2 to choice of action. In any distribution where state 1 has nonzero probability, action  $a$  is optimal.

### 2.6.1 Two-state UMDPs

We next show evidence that in two-state UMDPs, if the answer to the open problem 2.6.2 on the finiteness of the regions is positive, optimal sequences are periodic. More specifically, we show that under conditions of a finite number of regions and a finite number of cluster points of the set of the images of a point<sup>1</sup> (distribution over system states), optimal action sequences become periodic in two-state UMDPs. We saw in the previous section that under the same conditions, periodicity failed for a three-state UMDP. A few basic definitions and the argument follows.

The two-dimensional simplex may be mapped to the interval  $\mathcal{X} = [0, 1]$ , in which case a point  $x \in [0, 1]$  represents the probability of being in one of the two states, say state  $s_1$ . The stochastic transition matrices corresponding to actions become linear functions from  $[0, 1]$  to  $[0, 1]$  (see Figure 2.3(a)). The positive answer to open problem 2.6.2 then states

---

<sup>1</sup>Images of a point are obtained by executing an optimal action at each subsequent resulting distribution point.

that for each action  $a$ ,  $region^*(a)$  is composed of a finite number of subintervals of  $\mathcal{X}$ , and an optimal policy that partitions  $\mathcal{X}$  into a finite number of intervals—each interval mapped to a single action—always exists.

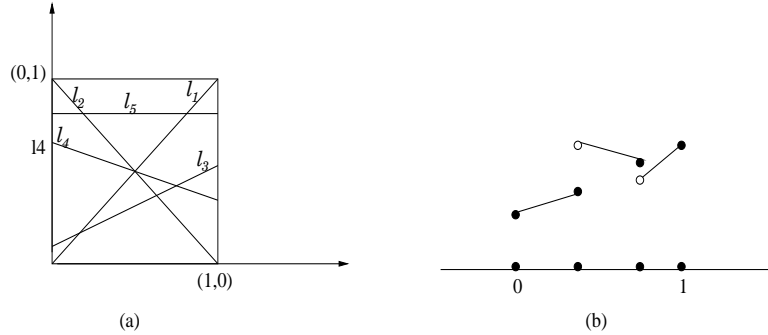


Figure 2.3: (a) Several linear functions in  $\mathcal{L}$  from  $[0, 1]$  to  $[0, 1]$ . Functions  $l_1$  and  $l_2$  correspond to the identity and permutation matrices respectively. (b) A function  $f \in \mathcal{F}$  consisting of three intervals and two border points.

Let  $\mathcal{L}$  denote the set of linear functions mapping  $[0, 1]$  to  $[0, 1]$ . Let  $\mathcal{F}$  denote the set of functions  $f : \mathcal{X} \rightarrow \mathcal{X}$ , such that over a finite partitioning of  $\mathcal{X}$  into subintervals  $I_i$ ,  $f$  is a linear function  $f_i \in \mathcal{L}$ . We call each  $f_i$  a segment of  $f$ , each interval, where  $f$  takes the same function definition, an interval of  $f$ , and end points of intervals, where  $f \in \mathcal{F}$  changes function definition, partition points. Thus each  $f \in \mathcal{F}$  has a finite number of partition points. If a policy partitions  $\mathcal{X}$  into a finite number of intervals, such a policy corresponds to a function  $f \in \mathcal{F}$ . See Figure 2.3(b).

Define the *orbit* of a point  $x$ , denoted by  $O(x)$ , to be the set of *images* of  $x$  under  $f$ , thus  $O(x) = \{f^k(x), k \geq 0\}$ , where  $f^0(x) = x$  and  $f^k(x) = f(f^{k-1}(x))$ . A point  $y$  is a cluster point of a set  $S$ , if  $y \notin S$ , and  $\forall \epsilon > 0, \exists x \in S$ , such that  $|y - x| < \epsilon$ .

The following lemma shows the difference between the one dimensional (two-state) and the two-dimensional (three-state) cases. It says that, in the one dimensional case, images of a point eventually visit the same intervals of  $f$  in a periodic manner, if its orbit has a finite number of cluster points. In the two-dimensional (three-dimensional unit simplex) example of the last section, we saw this was not the case even though there was a single cluster point

(the stationary point).

**Lemma 2.6.3** *Consider any function  $f \in \mathcal{F}$ . Then for any  $x \in [0, 1]$  such that  $O(x)$  has a finite number of cluster points,  $f^k(x)$  eventually visits some subset of the intervals of  $f$  in a periodic manner, i.e.  $\exists N \exists p, \forall k \geq N, f^k(x) \in I \Leftrightarrow f^{k+p}(x) \in I$ , where  $I$  is a subinterval of the partition.*

**Proof.** If the set of the cluster points of  $O(x)$  is empty, or  $O(x)$  is finite, then the images of  $x$  eventually become cyclic:  $\exists k, p, f^{k+p}(x) = f^k(x)$ , from which periodicity follows.

Otherwise, let us assume there are  $j \geq 1$  cluster points of  $O(x)$ . Let  $\epsilon > 0$  be less than the minimum distance between two cluster points or a cluster point and the closest partition point of  $f$ , other than the cluster point itself if the cluster point is a partition point. Take any cluster point  $p$ , and assume  $I = (p - \epsilon, p)$  is the interval containing points of  $O(x)$  arbitrarily close to  $p$ . Consider  $f(I)$ , defined from extension of  $f$  to sets of points:  $f(I) = \{f(x) | x \in I\}$ .  $f(I)$  is an interval, as  $I$  contained no partition point of  $f$ .  $f(I)$  has length no greater than length of  $I$ , as each segment of  $f$  belongs to  $\mathcal{L}$ . Note that an end point of  $f(I)$ , the point corresponding to the image  $f(p)$  (or if  $f$  has a different definition on  $I$ , say  $l$  and on  $p$ , then  $l(p)$ ) is a cluster point of  $O(x)$ . Consequently,  $f(I)$  cannot contain any partition point or cluster point of  $O(x)$  itself as this would contradict the assumption that  $\epsilon$  was the minimum distance. Finally,  $f(I)$  has nonzero length, i.e.  $f(I)$  may not be a single point, otherwise  $x$  would be cyclic. Thus  $f(I)$  has the same properties as  $I$ , and the same applies to  $f(f(I))$  and so on. Thus in no more than  $2j$  applications of  $f$ , we are back at the same side of some cluster point (same side of  $p$  must be repeated first), as each cluster point has at most two sides. This new interval is a potentially smaller interval, as each segment of  $f$  may have slope less than 1. If we repeat applying  $f$  again, the same regions are repeated.  $\square$

We close this section with a conjecture on the computability of the two-state problem. We conjecture that under the discounted criterion, there always exist optimal policies that partition the  $[0, 1]$  interval into a finite number of intervals, and given such, the orbit of any point has a finite number of cluster points, or, by Lemma 2.6.3, optimal actions sequences are periodic. Once periodicity is established, we expect that in finite time, given any initial

point, a periodic optimal action sequence can be found. Thus,

**Conjecture 2.6.4** *The two-state infinite-horizon discounted UMDP is computable.*

In order to prove that all orbits have a finite number of cluster points, it suffices to show the following statement, which we expect holds<sup>2</sup>: there is a bound  $K$ , such that any point whose orbit includes a partition point, maps to the partition point within a number of applications of  $f$  that is less than  $K$ . The proof that the statement is sufficient is similar to the proof of Lemma 2.6.3 and we do not include it here. We expect that proving open problem 2.6.2, at least for the two-state case, is the key to establishing Conjecture 2.6.4.

## 2.7 Discussion

In the next chapter, we establish undecidability of many UMDP problems. We note that the undecidability is with respect to increasing state set size  $n$ . With  $n$  fixed, the number of bits in the representation of the numbers is still a source of an unbounded number of problems. In the last section we conjectured that the two-state UMDP is decidable. From properties such as non-periodicity it appears that the UMDP problem, for some fixed  $n$ , may also be undecidable, though proofs of such should be difficult, and would involve encoding undecidable problems in the input numbers. Settling open problems such as 2.6.2 should further shed insight into difficulties of solving UMDPs.

The periodicity property discussed in 2.5 corresponds to the existence of optimal *finite controllers* for POMDPs. Briefly, a finite controller for a POMDP is a finite state machine which prescribes an action to execute at each of its own states, and when an action is executed and an observation observed, it changes state based on the observation, its current state, and the action executed. A finite controller is an elegant way of controlling a POMDP, and algorithms generating finite controllers show promise in fast finding optimal or near optimal solutions in many POMDP problems [49]. In case of UMDPs, because the output of a finite controller is just a periodic action sequence, a UMDP problem with a given initial

---

<sup>2</sup>This property is also explored within the context of (extended) *finitely transient* policies by Cassandra [18]. Extended finitely transient policies are desirable as they always define piecewise linear value functions [114, 115].

distribution has an optimal finite controller if and only if an optimal action sequence exists for it. Thus, the non-periodicity of action sequences under the three-state UMDP shows that there are UMDPs that do not have optimal finite controllers. It is interesting to note that in case of POMDPs, it is easier to construct counterexamples: for a two-state 2-armed bandit problem, it is easy to see that its optimal control does not yield a finite controller. On the other hand, as discussed in the previous section, we expect that in the unobservable two-state case, periodicity, or existence of optimal finite controllers, holds. This raises the possibility that the two-state UMDP is decidable, while the two-state POMDP may not be. As established in the next chapter, we find that unfortunately it is the case that the question of whether an optimal finite controller exists is undecidable in general.

## Chapter 3

## UNDECIDABILITY IN POMDPS AND PROBABILISTIC PLANNING

We show in this chapter that many natural decision problems in infinite-horizon POMDP and probabilistic planning problem domains are undecidable. For this purpose, we define the emptiness problem for probabilistic finite automata and describe in some detail the undecidability proof presented in [24]. We then show how the reduction can be used and modified in establishing various undecidability results for optimal computations and in many cases even approximations.

**3.1 Introduction**

In this chapter we show that several basic problems, at the heart of dynamic decision making and planning under uncertainty are extremely hard; in fact they are undecidable. The undecidability had been a matter of conjecture [98, 76], and the basic question remained whether there was some simplifying structure—at least in some of the easier problems such as the discounted POMDPs—that rendered them solvable in finite time. If there were such a simplifying structure, it could potentially be used to make algorithms and heuristics for the problems more efficient. We saw in Chapter 2 that even discounted UMDPs lack simplifying properties that could make them decidable. This chapter settles the questions and shows that even in the case of discounting, many basic problems are undecidable. In addition, we establish inapproximability results, under a few undiscounted optimality criteria, and prove the undecidability of several related problems. An interesting consequence of our results on the impossibility of finding approximately optimal plans is that if the length of a candidate solution plan is bounded in size—even by an exponential function of the input description length—the solution found can be arbitrarily suboptimal.

We show undecidability by making use of the undecidability of the emptiness problem for probabilistic finite automata, shown by Paz [99] and Condon and Lipton [24]. The

undecidability of the emptiness problem is in itself somewhat surprising, as the corresponding problems for nondeterministic and deterministic finite automata are simple reachability computations. Such hardness results point to an added power randomized computations bring to space bounded machines such as finite automata. As we describe in this chapter and was first developed by Freivalds [42], unlike deterministic automata, probabilistic finite automata have a certain weak ability to count that lies at the core of all the reductions we use.

We use the proof in [24] instead of [99], as it was more useful to us in establishing other undecidability results, and in particular for showing the undecidability of a problem left open in [99]. In section 3.2 we define the string-existence or the emptiness problem for probabilistic finite automata, and describe the reduction in [24] that proves the problem is undecidable. We make extensive use of the details of the reduction in some of our proofs, so we describe the reduction in detail in 3.2.2. The rest of the chapter consists of showing the undecidability of related problems. In section 3.2.3 we describe an inapproximability result, which will be used in showing inapproximability for several optimization problems, and in section 3.2.4 we show the undecidability of the threshold isolation problem [105]. We then proceed to showing undecidability in probabilistic planning and POMDP (specifically UMDP) problems in sections 3.3 and 3.4, and close with a summary.

### 3.2 Undecidability of the Emptiness Problem

The (language) *emptiness* problem for PFAs is as follows:

**Problem 3.2.1** *Given a PFA and a desired threshold  $\tau$ , decide whether or not there is some input string  $w \in \Sigma^*$  that the PFA accepts with probability exceeding threshold  $\tau$ .*

Both Paz [99] and Lipton and Condon [24] establish the undecidability of the problem:

**Theorem 3.2.2** [99][24] *The emptiness problem for PFAs is undecidable.*

Below, we describe the properties of the reduction in [24] from a high level, and then give a more detailed explanation of the proof. The details of the proof are also used to develop subsequent undecidability results on probabilistic planning and POMDP problems, most

notably Lemma 3.4.3, which establishes the undecidability of optimal policy construction for discounted-total-reward infinite-horizon POMDPs.

### 3.2.1 Properties of the Reduction

In [24], the (undecidable) question of whether a Turing machine (TM) accepts the empty string is reduced to the question of whether a PFA accepts any string with probability exceeding a threshold. The PFA constructed by the reduction tests whether its input is a concatenation of *accepting sequences*. An accepting sequence is a legal sequence of TM configurations beginning at the initial configuration and terminating in an accepting configuration.

The reduction has the property that if the TM is accepting, *i.e.* it accepts the empty string, then the PFA accepts sufficiently long concatenations of accepting sequences with high probability. But if the TM is *not* accepting, the PFA accepts all strings with low probability. We next formalize these properties and use them in subsequent undecidability results. The following section explains how the PFA generated by the reduction has these properties.

**Theorem 3.2.3** *There exists an algorithm which, given a two counter TM as input and any rational  $\epsilon > 0$  and integer  $K \geq 1$ , outputs a PFA  $M$  satisfying the following:*

1. *If the TM does not accept the empty string, the PFA  $M$  accepts no string with probability exceeding  $\epsilon$ .*
2. *If the TM is accepting, then let string  $w$  represent the accepting sequence. We have  $\lim_{i \rightarrow \infty} p^M(w^i) = 1 - (1/2)^K$ , and  $\forall i, p^M(w^i) < 1 - (1/2)^K$ .*

We conclude this section making two additional points about the emptiness problem.

- Due to the separation between the acceptance probability of the PFA in the two cases of the TM accepting the empty string or otherwise, the emptiness problem remains undecidable if the strict inequality in the description of the existence problem is replaced by a weak equality ( $\geq$ ) relation.

- Although the problem is posed in terms of the existence of finite strings, the result holds even if the strings have infinite length.

### 3.2.2 Details of the Reduction

The class of TMs used in the reduction in [24] are *two-counter* TMs, which are as powerful as general TMs. The constructed PFA is supposed to detect whether a sequence of computations represents a valid accepting computation (accepting sequence) of the TM. This task reduces to the problem of checking the legality of each transition from one configuration of the TM to the next, which amounts to verifying that

- the first configuration has the machine in the start state
- the last configuration has the machine in the accepting state
- each transition is legal according to the TM's transition rules.

All these checks can be carried out by a deterministic finite state automaton, except the check as to whether the TM's counter contents remain valid across consecutive configurations. The PFA rejects immediately if any of the easily verifiable transition rules are violated, which leaves only the problem of validating the counters' contents across each transition.

On each computation step taken by a two-counter TM the counters' contents either stay the same, get incremented by 1, or get decremented by 1. Assuming without loss of generality that the counter contents are represented in unary, this problem reduces to checking whether two strings have the same length: given a string  $a^i b^j$ , does  $i = j$ ?

Although this question cannot be answered exactly by any PFA, a *weak equality* test developed in [24] and inspired by the work of Freivalds [42] can answer it in a strict and limited sense which is nonetheless sufficient to allow the reduction. The weak equality test, shown in Fig. 3.1, works as follows. The PFA scans its input string  $a^i b^j$ , and with high probability enters an *Indecision* state (equivalently we say the outcome of the test is Indecision). In the figure, the event of Indecision is given a thick edge to denote its high

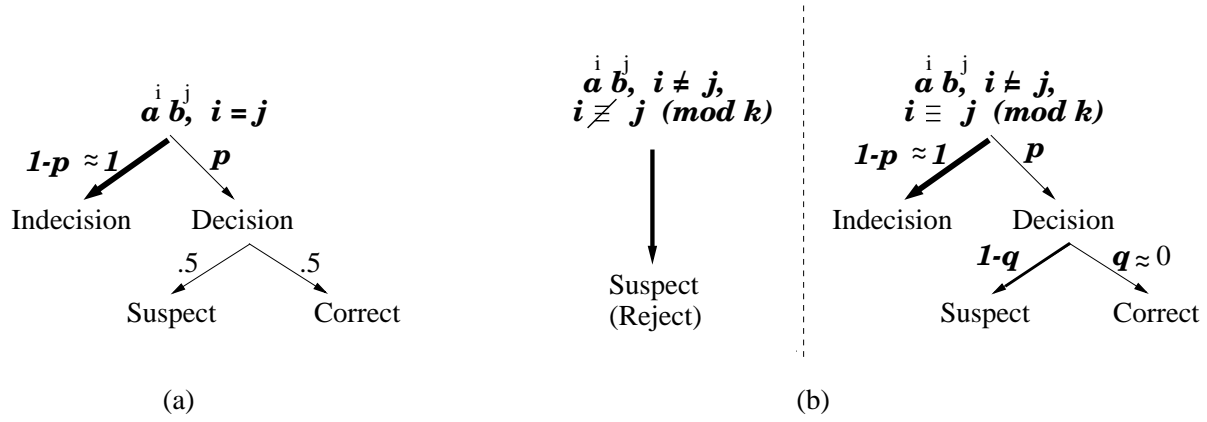


Figure 3.1: Outcomes of the weak equality test when  $a^i b^j$  is read, (a) when  $i = j$ , (b) when  $i \neq j$ . While the probability of making a decision is minute, if a decision is made, the chances of Suspect and Correct is the same when  $i = j$ , but Suspect is much more likely when  $i \neq j$ .

probability. With some low probability the PFA enters one of two “decisive” states. If the substrings have equal length (Fig. 3.1a) the PFA either enters a *Correct* state or a *Suspect* state. It enters these two states equiprobably. However, suppose that the PFA enters a decisive state but the input string is composed of *unequal*-length substrings ( $i \neq j$ , Fig. 3.1b). In this case the *Suspect* outcome is at least  $2^k$  (for  $k \geq 2$ ) times more likely than the *Correct* outcome, where the *discrimination factor*  $k$  can be made as large as desired by increasing the size of the PFA. This test is described in more detail in Appendix B.

The PFA of the reduction carries out a *global* test of its own, not unlike the weak equality test (Fig. 3.2), on a candidate accepting sequence for the TM, and uses the weak equality test to check for counter increments or decrements on consecutive configurations. Given a candidate accepting sequence, if the outcome of *all* the tests are decisive and *correct*, the PFA accepts the input. If the outcome of *all* the tests are *suspect*, the PFA rejects the input. The probability of an all decisive test is, of course, very small, but positive. The PFA remains in the *global-indecision* state until it detects the start of the next candidate accepting sequence (start configuration of the TM), or until it reaches the end of the input. It follows from the outcome of the individual tests that, if the PFA is given an illegal



sequence, the PFA can instead increment an *all-decisive* counter which has a finite upper limit  $K$ . The PFA accepts its input if and only if the all-decisive counter reaches  $K$ , and it has seen an all *correct* on a candidate sequence. Hence, if the TM is accepting, the PFA accepts concatenation of sufficiently many accepting sequences with probability arbitrarily close to  $1 - (1/2)^K$ . In addition, for the cases when the TM is not accepting, the acceptance probability of the PFA can be made as small as desired for a given counter upper limit  $K$ , by choosing the discrimination factor  $k$  of the weak equality test to be large.

### 3.2.3 Undecidability of Approximations

The question of approximability is an important one, especially when computing an optimal answer is impossible. Unfortunately, it follows from the next corollary that approximations, such as computing a string which the PFA accepts with probability within an additive constant or multiplicative factor  $\epsilon < 1$  of the maximum acceptance probability of the PFA<sup>1</sup> are also uncomputable.

**Corollary 3.2.4** *For any fixed  $\epsilon, 0 < \epsilon < 1$ , the following problem is undecidable: Given is a PFA  $M$  for which one of the two cases hold:*

- *The PFA accepts some string with probability greater than  $1 - \epsilon$ .*
- *The PFA accepts no string with probability greater than  $\epsilon$ .*

*Decide whether case 1 holds.*

**Proof.** The corollary is an immediate consequence of the properties outlined in Theorem 3.2.3, and the fact that  $\epsilon$  in the reduction can be made as small as desired.  $\square$

### 3.2.4 Undecidability of the Threshold-Isolation Problem

There might be some hope for decidability of the emptiness problem for special cases, for example for problems for which the given threshold is *isolated* for the given PFA:

---

<sup>1</sup>The maximum acceptance probability is taken as the supremum over the acceptance probability over all strings.

**Definition 2** [105] *Let  $M$  be a PFA. The threshold  $\tau$  is  $\epsilon$ -isolated with respect to  $M$  if  $|p^M(x) - \tau| \geq \epsilon$  for all  $x \in \Sigma^*$ , for some  $\epsilon > 0$ .*

**Definition 3** *The threshold-isolation problem<sup>2</sup> is, given a PFA  $M$  and a threshold  $\tau$ , decide whether, for some  $\epsilon > 0$ , the threshold  $\tau$  is  $\epsilon$ -isolated for the PFA  $M$ .*

Isolated thresholds are interesting because PFAs with isolated thresholds have less expressive power than general PFAs, thus the corresponding decision problems are easier. General PFAs are powerful enough to accept even non-context-free languages (see Chapter 2 for an example). However, Rabin in [105] showed that PFA with isolated thresholds accept regular languages. A natural question then is: given a PFA and a threshold, whether the threshold is isolated for the PFA. If we can compute the answer and it is positive, then we can presumably compute the regular language accepted by the PFA, and see whether it is empty or not. That would afford at least the opportunity to recognize and solve a special case of the general emptiness problem.

The decidability of the isolation problem was raised by Paz [99], and was heretofore an open question to the best of our knowledge. The reduction in [24] shows that recognizing an isolated threshold is hard as well:

**Corollary 3.2.5** *The threshold-isolation problem is undecidable.*

**Proof.** As stated in Theorem 3.2.3, we can design the reduction with  $\epsilon = 1/3$ , and  $K = 1$ . It follows that if the TM is not accepting, then there is no string that the PFA accepts with probability greater than  $1/3$ , while if the TM is accepting, there are (finite) strings that the PFA accepts with probability arbitrarily close to  $1/2$ . In other words, the threshold  $1/2$  is isolated iff the TM is not accepting.  $\square$

### 3.3 Undecidable Problems in Probabilistic Planning

Our work on computability of POMDPs was originally motivated by questions about the computability of probabilistic planning problems, such as the problems introduced in [74,

---

<sup>2</sup>In the PFA literature, such as [99] and [105], a threshold is referred to as a cutpoint, and the problem referred as cutpoint-isolation.

12].

The probabilistic planning problem, studied by Kushmerick, Hanks, and Weld [74], for example, involves a finite set of states, a finite set of actions effecting stochastic state transitions, a start state (or probability distribution over states), a goal region of the state space, and a threshold  $\tau$  on the probability of plan success. The problem is to find *any* sequence of actions that would move the system from the start state to a goal state with probability at least  $\tau$ .

While it had been well established that restricted versions of this problem were decidable, though intractable as a practical matter [98, 14, 78], the complexity of the general probabilistic planning problem (i.e. without restrictions on the nature of the transitions or the length of solution plan considered) had not been determined.

The results of the previous section establish the uncomputability of such problems in the general case—when there is no restriction imposed on the length of solution plans considered. Uncomputability follows when it is established that a sufficiently powerful probabilistic planning language can model any given PFA, so that any question about a PFA can be reformulated as a probabilistic planning problem.

This is the case for the probabilistic planning model investigated in Kushmerick, Hanks, and Weld [74]. This model is based on STRIPS propositional planning [40] with uncertainty in the form of (conditional) probability distributions added to the action effects. It is established by Boutilier, Dean, and Hanks [12] that the propositional encoding of states is sufficient to represent any finite state space, and the extended probabilistic STRIPS action representation is sufficient to represent any stochastic transition matrix. Thus the emptiness problem (“is there any input string that moves the automaton from the start state to an accepting state with probability at least  $\tau$ ?”) can be directly reformulated in the planning context (“is there any sequence of actions that moves the system from a start state to a goal state with probability at least  $\tau$ ?”). An algorithm that solved the planning problem would answer the question of whether or not such a plan exists by either generating the plan or terminating having failed to do so, thus solving the equivalent emptiness problem. Therefore, as a corollary of the undecidability of the emptiness problem for PFAs we obtain:

**Theorem 3.3.1** *The plan-existence problem is undecidable.*

We also note that due to the tight correspondence between PFAs and probabilistic planning problems, the other undecidability results from the previous section apply as well:

- “Approximately satisficing planning,” generating a plan that is within some additive or multiplicative factor of the threshold is undecidable.
- Deciding whether the threshold for a particular planning problem represents an isolated threshold for that problem is undecidable.

### 3.4 Undecidable Problems for POMDPs

We first define the *policy-existence* (action sequence existence) problem for POMDPs, under any given optimality criterion. The space of policies considered in the following definitions is an important consideration. All of the lemmas hold when the space of policies includes any one or more of the following sets: finite action sequences of indefinite length, infinite sequences, or algorithms that create such finite or infinite sequences (using some kind of policy representation).

**Definition 4** *The policy-existence problem (with respect to an optimality criterion) is: given a POMDP and a threshold, does there exist a policy with expected value greater than the threshold?*

#### 3.4.1 Undecidability for Positive-Bounded Models under Total Undiscounted Reward

The most direct result involves a special case of infinite-horizon undiscounted total-reward models called the *positive bounded* models [103]. The essential feature of this model is that the reward structure and system dynamics for a problem must ensure that the total reward gathered is bounded both above and below, and is convergent (over action sequences) over an infinite horizon.

The PFA emptiness problem or the planning problem can easily be posed as a positive-bounded POMDP:

- the same observation  $o$  is received regardless of the state and action (non-observability)
- unit reward is gathered on the execution of *any* action on the goal state (Fig. 3.3b)
- the execution of *any* action at the goal state leads to an absorbing state: the system stays in that state and gathers no additional rewards (Fig. 3.3b)
- all other states and actions incur no reward.

From this equivalence we can immediately establish the following lemma:

**Theorem 3.4.1** *The policy-existence problem for positive-bounded problems under the infinite-horizon total reward criterion is undecidable.*

**Proof.** We describe a reduction from the plan-existence problem. Since any planning problem can be posed as a positive-bounded POMDP, we can easily verify that an effective algorithm for that problem could be used to solve the plan-existence problem, and by Corollary 3.3.1 such an algorithm cannot exist. To see this, note that a plan, say a finite sequence of actions, exists for the planning problem with probability of reaching the goal (success probability) exceeding  $\tau$  if and only if a finite sequence of actions exist with value exceeding  $\tau$  for the corresponding UMDP model (as outlined above and in Fig. 3.3b): Let  $p$  denote the success probability of a finite sequence of actions  $w$  in the planning problem. Then  $p$  is the expected total reward of action sequence  $wa$  in the corresponding UMDP model. Conversely, if  $v$  is the value of a sequence  $w$  in the UMDP model, then  $v$  is the success probability of sequence  $w$  in the planning problem.

A similar equivalence holds for infinite action sequences. □

### 3.4.2 Undecidability under the Average Reward Criterion

The indirect connection to PFAs allows extension of the previous result to *all* undiscounted total-reward models, and to average-reward models as well.

**Theorem 3.4.2** *The policy-existence problem under the infinite-horizon average reward criterion is undecidable.*

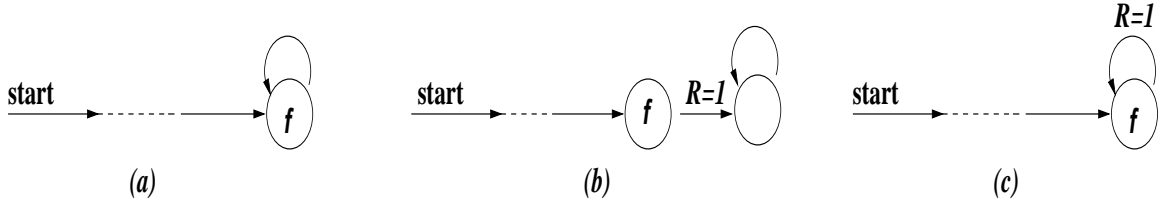


Figure 3.3: (a) A PFA, or the criterion of maximizing the probability of reaching a goal state ( $f$ ) in a probabilistic planning model. (b) The PFA emptiness problem modeled as a total reward criterion: The old accepting (goal) state  $f$ , on any action, now has a transition to an extra absorbing state giving reward of 1.0. All other rewards are zero. (c) Similarly, a PFA emptiness problem modeled as an average reward problem

**Proof.** The proof is complete once we observe that questions on acceptance probability of strings for a given PFA can readily be turned to questions on the value of similar strings in a related UMDP model. This transformation is achieved by modeling the probability of reaching the accepting state  $f$  using rewards (Fig. 3.3c). It can be verified that there is a string accepted by the PFA  $M$  with probability exceeding  $\tau$  if and only if there is a string with average reward greater than  $\tau$  for the corresponding UMDP model. To see this, assume for some string  $w$ ,  $p^M(w) > \tau$ , and denote by  $\mathcal{V}(w)$  the average reward of  $w$  under the corresponding UMDP model. We must have, for any action  $a$ ,  $p^M(w) \frac{k+1}{k+|w|} \leq \mathcal{V}(wa^k)$ . The inequality follows from writing  $\mathcal{V}(w)$  in terms of the probability of reaching the goal state on each prefix of  $w$ . We thus have  $\lim_{k \rightarrow \infty} \mathcal{V}(wa^k) \geq p^M(w)$ . Hence for some  $k$ ,  $\mathcal{V}(wa^k) > \tau$ . Conversely, we can verify that for any string  $w$ ,  $\mathcal{V}(w) \leq p^M(w)$ , so if  $\mathcal{V}(w) > \tau$ ,  $p^M(w) > \tau$ .

A similar equivalence holds for infinite strings. □

### 3.4.3 Undecidability of the Discounted Criterion

We turn now to the most commonly studied model: maximizing total expected discounted reward over an infinite horizon. Here, as in the proof of Lemma 3.4.2, we make a small change in the PFA constructed in the emptiness reduction.

**Theorem 3.4.3** *The policy-existence problem under the infinite-horizon discounted criterion is undecidable.*

**Proof.** Let us take the PFA constructed in the reduction of Section 3.2.2 and change it to a *leaky* PFA as follows. Let  $d$  be any rational value such that  $0 < d < 1$ . The leaky PFA, upon reading an input symbol, continues as the original PFA with probability  $1 - d$ . Otherwise, we say that it *leaks*, and in this case it makes a transition to either an absorbing rejection state or the absorbing accepting state, each with equal overall probability  $d/2$ . It is not hard to verify that maximizing the probability of reaching the accepting state in such a leaky PFA corresponds to maximizing the expected total discounted reward in a UMDP with a reward structure as described in the proof of Lemma 3.4.2 and Fig. 3.3a, where the discount factor is  $\beta = 1 - \frac{d}{2}$ . We show that if the TM is accepting (see Section 3.2.1), then the leaky PFA accepts some strings with probability greater than  $1/2$ , while if the TM is not accepting, every finite string is accepted with probability less than  $1/2$ .

Assume the TM is accepting, and let  $w$  be an accepting sequence of the TM. We assume the original PFA constructed in the reduction accepts only after  $K \geq 2$  decisive outcomes (the all-decisive counter limit  $K$  is explained in Section 3.2.2).

Let  $q_j$  denote the probability that the PFA "halts" (*i.e.*, goes into one of the absorbing states) on reading  $w^j$ . Let  $p_j$  denote the probability that the PFA has leaked *given* that it halts, *i.e.*, it halts due to the leak and not due to the remaining possibility of having  $K$  decisive outcomes (which has  $1 - p$  probability). Hence, given that the PFA halts on  $w^j$ , the probability of acceptance is  $p_j/2 + (1 - p_j)(1 - 1/2^K)$ , where the first term corresponds to the probability that it leaks and accepts (given that it halts) and the second is the probability that it doesn't leak and accepts. Thus the overall probability of acceptance is

$$p^M(w^j) = q_j[p_j/2 + (1 - p_j)(1 - 1/2^K)].$$

Now, for some  $\epsilon > 0$ ,  $(1 - 1/2^K) = 1/2 + \epsilon$ , since  $K \geq 2$ , thus

$$p_j/2 + (1 - p_j)(1 - 1/2^K) = 1/2 + (1 - p_j)(1/2 + \epsilon) = 1/2 + (1 - p_j)\epsilon.$$

Therefore we may argue that, as the probability  $q_j$  of the leaky PFA halting approaches 1 with increasing  $j$  of  $w^j$ ,  $1 - p_j$  is bounded from below by a constant greater than 0, to show that the acceptance probability exceeds  $1/2$  for some  $j$ . Let  $E_j$  be the event that the PFA halts but does not leak on  $w^j$ , so that it has made  $K$  decisions, and let  $Pr(E_j)$  denote

its probability. Thus  $p_j = 1 - Pr(E_j)$ . We note that when  $j < K$ ,  $Pr(E_j) = 0$ , ( $p_j = 1$ ) but  $Pr(E_K) > 0$ , as there is some probability that the PFA makes  $K$  decisions on  $w^K$ . We now argue that  $Pr(E_j) < Pr(E_{j+1})$  for  $j \geq K$ . Given that the PFA does not halt on  $w^j$ ,  $j \geq K - 1$ , the probability that it leaks on  $w^{j+1}$  remains a constant  $1 - (1 - d)^{|w|}$ , while the probability that it makes the  $K$ th decision increases somewhat because the probability that it has made  $K - 1$  decisions on reading  $w^j$  is increasing with  $j$ . Thus given that it halts on  $w^{j+1}$  the portion of probability mass goes slightly in favor of event  $E_{j+1}$ . Thus  $1 - p_j = Pr(E_j) \geq Pr(E_K) > 0, \forall j \geq K$ .

Assume the TM is not accepting. A candidate (accepting) sequence refers to a sequence of TM configurations where the first one is the initial TM configuration and the last is an accepting configuration. Any input string  $s$  can be viewed as a concatenation of  $j \geq 0$  candidate sequences appended with a possibly empty string  $u$  where none of the prefixes of  $u$  is a candidate sequence:  $s = w_1 w_2 \cdots w_j u, j \geq 0$ . If  $j < K$ , then probability of acceptance of the leaky PFA is  $qp/2$ , where  $q < 1$  is the probability of halting on  $s$  and  $p$  is the probability of leaking given that the PFA halts. Here,  $p = 1$  because there is no other possibility for halting, but  $q < 1$ . If  $j \geq K$ , then probability of acceptance is  $q(p/2 + (1-p)(1/2 - \epsilon))$ , where  $q < 1$  is the probability of halting on  $s$  and  $p < 1$  is the probability of leaking given that the PFA halts. Given that the PFA halts and does not leak, the probability of acceptance is strictly less than  $1/2$ , as the PFA is keeping a counter, and the probability of  $K$  suspect outcomes is more than  $1/2$  (for appropriately small  $K > 1$ , such as  $K = 2$ ).

A similar conversion to the one in the proof of Lemma 3.4.1 reduces the string-existence problem for the leaky PFA to the question of policy-existence in a UMDP under the discounted criterion, thus completing the proof.  $\square$

The addition of the finite counter to the PFA in the reduction in [24], while unnecessary for their proof of the undecidability of emptiness, is crucial in making the above proof work. Without it, or when  $K = 1$ , the probability of PFA acceptance can not exceed  $1/2$  whether or not the TM is accepting. On the other hand, infinite sequences corresponding to valid but non-halting TM computations would approach  $1/2$  with our construction of the leaky PFA. Thus with  $K = 1$ , there would be no distinction between cases of TM accepting and many cases of TM not accepting.

#### 3.4.4 Undecidability under a Negative Model

The optimality criteria studied to this point involve maximizing the expected benefits of executing a policy. An alternative goal would be to choose a policy likely to avoid disaster. In these cases (*state-oriented negative* models, see for example [103]) the objective is to minimize the probability of entering one or more designated negative states over the infinite horizon. We use the reduction in the previous proof to establish the undecidability of this particular negative model; the technique should be applicable to other negative models as well.

**Theorem 3.4.4** *Policy existence under the state-oriented negative model is undecidable.*

**Proof.** We reduce the string-existence question for the leaky PFA in reduction of Lemma 3.4.3 to this problem. Note that in the string-existence reduction for the leaky PFA, if the TM is accepting, there exist infinite (and therefore finite) sequences of symbols on which the probability of acceptance of the leaky PFA exceeds  $1/2$ . If the TM is rejecting, the probability of acceptance of no infinite sequence is over  $1/2$  (an infinite sequences with acceptance equals  $1/2$  may exist.). Take the rejecting absorbing state of the leaky PFA to be the state to avoid and the (undecidable) question would be whether there is an infinite sequence that avoids the rejecting state with probability exceeding  $1/2$ .  $\square$

#### 3.4.5 Inapproximability in POMDPs

It is not hard to argue that an inapproximability result similar to the one for PFAs also holds for POMDPs under the total undiscounted reward and the average reward criteria. However, under the discounted criterion, the optimal value is approximable to within any  $\epsilon > 0$ , due to the presence of the discount factor.

#### 3.4.6 Existence of Optimal Periodic Sequences

The question of the existence of optimal periodic sequences for UMDPs or optimal finite controllers for POMDPs in general (see the discussion section 2.7) is an interesting one. We are assuming that the initial distribution is given as part of the problem. A similar case to

the threshold-isolation can be made for motivating such a question: given a POMDP and an initial distribution, if we could determine whether it has an optimal finite controller, and if the answer is positive, then possibly finding the controller would be easier, as only the space of finite controllers needs to be searched. Next, we sketch the undecidability of the problem under the undiscounted total reward criterion. Consider the UMDP model, where an optimal finite controller corresponds to a periodic action sequence.

The PFA used in the reduction of section 3.2.2, say with  $K = 2$ , accepts no finite string with probability  $3/4$  or higher, and, when the TM is accepting, this probability is achieved in the limit by concatenating accepting sequences. When the TM is not accepting, the probability of PFA accepting any string is no more than  $1/4$ . Call the corresponding UMDP  $M_1$ , and let  $M_2$  be the UMDP model of Chapter 2 which did not have an optimal periodic sequence. We could construct another UMDP problem with a choice of transition to  $M_1$  and another to  $M_2$ . The transition to  $M_1$  gives the total expected rewards of close to  $3/4$  or less than  $1/4$  depending on whether the corresponding TM is accepting, and the transition to  $M_2$  can be designed to always give a total expected reward strictly in  $(1/4, 3/4)$ , for example around  $1/2$ . Consequently, for such a UMDP problem, the optimal sequence is periodic if and only if the TM is accepting.

The problem under the discounted criterion is also undecidable, but the proof is more intricate than above. Variations to the proof of Lemma 3.4.3 seem not to work and we explain the difficulty next. In the reduction, when the TM is accepting, there exists an optimal periodic sequence for the constructed PFA. On the other hand, a nonempty subset of those TMs that are not accepting have aperiodic configuration sequences. However, it is not clear that a TM with an aperiodic configuration sequence, the corresponding PFA's optimal action sequences (which have success probability  $1/2$ ) need necessarily correspond to the TM's aperiodic legal configuration sequence and consequently be aperiodic. Note that the PFA can only reject when the sequence is composed of false candidate accepting sequences. If a sequence never ends in an accepting or rejecting configuration, has a few miscounts and is otherwise legal and ends in a periodic sequence of configurations, it can have the limiting acceptance probability of  $1/2$  for the leaky PFA and therefore be optimal. Next, we circumvent this difficulty by redefining how the PFA accepts, so that when the

corresponding TM has an aperiodic configuration sequence, the optimal sequence for the PFA would also be aperiodic. We sketch the argument next.

The PFA of the reduction is leaky as before to capture discounting. However, in this case, the maximum probability of acceptance is  $1/2$  whether or not the TM is accepting. On reading a candidate TM sequence, on each transition from one configuration to the next, the leaky PFA tests the legality of the transition, and if any of the easy to check transition conditions fail, it rejects. We assume only counter contents may be illegal. The PFA performs a weak equality test as before to verify the legality of change in the counters in each transition. If the outcome is Suspect on *any* test, the PFA rejects immediately, and if the outcome is Correct (again on any test), it accepts immediately, and otherwise, in case of Indecision, it continues. If the PFA reaches the end of the input, it rejects. If it reaches the configuration corresponding to the end TM configuration (whether accepting or rejecting), it reinitializes to check for the start configuration and continues. Therefore, if the TM halts or ends in a periodic sequence, there is an optimal periodic sequence for the leaky PFA that achieves the maximum probability  $1/2$  in the limit. However, if the TM does not halt and its legal configuration sequence is aperiodic, the only optimal sequence for the PFA has to be aperiodic, as otherwise the probability of acceptance would be strictly less than  $1/2$  : any illegal sequence would have an illegal transition over which it's more likely that the PFA rejects given than it makes a decision on that particular test.

It follows that the Turing machine halting problem reduces to the periodic sequence existence problem: if the leaky PFA has an optimal periodic sequence, we know that the TM either halts or eventually repeats the same sequence of configurations forever. We can simulate the TM to determine which is the case in finite time. If the leaky PFA has no optimal periodic sequence, we know that the TM does not halt (and never repeats a configuration either). It follows that finite-controller existence is undecidable for discounted POMDPs as well.

We note that, as a consequence of Lemma 3.4.3 the seemingly easier problem of whether there exists a finite sequence that is optimal is also undecidable: we can add a single action to the start state of the leaky PFA of Lemma 3.4.3. This action leads to the accepting and rejecting states with equal  $0.5$  probability. Then the UMDP would have an optimal finite

sequence if and only if the TM is not accepting. Consequently, we also see that the related problem of given a UMDP, an initial distribution and an action sequence, whether we can do better than the given action sequence. (*i.e.*, the UMDP has higher expected value) than what the action sequence would provide, is undecidable, even in the discounted case.

### 3.5 Summary and Discussion

In this chapter we have shown that many natural infinite-horizon POMDP problems are undecidable. At this point, it is perhaps evident to the reader that the ideas behind the reductions are quite versatile and can be used to show undecidability in many problems arising in infinite-horizon POMDPs and related models, as we demonstrated for several such problems in this chapter.

We can attribute the undecidability of the problems to the combined effect of three significant sources of hardness in the underlying models. One source of hardness is the partial observability of the system state. The corresponding problems under fully observable models are decidable—for example, infinite-horizon fully observable MDPs are solvable in polynomial time. In general, it appears that partial observability significantly increases the difficulty of finding optimal and near optimal control policies [98, 45]. The second and third sources of hardness are stochastic state transitions and the criterion of optimal control under the infinite horizon: deterministic POMDPs and finite horizon POMDPs are decidable, though due to partial observability, many such problems are intractable [76]. It should be pointed out that the discounted POMDP is undecidable even under a constant discount factor, while in this case it is efficiently approximable, and therefore basically tractable. In this case, the undecidability likely stems from the problem that the optimal values of the corresponding infinite sums may not be finitely representable.

It appears now that while the presence of a discount factor aids in finding approximate solutions, it does not change the hardness of finding optimal solutions (for example policies) in the worst case. We briefly give three examples to illustrate. In all the worst case constructions, the instance is constructed in a way so that the structure of the optimal solution and other properties of interest are insensitive to the magnitude of the discount

factor. In this chapter, we showed an undecidable discounted POMDP problem. We can verify in this case that the hardness of the problem remains no matter what discount factor  $\beta$ ,  $0 < \beta < 1$ , is used (and the decision question adjusted). A second example is the UMDP problem given in Chapter 2 where the optimal sequence remains the same aperiodic sequence no matter what discount factor is used. Finally, it is noted by Littman [76] that the undiscounted MDP example problems which are used to show several policy improvement algorithms inefficient (see Chapters 4 and 7 for background on policy improvement) also work in the same way in showing inefficiency if a discount factor is added to the problem.

## Chapter 4

## FULLY OBSERVABLE MARKOV DECISION PROCESSES

This chapter provides the basics for the next two chapters, including problem definitions for infinite-horizon fully observable MDPs, in particular a subclass we call MDP(2), describing representation and algorithms for MDPs, and the MDP linear programming formulation. We also describe a first attempt at analyzing the policy iteration algorithm—a common dynamic programming method for solving MDPs—by showing several constraints on the pattern of action selections of the algorithm that limit the number of iterations of the algorithm. We use a probabilistic proof method to show that a process satisfying only such constraints can still take exponentially many iterations before stopping. We conclude with a short discussion of the failed approach and motivate the next chapter.

#### 4.1 Preliminaries

Throughout the remainder of the thesis, problems are represented and solved on graphs with  $n$  vertices and  $m$  edges. Generic vertices in graphs are notated with letters  $u$ ,  $v$ ,  $z$ , and  $s$ , and at other times we use an indexed notation  $v_i, 1 \leq i \leq n$ . The graphs, with the exception of graphs of the MDP(3) model to be discussed below, are directed multigraphs, *i.e.* there can be multiple directed edges between two vertices. An edge with start (tail) vertex  $u$  and end (head) vertex  $v$  is called a  $u$ - $v$  edge, and it is also an *in-edge* of vertex  $v$  and an *out-edge* or simply an edge of vertex  $u$ . We assume without loss of generality that each vertex has at least one out-edge. A walk is a finite progression of edges denoted by  $e_1 e_2 \cdots e_k, k \geq 1$  such that the end vertex of edge  $e_i$  is the start vertex of edge  $e_{i+1}$ . The start vertex of  $e_1$  and the end vertex of  $e_k$  are respectively the start and end vertices of the walk. A path is a walk with no repeated vertices except that the start vertex and the end vertex can be identical, in which case it is also called a cycle. A  $u$ - $v$  walk (path or cycle) is one with start vertex  $u$  and end vertex  $v$ .

Edges in the graphs have numeric parameters associated with them, which we assume are rational. We make the usual assumption that fractions are represented by ratios of integers. We use  $W$  to denote the absolute value of the integer with largest magnitude in the input. On such graph representations, a polynomial (time) algorithm is one with running time polynomial in  $n$ ,  $m$  and  $\log W$ , *i.e.* polynomial in the binary representation of the input. A strongly polynomial (time) algorithm is one whose run time is bounded by a polynomial in  $n$  and  $m$  only, and independent of  $W$  (the basic assumption is that all arithmetic operations such as division take constant time). At times, we use  $\tilde{O}$ , called soft  $O$ , to hide  $\log n$  and  $\log m$  factors.

The dynamic programming algorithms that we describe are iterative (or staged or phased), with each vertex starting with an initial value that can change during an iteration. Let  $t = 0$  denote the first time point, before the first iteration we consider the change in vertex values in an algorithm. Iteration  $i$ ,  $i \geq 1$ , occurs between time points  $t = i - 1$  and  $t = i$ , and the value of vertex  $v$  at time  $t$  is denoted by the sequence notation  $x_v^{(t)}$ , thus its initial value is  $x_v^{(0)}$ , and when we use indexed notation, for example for a vertex  $v_j$ , we use  $x_j^{(t)}$  to denote the value of the  $j$ th vertex at time  $t$ .

## 4.2 Fully Observable Markov Decision Processes

We will be using mostly graph terminology for the fully observable Markov decision problems in the remainder of the thesis to facilitate describing how the algorithms work with the graph representation. A Markov decision process (MDP) consists of a set  $V$  of  $n$  vertices or (system) states, and for each vertex  $v \in V$ , a finite set  $E_v$  of edge choices or actions (Fig. 4.1). Time is discretized and at each time point  $t$ ,  $t = 0, 1, \dots$  the system occupies one vertex  $v^{(t)}$ , which is called the state (or vertex) of the system at time  $t$ , and  $v^{(0)}$  is the initial state of the system. The means of change in vertex is edge choice. By choosing edge  $e \in E_u$  when the system is in state  $u$ , a *reward* of  $r(e)$  is obtained and the system transitions to vertex  $v$  with probability  $Pr(v, e)$ . The effects of edges, *i.e.* the rewards and the transition probabilities, do not change with time, and they are completely specified as part of the problem instance. A *policy*  $\mathcal{P}$  is a mapping assigning to each vertex  $v$  a single

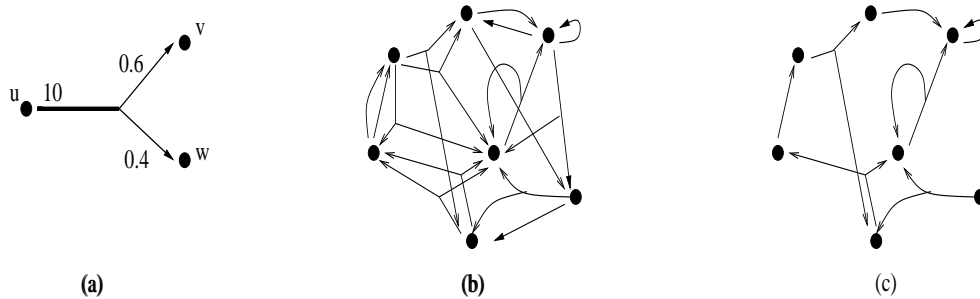


Figure 4.1: (a) An edge (action) in a general MDP model shown as a directed branching (hyper) edge. If the edge is chosen at state  $u$ , a reward of 10 units is obtained, and with probability 0.6 the next state is state  $v$ , and otherwise it is state  $w$ . (b) A hypergraph representation of an MDP problem, where at each state there are one or more edges available. Rewards and transition probabilities are not shown, but understood to label the edges. (c) A policy, *i.e.* an assignment of single edge to each vertex.

edge choice  $\mathcal{P}(v) \in E_v$ . In Fig. 4.1 graphical representations of an action, an MDP problem, and a policy are shown.

The value (vector) of a policy, denoted  $\mathcal{V}_{\mathcal{P}}$  is a vector of  $n$  values, where  $\mathcal{V}_{\mathcal{P}}[i]$  is the *value* of vertex  $v_i$  under policy  $\mathcal{P}$ , defined as the expectation of total reward  $\sum_{t=0}^{\infty} r(\mathcal{P}(v^{(t)}))$ , when  $v^{(0)} = v_i$ . We assume here that  $\mathcal{V}_{\mathcal{P}}[i]$  is bounded and well behaved for any policy  $\mathcal{P}$  and initial vertex  $v_i$ . We later show how extensions can be made to algorithms to discover ill-defined problems. By *evaluating* a policy we mean finding its value vector. This can be done in polynomial time by a matrix inversion (see also Section 4.5).

Define the *optimal value* of a vertex  $v$ , denoted  $x_v^*$ , to be the maximum value of vertex  $v$  over all policies. A desirable and simplifying property of MDPs is that there exists an *optimal policy* which simultaneously maximizes the value of all vertices. Thus problems we will be interested in are either computing the optimal value of a vertex or of all vertices, or finding an optimal policy. We refer to this optimization objective as the *total reward* (optimization) criterion.

### 4.2.1 Alternative Optimality Criteria

Two common alternative ways of defining vertex value under a policy  $\mathcal{P}$ , as described in the previous chapters, are either (1) the expectation of the *discounted* total reward,  $\sum_{t=0}^{\infty} \beta^t r(\mathcal{P}(v^{(t)}))$ , where  $\beta$  is the discount factor,  $0 \leq \beta < 1$ , or (2) the average reward per time step. Both alternative quantities are always well defined and it is not hard to reduce the corresponding problems to the total reward specification. For example, the discounted model can be transferred to an undiscounted total reward one by changing each edge in the old model to have an extra transition with probability  $1 - \beta$  to a zero valued vertex (a vertex with only one edge defined that leads back to itself and has reward zero), and the edge's old transition probabilities are normalized appropriately by being multiplied by  $\beta$ . The average reward model is reducible to the discounted model by choosing a  $\beta$  sufficiently close to 1.0.

These different value definitions each yield their corresponding optimization criteria and problems, which we shall call the discounted and the average reward criteria and problems. The average reward deterministic MDP (DMDP) problem is equivalent to the maximum or the minimum mean cycle problem [1]. The DMDP problem is the focus of Chapter 6.

## 4.3 MDPs as Monotone Linear Programs

The MDP problem can be solved by formulating it as a linear optimization problem (see for example Puterman [103]), where we solve for optimal value  $x_v$  for each vertex  $v$  in a single linear program (LP). The LP is to minimize<sup>1</sup>  $\sum_{v \in V} x_v$ , subject to a set of constraints, one for each edge  $e \in E_v$  of each vertex  $v$ , of the form:

$$x_v \geq \sum_{u \in V} Pr(u, e)x_u + r(e). \quad (4.1)$$

Extracting an optimal policy, once optimal values are computed is not hard: for each vertex at least one inequality corresponding to an edge will be tight at the optimal vertex

---

<sup>1</sup>This minimization formulation is the more common (and perhaps simpler) LP formulation for MDPs in the MDP literature [103, 32, 35]. However, for closely related problems such as shortest paths, this formulation—where variables corresponding to shortest paths values are assigned to the vertices and feasibility conditions are formulated in terms of them—is regarded as the dual LP [98, 1].

values, and optimal edges correspond to the tight inequalities. A subtlety is that if there are multiple edges with tight inequalities for several vertices, not all edge choices lead to optimal policies. However, it can be shown that if for each vertex, among its edges with tight inequalities, one with highest reward is picked, an optimal policy is obtained.

The aforementioned linear program is an instance of optimization over a set of inequalities  $\mathbf{Ax} \geq \mathbf{c}$  where  $\mathbf{A}^T$  is *pre-Leontief* (see Cottle and Veinott [26]). A Matrix is pre-Leontief if it has one positive entry in every *column*. We call such a set of inequalities a *monotone* set to connote the property that the direction of change in variable values in order to satisfy the inequalities remains the same, *i.e.* when a point  $\mathbf{x}$  satisfies all the inequalities, and a component  $\mathbf{x}[i]$  is increased, in order to satisfy the inequalities again, it is not necessary to decrease any component, but several components may need to be increased. A symmetric inequality system which we also call monotone is  $\mathbf{Ax} \leq \mathbf{c}$ , with  $\mathbf{A}^T$  pre-Leontief. In [26], it is shown that the set of solutions of the monotone system  $\mathbf{Ax} \geq \mathbf{c}$  (*resp.*  $\mathbf{Ax} \leq \mathbf{c}$ ), constitutes a semi-lattice relative to the *meet/min* operation  $\wedge$  (*resp.* *join/max* operation) defined as:

$$\mathbf{x} \wedge \mathbf{y} = (\min\{\mathbf{x}[1], \mathbf{y}[1]\}, \dots, \min\{\mathbf{x}[n], \mathbf{y}[n]\})^T.$$

Thus if the feasible set is not empty and it is bounded from below, there exists a feasible point, the greatest lower bound (glb) of the semi-lattice, where simultaneously all variable values are minimized. Therefore, the optimal vertex values in an MDP problem form the glb of the feasible set of the inequality system in the corresponding LP formulation. The dynamic programming MDP algorithms that we describe below start with infeasible points and approach the glb or the lub of the feasible set of the linear program. Thus the question arises whether the MDP algorithms can be extended to handle feasibility questions in any monotone linear inequality system (monotone LI), *i.e.* given a monotone linear inequality system, determine whether it is feasible and if so report the glb or the lub if they exist or otherwise some feasible point. In Section 5.4, we show this is the case for two variable per inequality systems (see below), and furthermore, we show efficiency of an algorithm in this class.

### 4.3.1 Two Variable Per Inequality Problems

Any linear programming problem is polynomially reducible to a corresponding problem where each inequality in the constraint set involves at most three variables. This is done by a standard technique of introducing new variables. Linear systems with two variables per inequality tend to be easier to solve in general and include important optimization problems such as shortest paths and many network flow problems [25, 1]. In Chapter 5, we extend the dynamic programming algorithms for MDPs to solve the feasibility problem in linear systems with two variable per inequality (TVPI or LI(2)). We assume that there are  $n$  variables denoted by  $x_i, 1 \leq i \leq n$ , and  $m$  inequalities, and each inequality, without loss of generality, is of the form  $ax_i \pm bx_j \geq c$ , where  $a$  and  $b$  are nonnegative numbers, but  $c$  is not restricted.

It follows from the definition of a monotone linear system that in a monotone TVPI problem, each inequality is of the *monotone* form  $ax_i - bx_j \geq c$ . We take single variable lower and upper bounds (where  $a = 0$  or  $b = 0$ ) to be of this form too. In general TVPI, non-monotone inequalities, of the form  $ax_i + bx_j \geq c$ , can also appear. As discussed above, the MDP algorithms appear to be geared to solve only monotone systems, but in case of two variables per inequality, it has been noted that the more general problem is in fact reducible to the monotone case: In Edelsbrunner et. al. [37] a reduction from a TVPI problem to a monotone version in the special case where coefficients are  $\pm 1$  is shown, and Hochbaum et. al. [54] note that the reduction works for general TVPI as well. This reduction, described next, is simple and efficient. We note that feasibility in general linear inequality systems is polynomially (in fact logspace) equivalent to the linear programming problem: the basic reduction from linear programming optimization to a feasibility problem equates the dual and the primal optimization functions subject to the dual and primal system of inequalities. Thus it is unlikely that a simple reduction from the general feasibility problem to the monotone case exists.

To reduce to a monotone TVPI, each variable  $x_i$  is replaced by the two variables  $x_i^+ = x_i$  and  $x_i^- = -x_i$ , each non-monotone inequality  $ax_i + bx_j \geq c$  by the two monotone inequalities  $ax_i^+ - bx_j^- \geq c$  and  $-ax_i^- + bx_j^+ \geq c$ , and each monotone inequality  $ax_i - bx_j \geq c$  is replaced

by two monotone inequalities  $ax_i^+ - bx_j^+ \geq c$  and  $ax_i^- - bx_j^- \leq -c$ . If the original system is feasible, it is not hard to see that the monotone system is feasible. On the other hand, given a feasible assignment of values for the monotone system, for each pair of value assignments  $x_i^+ = h$  and  $x_i^- = j$  in the feasible assignment, we can verify that letting  $x_i = 1/2(h - j)$  yields a feasible assignment in the original system. Thus, the reduction takes  $O(m)$  time, creates a monotone system with at most  $2n$  variables and  $2m$  inequalities, and conversion from a feasible point in one system to another is easy.

#### 4.4 MDP(3) and MDP(2)

Any MDP problem can be polynomially reduced to one where for each edge the maximum number of possible next vertices is two. This can be done by introducing extra vertices and edges as necessary. The corresponding LP would then have at most three variables per inequality. Our focus in Chapter 5 will be on a subclass of MDPs we call MDP(2), where each inequality involves at most two variables, and we use MDP(3) to refer to the general problem. Thus MDP(2) problems are a special case of TVPI problems. The algorithms we describe work on a directed graph representation of the MDP(2) model which we explain next.

Let us first consider a graphical representation of MDP(3). In Fig. 4.1 on page 51, parts a, b, and c, an example edge, a problem graph, and a policy in an MDP(3) model are shown. The edge in Fig. 4.1a corresponds to the linear inequality  $x_u \geq .6x_v + .4x_w + 10$ . Thus each edge in MDP(3) is a branching edge, or a hyper-edge, and is parameterized by three numbers: its reward and the transition probabilities to the two end vertices.

In MDP(2) problems, each edge corresponds to an inequality involving at most two vertex variables, for example  $x_u \geq .7x_v + 2$ . We take it that the remaining possibility of transition is to an absorbing vertex with zero reward. Such an edge is viewed as a directed  $u$ - $v$  edge (Fig. 4.2a). There can be multiple  $u$ - $v$  edges, thus an MDP(2) problem is viewed as a problem on a directed multigraph (Fig. 4.2b). A policy in MDP(2) is a subgraph of the MDP(2) graph where each vertex has out-degree 1. Thus an MDP(2) policy is composed of one or more cycles and paths to cycles, as in Fig. 4.2c.

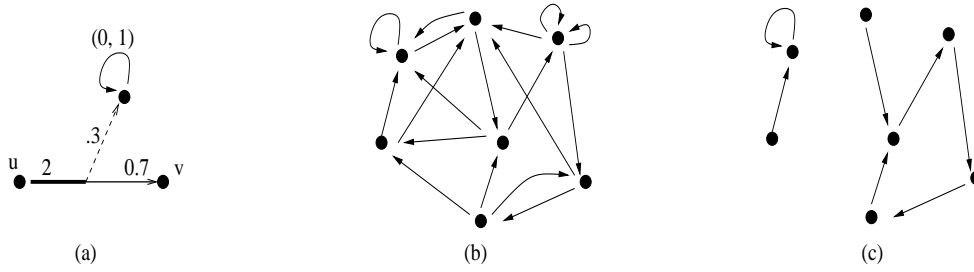


Figure 4.2: (a) A single MDP(2) edge referred to as  $u$ - $v$  edge, with reward 2 and slope (transition probability .7). The remainder of the transition probability is assumed to go to a zero reward vertex with no choice. (b) An MDP(2) problem graph (the side branches of edges not shown), and (c), an MDP(2) policy.

#### 4.4.1 Paths and Value Functions in MDP(2)

Each  $u$ - $v$  edge in an MDP(2) problem is parameterized by two numbers: its reward or  $y$ -intercept  $r(e)$ , which we denote by  $y_e$  for short, and the probability of transition to its end vertex, which we call its *slope*, and denote by  $m_e$ . We extend the two parameters of an edge to characterize walks in a straight-forward way: the slope  $m_p$  of a  $u$ - $v$  walk  $p$ , is the probability of reaching  $v$  from  $u$ , if  $p$  is followed, and the reward  $y_p$  is the expected reward of following the walk. Assume  $p = e_1 e_2 \cdots e_k$ , (where the end of  $e_i$  equals the start of  $e_{i+1}$ ) and let  $y_i$  and  $m_i$  denote the reward and slope parameters for edge  $e_i$  in the walk, and we obtain the following expressions for  $m_p$  and  $y_p$ :

$$m_p = \prod_{1 \leq i \leq k} m_i, \quad \text{and} \quad (4.2)$$

$$y_p = y_1 + m_1(y_2 + m_2(y_3 + \cdots)) = \sum_{1 \leq i \leq k} \left( \prod_{1 \leq j < i} m_j \right) y_i. \quad (4.3)$$

For each  $u$ - $v$  walk  $p$  define its *value function*  $f_p(x) = y_p + m_p x$ . The value  $f_p(x)$  is the expected value of  $u$ , if walk  $p$  is followed and the expected value of  $v$  is  $x$ . Thus  $f_p$  is the function composition of the constituent value functions corresponding to the single edges in  $p$ : If  $p = e_1 e_2 \cdots e_k$ , then  $f_p(x) = (f_1 \circ f_2 \circ \cdots \circ f_k)(x) = f_1(f_2(\cdots(f_k(x))))$ . Such

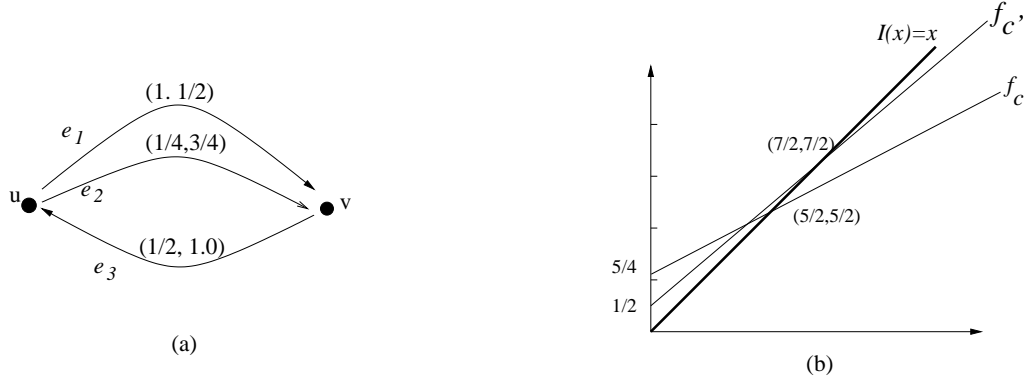


Figure 4.3: (a) Two  $u$ - $u$  cycles,  $c = e_1e_3$ , and  $c' = e_2e_3$ , where  $e_1$  and  $e_2$  are  $u$ - $v$  edges, and  $e_3$  is a  $v$ - $u$  edge.  $e_1$  has reward (y-intercept) 1, and transition probability (slope)  $1/2$ . (b) The value functions  $f_c(x) = 5/4 + 1/2x$ , and  $f_{c'}(x) = 1/4 + 3/4x$ , corresponding to  $u$ - $u$  cycles  $c$  and  $c'$  shown with the fixed points.

linear functions are nondecreasing because the slope  $m_p \geq 0$ . Different value functions can correspond to the same cycle, as a cycle  $c$  containing two vertices  $u$  and  $v$  can be considered a  $u$ - $u$  cycle or a  $v$ - $v$  cycle, and while the functions have the same slope  $m_c$ , they have different y-intercepts in general. When important, we refer to the cycle as a  $u$ - $u$  path or cycle to clarify that the value function  $f_c$  corresponds to starting and ending the cycle at a particular vertex  $u$  of the cycle. The *gain* of a  $u$ - $u$  cycle  $c$  at a particular value  $x$  of  $u$  is defined to be  $f_c(x) - x$ . In Fig. 4.3, two cycles and their value function are shown.

The value of vertex  $u$  under a policy that has a  $u$ - $u$  cycle  $c$  in it, where  $c = e_1e_2 \cdots e_k$ ,  $e_i$  having parameters  $y_i$  and  $m_i$ , is,

$$\begin{aligned} y_1 + m_1y_2 + m_1m_2y_3 + \cdots + \left( \prod_{1 \leq i < k} m_i \right) y_k + \left( \prod_{1 \leq i \leq k} m_i \right) y_1 + \cdots &= y_c + m_c y_c + m_c^2 y_c + \cdots \\ &= y_c \sum_{i \geq 0} m_c^i. \end{aligned}$$

This quantity is unbounded if  $m_c \geq 1$  and  $y_c \neq 0$ . We assume for any  $u$ - $u$  cycle  $c$ ,  $m_c < 1$ . Then the value of  $u$  under cycle  $c$  is  $\frac{y_c}{1-m_c}$ , which is also the unique intersection of the function  $f_c$  with the identity function  $I(x) = x$ . We call this intersection point the *fixed point* of  $f_c$ . Thus our assumptions on MDP(2) can be summarized as:

1. Edges have nonnegative slope not greater than 1.
2. All cycles have slope less than 1.

The first assumption follows directly from the semantics of MDPs, and is not necessary in our analysis of efficiency of algorithms. However, it does allow for Dijkstra-style shortest-paths algorithms that speed up computations, as we will see in the next chapter. The second assumption prevents ill-defined problems as seen above, and makes analysis easier in general. In the section on TVPI, we generalize the algorithms and remove these assumptions.

We see that policy evaluation in MDP(2) is simple as a policy is a subgraph where each vertex has out-degree one. For each cycle in the policy, we can find the value of a vertex in the cycle by computing the slope and the y-intercepts of the cycle. For a vertex  $u$  with path  $p$  to a vertex  $v$  in a cycle with value  $x_v$  under the cycle, the value of vertex  $u$  is  $f_p(x_v)$ . It follows that policy evaluation in MDP(2) takes  $O(n)$  operations.

#### 4.4.2 *Deterministic MDPs and the Mean Cycle Problem*

Two important and related MDP problems are deterministic MDPs under the discounted and average reward criteria. In a deterministic MDP, actions' effects, *i.e.* the transition probabilities, are deterministic, thus each action may lead to exactly one state. The deterministic MDP with the average reward criterion is equivalent to the minimum (or maximum) mean cycle problem [1]: Given a weighted directed graph, find the mean weight of the cycle with minimum mean weight among all cycles in the graph, where the mean weight of a cycle is the sum of its edge weights divided by its number of edges. Negation of weights shows that the maximization problem is equivalent to the minimization problem, and the maximization problem is the maximum reward per step problem of the deterministic MDP under the average reward criterion. We will refer to this problem as the mean cycle problem, and it will be the main subject of Chapter 6.

The mean cycle problem can be reduced to the discounted deterministic MDP problem, by picking the discount factor  $\beta$  to be sufficiently close to 1. The average weight of the optimal cycle found in the discounted problem is then the maximum mean value. See

for example [125] for details of the reduction. On the other hand the deterministic MDP problem with discount factor  $\beta < 1$  is a special case of the MDP(2) problem, where all edges have slope equal to  $\beta$ . We use the term deterministic MDPs to refer to the discounted deterministic problem.

#### 4.5 Iterative Dynamic Programming Algorithms For MDPs

Two commonly used dynamic programming algorithms for MDPs are value iteration and policy iteration. We describe the details of each algorithm and discuss different aspects of the algorithms briefly in the context of the MDP(2) problem. Books, such as [32, 10, 103], offer extensive material on dynamic programming techniques for MDPs.

Rough pseudocode for versions of the two algorithms are shown side by side in Fig. 4.4 to illustrate their small difference. Both algorithms can be seen to keep track of a vector of vertex values and visit policies from one iteration to another. The important difference is the extra policy evaluation step in policy iteration. The policy iteration algorithm is the main dynamic programming algorithm that we analyze in Section 5.3, while algorithms based on value iteration are the subject of Chapter 6.

Both algorithms keep track of a changing value for each vertex through the iterations. In the case of MDP(2), using the assumption that cycles have slope less than 1, and otherwise when there is discounting, the initial vertex values can be arbitrary for the purpose of convergence to the optimal vector of values. With  $x_v^{(t)}$  denoting the value of vertex  $v$  at time point  $t$ , and  $x_v^{(0)}$  the initial value, the value of a  $u$ - $v$  edge  $e$  in iteration  $t$ ,  $t \geq 1$ , is  $f_e(x_v^{(t-1)}) = m_e x_v^{(t-1)} + y_e$ . A best edge is the edge having the highest value<sup>2</sup>, thus each iteration of value iteration does the following computation, called a *dynamic programming step* or a *Bellman step* or equation, for each vertex:

$$x_v^{(t)} \leftarrow \max_{v-u \text{ edge } e} f_e(x_u^{(t-1)}), \quad t \geq 1.$$

For policy iteration, using  $\mathcal{P}^{(t)}$  to denote the policy at time  $t$ ,  $t \geq 1$ , lines 3 and 4 of the

---

<sup>2</sup>In an implementation, it is natural that vertices are examined in order. In that case, the most recent value instead of the value from the previous iteration may often be used.

<ol style="list-style-type: none"> <li>1. Vertices start with initial values</li> <li>2. Repeat</li> <li>3. Each vertex gets its best edge value</li> <li>4. Until small change in value</li> </ol>	<ol style="list-style-type: none"> <li>1. Vertices start with initial values</li> <li>2. Repeat</li> <li>3. Each vertex chooses its best edge</li> <li>4. Evaluate policy</li> <li>5. Until no change in policy</li> </ol>
---	--

Figure 4.4: Value iteration and policy iteration algorithms.

pseudo-code do the following,

$$\mathcal{P}^{(t)}(i) \leftarrow \arg \max_{v_i-v_j \text{ edge } e} f_e(x_j^{(t-1)})$$

$$x_i^{(t)} \leftarrow \mathcal{V}_{\mathcal{P}^{(t)}}[i], \quad t \geq 1.$$

We now discuss aspects of each algorithm.

#### 4.5.1 Value Iteration

The value iteration algorithm is specially suited under the discounted total reward criterion, in which case formulating the stopping condition is relatively easy. In this case, the test in line 4 of value iteration is designed so that after the loop, vertex values are as close to their optimal values as desired. The choice of edges that gave the values in the last iteration form a policy with values close to the optimal. Specifically, in [103] it is shown that if for any vertex  $v$ , if  $x_v^{(t)} - x_v^{(t-1)} \leq \epsilon(1 - \beta)/(2\beta)$ , then for any vertex  $v$ ,  $|x_v^* - x_v| < \epsilon/2$ , and the choice of edges made in iteration  $t$  forms a policy that is  $\epsilon$ -optimal: the value of no vertex under the policy is more than  $\epsilon$  less than its optimal value. In the absence of discounting, finding a good stopping condition for obtaining  $\epsilon$ -approximate values and policies is more complicated, and can involve computations that are more expensive than using alternative means of solving the MDP.

When the initial vertex values are set to zero, value iteration can be viewed as a dynamic programming algorithm for computing optimal policies and values for the finite-horizon

MDP problem—the problem of finding an optimal policy for a given horizon  $k$ , *i.e.* given  $k$  time steps to make edge choices. With increasing number of iterations, the vertex values converge to the optimal (infinite-horizon) values, and the choice of vertex edges converge to an infinite-horizon optimal policy in the limit (with proper assumptions, such as existence of discounting, or in our case, cycles having slope less than 1). However, value iteration can be slow. We see this in case of MDP(2), when a cycle may be optimal due to high slope  $m_c$ , but it may have a relatively small reward  $y_c$ . Thus value-iteration may take many iterations to converge to an optimal or near-optimal policy or bring the value of a vertex to within say a constant factor of its optimal value. This is basically formalized in the statement that value iteration is a pseudo-polynomial algorithm [119], meaning that it has a run time polynomial in  $n$ ,  $m$ , and  $W$  (versus  $\log W$ ) (see [119, 76]). However, as we will see, on MDP(2) problems, value-iteration variants or basically a sequence of value propagation or Bellman-Ford operations [1], are used in binary search schemes to give polynomial time algorithms for solving the MDP(2) problem. In Chapter 7, value iteration is shown to converge to an optimal cycle in the optimal mean cycle problem in strongly polynomial time, and forms the core of simple algorithms for efficiently finding an optimal cycle.

We also note that value iteration and policy improvement methods (see below) can be viewed as instances of the family of *distance-label correcting* algorithms for the shortest paths problems [1]. The basic operation of picking the value of a best edge in an iteration is also referred to as a Bellman-Ford push step, and such an iteration is called a Bellman-Ford phase in the context of the shortest paths algorithms of this nature, credited to Ford and Bellman. Variants of these algorithms are among the fastest for solving shortest paths problems in practice [1].

#### 4.5.2 Policy Improvement Algorithms

The policy iteration algorithm is an instance of the class of policy improvement methods<sup>3</sup>. Such methods keep track of vertex edge choices as well as values, and the edge choices

---

<sup>3</sup>The term policy iteration is also used to refer to the generic policy improvement methodology. Here, we will use policy iteration to refer to the specific policy improvement algorithm of Fig. 4.4b wherein all vertices simultaneously execute the dynamic programming step in each iteration. This version of policy iteration is the classic algorithm and is also referred to as greedy policy iteration.

made at each iteration define a policy. The algorithms may begin with an arbitrary policy. We assume in each iteration of any policy improvement method that a vertex chooses its choice of edge from the previous iteration if that edge remains a highest valued edge choice. These algorithms stop when the policy does not improve: the policy at iteration  $t$ , is the same as the policy at iteration  $t - 1$  or the initial policy<sup>4</sup>. It can be shown that the policy defined by the edge choices is strictly improved from one iteration to another in the sense that the value of no vertex decreases under a latter policy, and the value of at least one vertex increases. Since there are a finite number of policies, policy improvement methods eventually stop. Correctness is then established by showing that a policy  $\mathcal{P}$ , for which no edge of any vertex has a higher value than the value of the vertex under policy  $\mathcal{P}$ , is optimal. Such arguments appear in for example [103, 58]. Policy iteration has running time at worst pseudo-polynomial, basically due to it being as fast as value iteration in improving the value of the vertices [76].

We will discuss a few possibilities for policy improvement methods next. Call a vertex *switchable* when a different edge becomes higher valued than its edge choice of the previous iteration. Policy iteration alternates evaluating a policy and switching *all* switchable vertices to create a new policy. Another example of policy improvement would be one where each vertex is examined for switchability in a round-robin fashion, and whenever a (switchable) vertex changes edge, the algorithm evaluates the newly created policy, updating all vertex values. We will refer to this version as *scan*. In this work, we will concentrate on analyzing policy iteration, but we note whenever similar results apply to *scan*.

There are other policy improvement variations. The worst case running time of several heuristics for switching a single switchable vertex is shown exponential on MDP(3) by Melekopoglou and Condon [89]. A very plausible heuristic proved inefficient is one of choosing to switch the switchable vertex with maximum resulting difference in vertex value in the policy before the switch and the new policy after the switch. The counterexamples are shown under a total reward criterion, and Littman [76] notes that the algorithms are

---

<sup>4</sup>Policy iteration in Fig. 4.4 is presented in a way to emphasize its similarities to value iteration. In this way, we need to assume at least two iterations of the Repeat loop before the stopping condition is meaningful.

not even pseudo-polynomial by observing that the algorithms still take exponentially many operations to find the optimal policy on the same examples even when a constant discount factor is added to define a discounted problem. This is somewhat unexpected, as value iteration and policy iteration are pseudo-polynomial on MDP(3).

Policy improvement methods are dual algorithms for the LP formulation of 4.1 (see for example Denardo [32]). Policies correspond to the vertices of the feasible region of the dual linear program, and the algorithms jump from one vertex of the feasible region to another. These algorithms work their way to optimality in the feasible region of the dual, and to feasibility and in fact to the glb of the feasible region, where the optimal point (optimal values vector) exists, of the primal LP. Those that switch a single vertex at a time correspond to pivoting simplex algorithms, and differ on the choice of pivot. See also Denardo [32] for correspondence made between policy iteration and block pivoting in simplex methods. As mentioned above, these algorithms are also basically variations of distance-label correcting algorithm which in turn are identified with network simplex algorithms for shortest paths [1]. However, for several plausible heuristics of choice of pivot, examples have been constructed, the work in [89] being one, on which the corresponding simplex algorithms have exponential running time. It is an open question as to whether bad LP examples for the “scan” simplex algorithm exists. See chapter 7 for further discussion.

#### ***4.6 An Analysis of Policy Iteration at a High Level***

In this section, we describe a first attempt at deriving a subexponential bound on the number of iterations of policy iteration on general MDPs. We show that policy iteration abides by a few constraining rules that limit the type of policies it visits, and thus the number of iterations it takes to find an optimal policy. To some extent, the analysis is similar in approach to the analysis of Mansour and Singh [84]. We show that a process constrained this way can still take exponentially many iterations. Deriving an explicit example seems difficult so we use a probabilistic proof method to show the long run time. The bounds

derived by Mansour and Singh are also exponential, for example<sup>5</sup>  $2^{n/64}$ , though less than the total number of possible policies  $2^n$ . We discuss the weakness of the constraints in deriving a better bound and motivate investigating the MDP problem more closely in the following section.

Two interesting constraining rules on the sequence of policies that policy iteration visits arise when we look at properties of the set of all possible policies in an MDP problem and apply the kind of monotonicity properties that such a policy space enjoys and which we describe now. The question arises whether the constraints accumulate fast enough to stop policy iteration in a sub-exponential number of steps. The monotonicity properties can also be used to show the correctness of policy improvement algorithms as we will mention briefly. The monotonicity properties and the derived constraining rules also apply in related two-player MDP game problems, and a sub-exponential bound derived would have had desirable consequences in that context as well.

To see the constraints, it is best that we assume that each vertex has two action choices, denoted by 0 and 1, and each policy  $\mathcal{P}$  is thought of as a vector of  $n$  bits,  $n$  being the number of vertices, where  $\mathcal{P}[i]$  is the assignment of the action choice to vertex  $v_i$ . For example 000 denotes a policy in a three state problem where all states are assigned action 0. For a policy  $\mathcal{P}$ , recall  $\mathcal{V}_{\mathcal{P}}$  denotes its value vector, thus  $\mathcal{V}_{\mathcal{P}}[i]$  denotes the value of vertex  $i$  using policy  $\mathcal{P}$ . We write  $\mathcal{V}_{\mathcal{P}} < \mathcal{V}_{\mathcal{P}'}$ , if  $\mathcal{V}_{\mathcal{P}}[i] \leq \mathcal{V}_{\mathcal{P}'}[i]$ ,  $1 \leq i \leq n$ , and for some  $j$ ,  $\mathcal{V}_{\mathcal{P}}[j] < \mathcal{V}_{\mathcal{P}'}[j]$ . We extend this notation to policies, so we write  $\mathcal{P} < \mathcal{P}'$  whenever  $\mathcal{V}_{\mathcal{P}} < \mathcal{V}_{\mathcal{P}'}$ .

By switching a vertex in a policy we mean toggling its control choice in the policy to create a new policy (if the choice is 0, change it to 1, and vice versa), while leaving the rest of the policy unchanged. Similarly, switching a set of vertices in a policy means switching every vertex in the set, and leaving the rest of the vertices unchanged. Given a policy  $\mathcal{P}$ , a set  $S \neq \emptyset$  of the vertices is said to be *switchable* (in  $\mathcal{P}$ ) if switching  $S$  results in an increase in the value of at least one vertex in  $S$ , and the value of no vertex in  $S$  decreases. A vertex  $v$  is switchable if the set  $\{v\}$  is switchable. Switchability properties by definition are always considered with respect to a policy, and in what follows mention of an explicit policy is

---

<sup>5</sup> $n$  is the number of states and it is assumed without loss of generality, that each state has two action choices.

often omitted.

MDPs have a monotonicity property in that switching a switchable set is not only locally beneficial (with respect to the values of the vertices in the set) but also overall beneficial (the value of other vertices increase or remain the same). These properties stem from the monotonicity of the linear program formulation of MDP problem, but they also hold in the two player problem variations, with proper definitions of a value of a player's policy. The MDP problem under the total reward criterion also has other simplifying properties:

**Lemma 4.6.1** *Switching any non-empty set of switchable vertices in a policy  $\mathcal{P}$  results in an improved policy  $\mathcal{P}' : \mathcal{P} < \mathcal{P}'$ . If a set of vertices is switchable then a vertex in the set is switchable.*

The lemma is established in various forms in the process of showing that policy iteration is correct. See for example text books on MDPs [103, 58]. In each iteration, policy iteration switches a vertex if and only if the vertex is switchable. Therefore the lemma shows that policy iteration is correct: in each iteration, policy iteration switches a switchable set, and thus strictly improves the policy. Consequently, no policy is repeated, therefore policy iteration stops in a finite number of iterations. The policy that policy iteration ends in does not have a switchable vertex. Again, from the following corollary to Lemma 4.6.1 it follows that such a policy must be optimal<sup>6</sup>. The corollary is used in deriving the constraining rules.

**Corollary 4.6.2** *Consider two policies  $\mathcal{P}$  and  $\mathcal{P}'$  and assume  $\mathcal{P} < \mathcal{P}'$ . Let  $S$  denote the vertices where the choice of edges is different in  $\mathcal{P}$  and  $\mathcal{P}'$ . There is some vertex  $v \in S$  that is switchable in  $\mathcal{P}$ .*

We can now describe the constraints bounding the number of iterations of policy iteration. Consider the sequence of policies  $\{\mathcal{P}^{(t)}\}$  that policy iteration visits at time points  $t = 0, 1, \dots$ , where  $\mathcal{P}^{(0)}$  is the initial policy before iteration 1. We say a policy is *implied* at iteration  $t$ ,

---

<sup>6</sup>Existence of an optimal policy can be shown independently, or can also be shown by classifying vertex sets whose switching causes decrease in value, no change in value or gives mixed results, and showing that similar monotonicity and decomposability properties applies to them. These properties and their consequences are discussed in [82] in detail.

$t = 1, 2, \dots$ , if it can be constructed by switching a non-empty subset of the set of switchable vertices of policy  $\mathcal{P}^{(t)}$  (so policy  $\mathcal{P}^{(t+1)}$  is by definition a policy implied at iteration  $t$ ). Fig. 4.5 shows a possible trace of policy iteration on a three vertex problem. In the initial policy, all vertices have chosen choice 0. At iteration 2 (going from 111 to 001), the set of switchable vertices is composed of vertices one and two, as policy iteration switches all switchable vertices. Therefore, policies 011 (switch bit 1 only), 101 (switch bit 2 only), and 001 (switch the set of all switchable vertices) are all implied. From the decomposability and monotonicity properties, all policies implied at  $t$  are superior to policy  $\mathcal{P}^{(t)}$  and hence all policies  $\mathcal{P}^{(j)}, j \leq t$ . In the above example, we have  $111 < 011$ ,  $111 < 101$ , and  $111 < 001$ .

Consider any implied policy  $\mathcal{P}$  at some iteration  $t \geq 2$ , and a policy  $\mathcal{P}^{(j)}, j \leq t$ . We have  $\mathcal{P}^{(j)} < \mathcal{P}$ , so it follows from Corollary 4.6.2 that some vertex from the set of vertices where  $\mathcal{P}^{(j)}$  and  $\mathcal{P}$  differ in action choice must be switchable in  $\mathcal{P}^{(j)}$  (in iteration  $j$ ) and must have the bit value (action choice) that it has in policy  $\mathcal{P}^{(j+1)}$  in iteration  $j + 1$  as policy iteration switches all switchable vertices. Put another way, the switched bit value of at least one switchable vertex in  $\mathcal{P}^{(j)}$ , must be the same as the bit value in  $\mathcal{P}$ . We shall formulate this constraint as a Boolean disjunctive clause. Each transition from iteration  $t$  to iteration  $t + 1$  introduces a Boolean clause that all implied policies at iterations  $t + 1$  and later should satisfy. Let  $b_i$  denote the Boolean variable for the bit value (the action choice) of  $v_i$ . If vertex  $v_i$  in transition  $t$  changes bit value from 1 to 0, then the clause for time  $t$  will contain  $\bar{b}_i$  (negation of  $b_i$ ), and if it changes from 0 to 1, it will contain  $b_i$ . The (clausal) constraint for  $t$  is shown next to policy  $\mathcal{P}^{(t)}$  in Fig. 4.5. Hence we define a *legal* sequence of binary bit vectors, that policy iteration *could* visit, to be one where:

1. No vector repeats immediately (no empty clausal constraint).
2. The disjunctive clausal constraint produced at time  $t$  is satisfied by all the implied bit vectors at time  $t + 1$  and later.

Of course, due to the clausal constraints, an immediate consequence is that no vector can appear more than once in the entire sequence. We will next focus on legal sequences.

0	000
1	111 ( $b_1 \vee b_2 \vee b_3$ )
2	001 ( $\bar{b}_1 \vee \bar{b}_2$ )
3	101 $b_1$
4	100 $\bar{b}_3$

Figure 4.5: A legal sequence of policies visited by policy iteration in a three vertex MDP problem.

#### 4.6.1 The Growth of the Length of Legal Sequences

We are interested in the growth of the length of a longest legal sequence of bit vectors as a function of  $n$ , the number of bits or vertices. We denote this measure by  $L(n)$ . From the discussion above, it is the case that the number of iterations of policy iteration is upper bounded by  $L(n)$ . However, a lower bound on  $L(n)$  does not necessarily give a lower bound on policy iteration on MDPs as MDPs may have further structure.

The constraints on legal sequences are effective when we consider fixing the maximum size of the number of bits that switch in each iteration. For example, if we fix the maximum at 1, then we can have no more than  $n$  iterations. On the other hand, if we fix it at  $n$ , we cannot have more than 1 iteration. Thus, a question that arises is how large can  $L(n)$  get as  $k$ , the number of fixed vertices to switch at each iteration, increases from 1 to  $n$ . But intuition might say that for mid-range values of  $k$ , say  $n/2$ , a process constrained by the constraints may continue for a while, and  $L(n)$  can be large, possibly exponential.

We next argue that  $L(n)$  is indeed exponential in  $n$ . We don't have an explicit construction, but we give a simple argument using the probabilistic method. Consider the following process, where in each iteration each bit is selected independently at random, and if selected it is negated (switched). Thus at each iteration a clause with expected size  $n/2$  is constructed. An illegal iteration which produces an illegal sequence is one where a collection of bits are selected and negated that contain all bits satisfying some existing clause. We

show that with nonzero probability, exponentially long sequences of legal iterations may occur.

1. Repeat
2. Each bit is selected independently with probability  $1/2$ .
3. The selected bits are negated (switched).

Let us call a clause (critically) constrained at time  $t$  if a “small,” say  $\frac{n}{16}$ , fraction of the bits at time  $t$  satisfy it. We can see that the probability that a clause is constrained is exponentially small by using a Chernoff bound as follows. Let  $X_i$  denote the event that a bit  $i$  satisfies a clause at time  $t$ . Then  $X_i, 1 \leq i \leq n$ , are independent Bernoulli trials with probability of  $1/4$  of occurring: The probability that a bit satisfies a clause is the probability that it is in the clause first, which has probability  $1/2$ , times the probability that its current value satisfies the clause given that it is in the clause, which has conditional probability  $1/2$ . Then  $X = \sum_{i \leq n} X_i$  is binomially distributed with mean  $n/4$ , and using the Chernoff bound  $Pr(X < (1 - \delta)\mu) < e^{-\mu\delta^2/2}$ , for  $X$  distributed binomially [91], we obtain  $Pr(X < n/16) < e^{-n/16}$ .

The collection of the  $t$  events  $E_i$ ,  $E_i$  being that clause  $i$  is unconstrained at time  $t$ , is also mutually independent, as clause selections are performed independently, thus the probability of the event  $B_t$  that all  $t$  clauses at time  $t$  are unconstrained is greater than  $(1 - e^{-n/16})^t$ . Given event  $B_t$ , the probability that the next iteration, iteration  $t+1$ , switches all the bits satisfying a given existing (unconstrained) clause, is not greater than  $2^{-n/16}$ , and thus the probability that some clause becomes unsatisfied is not greater than  $t2^{-n/16}$ . Therefore the probability of a legal iteration given  $B_t$  is at least  $1 - t2^{-n/16}$ . Let  $C_t$  denote the event that all  $t$  clauses are unconstrained and legal transition is made at time  $t$ . Thus  $Pr(C_t) > (1 - t2^{-n/16})(1 - e^{-n/16})^t \geq (1 - t2^{-n/16})(1 - te^{-n/16}) \geq 1 - 2t2^{-n/16}$ . Therefore the probability of an illegal iteration at time  $t$  is not greater than  $1 - Pr(C_t) < 2t2^{-n/16}$ , and the probability of some illegal iteration at or before iteration  $t$  is not greater than  $2t^22^{-n/16}$ . We see that for say  $t = 2^{n/64}$ , the probability of an illegal iteration is strictly less than one, so there are exponentially long legal sequences with nonzero probability.

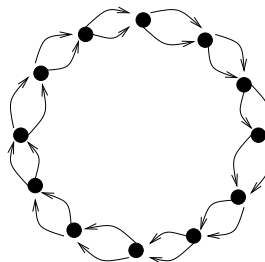


Figure 4.6: An MDP(2) “necklace” graph.

#### 4.7 Summary and Discussion

This chapter provided some background and definitions for the following two chapters in addition to describing a first attempt at analyzing the policy iteration algorithm. We will next discuss the failed analysis approach and motivate the analysis of the next chapter.

The clausal constraints of Section 4.6 ignore two structural properties of MDPs: (1) the underlying graph, or the connectivity of the vertices under the actions, and (2) the linearity of the problem. Thus the monotonicity properties given could hold for states that are not necessarily connected by the actions in a graph, or if connected, an action can represent any monotone, for example nondecreasing quadratic, function. Because of allowing for such a general monotone problem, it is not surprising that the clausal constraints do not constrain the number of iterations significantly.

Let us examine the connectivity and linearity constraints in a small example. Consider policy iteration on MDP(2). As policy iteration progresses on any MDP(2) graph, no cycle disappears and reappears in the policies seen during the progress of the algorithm, as the values of the vertices do not decrease during the algorithm: disappearance of a cycle establishes that the vertex values in the cycle strictly increase, thus the cycle cannot reappear. Now, consider MDP(2) graphs where every vertex has two edges connected to the same vertex in the form of a necklace as shown in Fig. 4.6. There are  $2^n$  possible cycles in the necklace graph, thus the constraint on cycle reappearance, which is basically a constraint derived from connectivity alone, does not reduce the number of iterations from

the worst case  $2^n$ . However, when a vertex  $u$  switches from a  $u$ - $v$  edge  $e_1$  to another  $u$ - $v$  edge  $e_2$  in an iteration  $t$ , it must be that  $f_{e_1}(x_v^{(t-1)}) < f_{e_2}(x_v^{(t-1)})$ . As these functions are linear and nondecreasing, and the value of vertices during policy iteration does not decrease, it follows that a vertex can switch at most twice during policy iteration on the necklace, and because at least one vertex switches in each iteration, this give a bound of  $2n$  iterations. A closer examination reduces the number of iterations to  $n$ . Thus the linearity of the problem, coupled with the graph structure, can constrain the number of iterations significantly. In the next chapter, we will formalize these constraints using *parametric analysis* and apply them to general MDP(2) problems.

## Chapter 5

**ALGORITHMS FOR MDP(2) AND TVPI PROBLEMS**

In this chapter, we describe new polynomial algorithms for MDP(2) and TVPI problems. We first present two algorithms for MDP(2) that are the simplified versions of existing TVPI algorithms. We then analyze policy improvement algorithms, with a focus on policy iteration. We establish that variants of policy iteration take polynomial time on MDP(2), and extend the algorithms to solve TVPI.

**5.1 Introduction**

In this chapter, we describe two main algorithmic techniques for efficiently solving the MDP(2) problem, one based on previous algorithms for solving TVPI, and another based on our analysis of the policy improvement methodology for solving general MDPs. The analysis leads to new polynomial time algorithms for MDP(2) and TVPI problems. The best running times for the new algorithms are currently  $O(mn^2(\log m \log W))$  for MDP(2), and  $O(mn^3(\log W))$  for TVPI. We expect that better analysis and/or variants of the algorithms can improve the running times, and that algorithms in this class will have competitive empirical performance with existing algorithms for TVPI and related problems.

Previous TVPI algorithms, for example the ones in [5, 87, 55, 20, 95], are based on using the Bellman-Ford value propagation (or value iteration) method [25], commonly coupled with binary or parametric search to find the optimal values. Several algorithms in this class have strongly polynomial bounds [87, 55, 20, 95]. We describe the basics of two such algorithms simplified for MDP(2). It would be interesting to extend these search techniques to solve MDP(3) problems, but the hypergraph structure of MDP(3) makes this difficult, and in particular the value propagation subroutines, at least without any modifications, fail. We describe these difficulties in Section 5.2.4.

The policy improvement variants which we show polynomial, may be seen to converge

quickly on the MDP(2) because they find and evaluate “quality” *cycles* in the graph, where evaluation of a cycle means finding the values of the vertices in the cycle if the cycle is used in a policy. A  $u$ - $u$  cycle has high quality for a vertex  $u$  roughly if the corresponding value function is highest or near highest valued among all  $u$ - $u$  cycles at the current value of  $u$ . These algorithms converge to the maximum vertex values from below, unlike the TVPI search methods, which narrow down to the optimal points from both sides of the points. We use two main techniques in analyzing policy improvement algorithms. One is parametric analysis, and another is viewing the algorithm as a Newton’s method for finding the zero of a *parametric* function. See for example [47, 1] for background on parametric analysis. The parametric function is a parameterized optimal value function, implicitly defined by the graph structure, and the Newton procedure converges to its fixed-point, *i.e.* the point at which it intersects the identity line (where optimal values lie). The parametric function is introduced in the section on parametric analysis, and it is linear, convex, and nondecreasing, guaranteed to intersect the identity line.

The parametric analysis yields a quasi-polynomial  $mn^{\log n}$  bound, for two policy improvement algorithms on MDP(2). The same analysis establishes that the two algorithms have strongly polynomial time for the deterministic (discounted) MDPs. The reason for the difference in run time is due to the number of slope changes in the parametric function. We show this number is bounded above by  $n$  in the deterministic case, which is tight, but by  $mn^{\log n}$  in the MDP(2) case. We also show that the algorithms advance from one slope change to a later one in polynomial time until they find the optimal values (the fixed point of the parametric function). The analysis of the same algorithms as Newton’s method gives polynomial bounds, but not strongly polynomial, on MDP(2). We give two analyses, one developed by us, and one due to Radzik [107] that establish the polynomial convergence. The policy improvement algorithms are then extended to handle TVPI problems. The algorithms shown polynomial are variations on policy iteration, and an open problem is to extend the analyses to the policy iteration algorithm.

The remainder of the chapter is organized as follows. In Section 5.2 we develop the ideas behind TVPI algorithms and present a binary search algorithm based on the work of Aspvall and Shiloach [5] and a strongly polynomial time algorithm based on Megiddo’s

technique [87].

We begin our analysis of policy iteration in Section 5.3 by defining and analyzing a parametric version of the MDP problem for the case of MDP(2) and deterministic MDPs. We then define *almost dags* or *adags*, a restriction of MDP(2) graph structure which makes the analysis more manageable. We analyze the run time of policy iteration on adags based on the insights obtained from the parametric analysis. We show a quasi-polynomial run time for policy iteration on adags, and a strongly polynomial run time for policy iteration on deterministic adags. The analysis on adags motivates our convergence analysis of policy iteration as a Newton’s method, and we take up that analysis in Section 5.3.3. It is shown that policy iteration is polynomial on adags, which motivates another polynomial policy improvement algorithm for adags and generalizations to efficient MDP(2) algorithms. In Section 5.4 we describe extensions to handle TVPI problems. We close the chapter with a discussion of open problems in Section 5.5. The discussion motivates Chapter 6.

## 5.2 Algorithms Based on Bellman-Ford Style Value Propagation

The class of algorithms we discuss in this section are all based on solution algorithms for the following *query* problem:

**Problem 5.2.1** *Given a vertex  $s$  and a value  $x$ , determine whether the optimal value of  $s$  is greater than  $x$ .*

The query problem is solved by making a series of value propagation operations similar to the ones used in the Bellman-Ford algorithm for finding shortest paths from a single source to all vertices. See for example [25, 1] for a description of shortest paths algorithms based on Bellman-Ford style value propagation. We next describe two algorithms for solving the query problem and then briefly present a binary search algorithm and a strongly polynomial algorithm for solving the MDP(2) problem using the solutions to the query problem.

### 5.2.1 Querying Based On Bellman-Ford Push Phases

We will describe two types of Bellman-Ford style algorithms for the query problem. The first, the Backward-Propagate algorithm given in Fig. 5.1, is perhaps simpler, and is similar

to the value iteration algorithm for MDPs. The algorithms can be changed to test whether  $x$  is equal to the optimal value of the queried vertex as well. Each iteration of either algorithm can be implemented to take  $O(m)$  operations with appropriate preprocessing and data structures, thus the run time of both of the algorithms is  $O(mn)$ . The proofs of correctness for both algorithms rely on the basic property that if  $f_c(x) > x$  for a  $u$ - $u$  cycle  $c$ , then  $x$  is less than the optimal value of  $u$  (there is at least one cycle under which  $u$  has a better value), and propagating value of  $x$  along the edges of the graph discovers the situation.

At the beginning of each iteration of Backward-Propagate, every vertex has a value that may be changed for the next iteration. Define the value of a  $u$ - $v$  edge  $e$  to be  $f_e(x)$  where  $x$  is the value of vertex  $v$  at the beginning of the iteration, and when  $v$  has value  $-\infty$ , then the value of the edge is defined to be  $-\infty$  as well, if its slope is nonzero, and otherwise will be its reward  $y_e$ . In each iteration, every vertex  $u$  chooses a highest valued edge and sets its value to it for the next iteration. Thus values of vertices get propagated from vertex to vertex along edges.

Backward-Propagate is limited in that it is guaranteed to solve the query problem only for those vertices that reside in a cycle in some optimal policy. This limitation is not severe, and the algorithm could still be used as the query subroutine, for example for algorithms that find the optimal policy for all vertices, or the strongly polynomial time algorithm described in next sections. We also remark that several variants of the algorithm are possible. For example, any lower bound (to the optimal value) to initialize vertex values would give a correct algorithm.

**Lemma 5.2.2** *Algorithm Backward-Propagate returns True only if the optimal value for queried vertex  $s$  exceeds  $x$ . If vertex  $s$  is in a cycle in some optimal policy, then the algorithm correctly solves query problem 5.2.1.*

**Proof.** Assume there is a  $s$ - $s$  cycle  $c$  where the value of  $s$  under the cycle is greater than  $x$ . Then we must have  $f_c(x) > x$ . Let  $c = e_0, \dots, e_{k-1}, k \geq 1$ , where  $e_0 = (u_0, u_1)$ , and  $e_{k-1} = (u_{k-1}, u_k), u_0 = u_k = s$  and let paths  $p_i = e_{k-i}, \dots, e_k, 1 \leq i \leq k$ , so that  $p_k = c$ . Then it's not hard to verify inductively that at iteration  $i$  vertex  $k - i$  will receive a value

1. Value of vertex  $s$  is set to  $x$ , all other vertex values to  $-\infty$ .
2. Repeat  $n$  times:
3. Each vertex chooses a highest valued edge and sets its value to it.
4. If vertex  $s$  gets a value greater than  $x$  then return True.
5. Return False.

Figure 5.1: The Backward-Propagate Algorithm

no less than  $f_{p_i}(x)$ .

Conversely, if vertex  $s$  gets a value greater than  $x$ , tracing the sequence of edges that resulted in the value, we must get a  $s$ - $s$  cycle  $c$  with  $f_c(x) > x$ , or a path with last edge having slope zero.  $\square$

The algorithm Forward-Propagate, given in Fig. 5.2, does not have the limitation of Backward-Propagate. While the algorithm Backward-Propagate propagates values in the reverse direction of the edges, the algorithm Forward-Propagate propagates values (in the form of queries) in the direction of edges. We obtained the idea of forward propagation from Oldham [95] where a similar algorithm is used in parametric binary search for solving the generalized shortest path problem. In each iteration of the Forward-Propagate algorithm one or more vertices may receive queries through one or more incoming edges. Querying for value  $y$ , through a  $u$ - $v$  edge means computing the minimum value  $x$  such that if value of  $v$  is greater than  $x$ , vertex  $u$  obtains a value exceeding  $y$ . In terms of the value function  $f_e$  corresponding to edge  $e$ , this means that we seek the minimum  $x$  such that for  $z > x$ ,  $f_e(z) > y$ . This is just an inverse computation on  $f_e$ , with special handling when  $m_e = 0$ : For each edge  $e$ , if slope  $m_e = 0$ , then if  $y \geq y_e$ , vertex  $v$  receives  $+\infty$  and ignores the value, and if  $m_e = 0$  and  $y < y_e$ , then vertex  $v$  receives  $-\infty$  and the algorithm returns True. Note that in line 5 of the algorithm, if a vertex receives a (finite) query value for the first time, the value is recorded for the vertex and propagated along its out-edges, and if the vertex has received a queried value before, the queried value is recorded and sent along the out-edges if only if the value is less than the last value the vertex was queried with.

1. Vertex  $s$  queries for value  $x$  through each of its out-edges.
2. Repeat  $n$  times:
3. For each vertex  $v$  receiving one or more queries do:
  4. Choose the minimum requested value  $\lambda$ . If  $\lambda = -\infty$ , return True.
  5. If  $\lambda \neq +\infty$ , and  $\lambda$  is less than the last queried value (if any), record  $\lambda$  as last value queried, and query  $\lambda$  along out-edges of  $v$ .
6. If in the  $n$ th iteration a vertex has received a value less than its last queried value return True. Otherwise return False.

Figure 5.2: The Forward-Propagate Algorithm.

**Lemma 5.2.3** *Algorithm Forward-Propagate, on a query for  $x$  on vertex  $s$ , returns True if and only if the optimal value of vertex  $s$  exceeds  $x$ .*

**Proof.** Assume the optimal value of  $s$  is greater than  $x$ . First note that if the algorithm returns True when restricted to making queries along some policy, then the algorithm would return True when run on the whole graph.

Now consider any policy  $\mathcal{P}$  in which vertex  $s$  has value greater than  $x$ . In this policy, vertex  $s$  has a path to vertex  $v$  in a cycle (including the possibility that  $v = s$ , *i.e.*  $s$  is itself in a cycle and some edge of the policy may have slope 0). Imagine making queries along the edges of this policy only. If in  $n$  iterations, a vertex receives a query for  $-\infty$ , we are done. Otherwise, the queries go around the cycle at least once (as the total path to cycle length and cycle length is not greater than  $n$ ). Thus vertex  $v$  will receive a 2nd query lower than the first, as we have  $f_c(x) > x$ . Once this happens, in every future iteration, some vertex in the cycle receives a query for a value less than its last received query value.

Now assume the algorithm returns True. It is not hard to see that if some vertex  $v$  has received  $-\infty$ , then the policy under which  $s$  has value exceeding  $x$  exists. Otherwise, the algorithm has had exactly  $n$  iterations. We trace the sequence of queries that resulted in a return value of True. This sequence forms a walk. If in iteration  $n$  a vertex receives a request less than its past query, the requested value must have gone around a cycle once, and the first vertex in the cycle must have received a query less than  $x$  to query again.

It follows that the cycle and the path to the cycle form a policy under which  $s$  has value exceeding  $x$ .  $\square$

### 5.2.2 Binary Search

Once we have an efficient algorithm for the query problem, we can use binary search to narrow down the range of the optimal value for any vertex efficiently, and eventually find the optimal value. This is the method used by Aspvall and Shiloach [5] for solving TVPI problems in polynomial time. To prove polynomial time for binary search we need a lemma such as 5.2.4 that gives rough bounds on the size and closeness of values of vertices under policies. Variants of the following lemma are proved elsewhere (such as [5]) as well. We will also use the lemma later when we analyze policy iteration.

**Lemma 5.2.4** *The value of any vertex under any policy lies in the range  $[-2nW^2, 2nW^2]$ . Two distinct values of a vertex under different policies are not closer than  $W^{-3n}$ .*

**Proof.** For any  $u$ - $u$  cycle  $c$ ,  $|y_c| \leq nW$ , and  $1 - m_c \geq 1 - \frac{1}{W}$ , thus the value of a cycle  $c$  (for any of its vertices), which is  $\frac{y_c}{1-m_c}$ , is not more than  $\frac{nW^2}{W-1} \leq nW^2$  in magnitude. Thus the value of any vertex, which has a path of value at most  $nW$  to some cycle is not more than  $2nW^2$  in magnitude.

Two distinct values of a vertex  $u$  under two cycles  $c$  and  $c'$  with intercepts  $y_1$  and  $y_2$  and slopes  $m_1$  and  $m_2$  respectively have a difference of  $\delta = \frac{y_1}{1-m_1} - \frac{y_2}{1-m_2} = \frac{((1-m_2)y_1 - ((1-m_1)y_2))}{(1-m_1)(1-m_2)}$ . Assume without loss of generality that  $\frac{y_1}{1-m_1} > \frac{y_2}{1-m_2} \geq 0$ . If we assume  $y_1 \geq y_2$  then we have  $\delta > (1 - m_2)y_1 - (1 - m_1)y_2 \geq y_1[(1 - m_2) - y_2/y_1(1 - m_1)] \geq y_1(m_1 - m_2)$ , and  $y_1 \geq \frac{1}{W}$ . Edge slopes are ratios of two nonnegative integers, each not greater than  $W$ , thus cycle slopes can be thought to be ratios with nonnegative integer numerator and denominator, where the denominator is not greater than  $1/W^n$ . Consequently, the smallest nonzero difference between two cycle slopes is not less than  $1/W^{2n}$ . Therefore  $\delta > \frac{1}{W^{2n+1}}$ . A larger difference is obtained if  $y_1 < y_2$  so  $1 - m_2 > 1 - m_1$ . The difference can be further reduced by a path of slope no less than  $\frac{1}{W^n}$ , thus we obtain the smallest difference is greater than  $W^{-3n}$ .  $\square$

Thus in  $O(\log W^n)$  queries, the interval for the optimal value for a queried vertex  $s$  is narrowed enough so no value of  $s$  under any policy, except its optimal value, appears in the interval. Forward-Propagate can be slightly modified to also return a policy under which  $s$  has value exceeding  $\lambda$ , and the value of  $s$  under the policy can be evaluated in  $O(n)$  time. Thus in  $O(mn^2 \log W)$  the optimal value and a policy supporting the value for any vertex can be reported using Forward-Propagate. Note that computing an  $\epsilon$ -approximation would require only  $O(mn \log(nW\epsilon^{-1}))$  time. We can apply the algorithm to each vertex so that in  $O(mn^3 \log W)$  time optimal values for all vertices are found. We expect that the speed ups that would reduce the run-time by a factor of  $n$  should be possible. For example, Aspvall [4] speed ups a similar algorithm to run in  $O(mn^2 \log W)$  time to solve the feasibility problem for the more general TVPI systems. Note that in this case, we want the extreme points of the feasible region, and not just any feasible point.

### 5.2.3 A Strongly Polynomial Time Algorithm

Megiddo proposed the first strongly polynomial time algorithm for TVPI, with a run time of  $O(mn^3 \log m)$  time [87]. In this section we will briefly describe the ideas behind the algorithm, as simplified for MDP(2). Other strongly polynomial algorithms, all based on the solution to the query problem 5.2.1 and having run time  $\tilde{O}(mn^2)$ , appear in [55, 20, 95].

Let us define an *optimal cycle* of a vertex, when the vertex resides in at least one cycle, to be a cycle under which the value of the vertex is no less than the value of the vertex under any other cycle containing it. Note that the optimal value of a vertex may be greater than the value of the vertex under its optimal cycle, when a vertex has a better path to another cycle. If an optimal cycle is found for each vertex, then computing an optimal policy and optimal values can be done by finding the value of each vertex under its optimal cycle, in  $O(n^2)$  time, and in another  $O(mn + n)$  iterations of value iteration (or push-phases), optimal values and an optimal policy for all vertices can be computed. Next we describe an  $\tilde{O}(mn^3)$  time algorithm for finding optimal cycles.

Recall that paths and cycles correspond to linear functions. Note that in the set of all cycles that share a vertex  $s$ , an optimal  $s$ - $s$  cycle  $c$  has the property that at the optimal

value  $x^*$  for  $s$ ,  $f_c(x^*) > f_{c'}(x^*)$ , for any other suboptimal  $s$ - $s$  cycle  $c'$ . Thus in order to find an optimal cycle, the algorithm first finds, for any pair of vertices  $u$  and  $v$ , the *highest valued*  $u$ - $v$  path: for two  $u$ - $v$  paths  $p_1$  and  $p_2$ ,  $p_1$  is higher valued than  $p_2$  if  $f_{p_1}(x^*) > f_{p_2}(x^*)$ . Thus the point of comparison for paths and cycles is at the optimal value of the end vertex of the path.

Two major problems immediately surface: there are exponentially many paths and cycles, but secondly and a more basic problem is that optimal values are not known and are to be solved for. The second problem is solved by the realization that to compare two  $u$ - $v$  paths  $p_1$  and  $p_2$ , locating the intersection point  $\lambda$  of the two linear functions with respect to the optimal value  $x^*$  of  $v$  suffices. For example, if  $\lambda \geq x^*$  then the function that dominates on the left side of  $\lambda$  is at least as good as the other at the optimal value. The relative position of  $\lambda$  with respect to  $x^*$  is obtained by querying for value of  $v$  with  $\lambda$ , using either of the algorithms Forward-Propagate or Backward-Propagate. The proof of correctness under Backward-Propagate is based on the observation that for some vertices, their optimal value is equal to their value under an optimal cycle, and, for these vertices, correct path comparisons are made.

The problem of an exponential number of paths is solved by using dynamic programming and using a “path doubling” speed-up technique, *i.e.*, for any ordered pair of vertices, first a best edge (if any edge exists) is obtained, and the information is used to compute best path among paths of length not more than 2 between any two vertices, and, in general, from information on a best path from paths of lengths not greater than  $k$ , the algorithm obtains best paths of length no greater than  $2k$ . We will next describe how this is achieved in more detail and give time bounds.

When a best path of length  $k$  or less has been found for any two vertices, a highest valued  $u$ - $v$  path of length not greater than  $2k$  is obtained as follows: for each possible third vertex  $s$ , the best  $u$ - $s$  path of length  $k$  or less is composed with a best  $s$ - $v$  path of length  $k$  or less. These newly created paths, of which there are no more than  $n$  many, are then compared to find the best path of length not greater than  $2k$ . It can be shown inductively that the procedure finds best optimal cycles for all vertices in  $n$  iterations.

As there are  $n^2$  ordered pairs of vertices, and each comparison requires a relatively ex-

pensive  $O(mn)$  query time, median finding can be used to save on the number of queries [87, 20]: the algorithm repeatedly pairs paths to compare, finds the median of the intersection points of the paired paths and queries the median. In this way, with one query,  $1/4$  of the paths are removed from consideration, and since there are  $O(n)$  paths to compare,  $O(\log n)$  median findings and queries are needed, each taking  $O(n + mn)$  time, which translates to  $O(mn \log n)$  time for finding a best  $u$ - $v$  path of a given length  $2k$ , resulting in  $O(mn^3 \log n)$  time for finding all  $n^2$  best paths. Since the path-doubling loop repeats for  $O(\log n)$  time, for each vertex finding its best cycle, takes  $\tilde{O}(mn^3)$  time. An extra time saving technique given in [20] further saves time on the median findings to reduce the running time to  $\tilde{O}(mn^2)$ .

#### 5.2.4 Difficulties with Extending to MDP(3)

A question that arises immediately is whether these (TVPI) binary search methods generalize to MDP(3). At the core of all the algorithms is of course the solution to the query problem 5.2.1. The two value propagation algorithms which solve the query problem basically depend on the property that a value propagated is “supported” by a cycle if and only if the value is less than the value under the cycle: in case of Backward-Propagate, the value propagated along the cycle, increases when it makes it back to the starting vertex, and in case of Forward-Propagate, it decreases, both implying that the value of the vertex under the cycle is higher.

In case of MDP(3), there are no cycles. Forward propagation is not well defined, as there can be many combinations of values of the neighboring vertices (defining a linear function) that can satisfy a vertex request. A symbolic representation of combination of values that can satisfy a request may be used, for example, we may begin by a line representation for the first request. But it appears that the symbolic representation grows exponentially with each request and no technique of compactly representing and efficiently computing with such structures appears to us. The Backward-Propagate is well-defined if we initialize all vertices with some initial values, but it does not work, for example, when a vertex in question is initialized with the queried value and other vertices are initialized with lower

bounds, unlike in MDP(2) problems. This is because the queried value, when propagated, can dissipate or diminish if other initial values are too small.

In summary, exhibiting an efficient simple query procedure that allows binary search for MDP(3), just as the case is one for MDP(2), would be very useful and is an interesting open problem.

### 5.3 Analysis of Policy Iteration

We begin by considering a parametric analysis of the MDP(2) problem which we link to the progress of policy iteration. We also simplify by analyzing policy iteration on problems where all cycles share a single vertex. The analyses lead to polynomial time policy improvement algorithms for MDP(2).

#### 5.3.1 A Parametric Analysis of MDP(2)

For any MDP(2) problem with vertices  $u$  and  $v$ , not necessarily distinct, let  $C_{u,v}$  denote the optimal value of vertex  $u$  in the MDP(2) problem with all in-edges to  $v$  deleted (if all paths from  $u$  involve  $v$  define  $C_{u,v} = -\infty$ ). Thus  $C_{u,v}$  is the highest value of  $u$  using the best policy that doesn't involve vertex  $v$ . Define  $F_{u,v}(x)$  as follows:

$$F_{u,v}(x) = \max_{u-v \text{ paths } p} (f_p(x), C_{u,v}),$$

where maximization is taken over all possible  $u$ - $v$  paths. Note that at any value  $x$  of vertex  $v$ , the highest value vertex  $u$  can have is either  $C_{u,v}$ , or a path exists from  $u$  to  $v$  which gives the highest value to  $u$  over all possible walks. This is because if a walk with a cycle has a better value than any path, then it follows that a cycle in the walk has a better value, contradicting the assumption on  $C_{u,v}$ . Thus  $F_{u,v}(x)$  is the optimal value vertex  $u$  can have as a function of value  $x$  of a vertex  $v$ , where we assume  $v$  is no longer a choice vertex, but has a fixed value  $x$  ( $v$  with fixed value  $x$  can be modeled with a self edge with  $y_e = x$ , and  $m_e = 0$ ). We will investigate the changes in  $F_{u,v}$  as  $x$  is changed.

**Lemma 5.3.1** *Consider any MDP(2) problem.  $F_{v,s}(x)$ , for any  $u, v \in V$  is a piecewise*

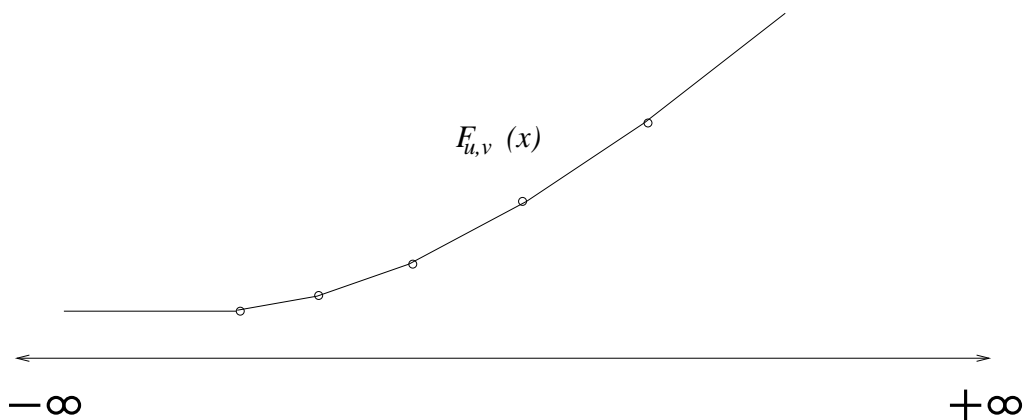


Figure 5.3:  $F_{u,v}(\lambda)$  is convex, piecewise linear, and nondecreasing. The breakpoints (points of slope change) are circled.

*linear nondecreasing convex function, with number of slope changes no more than an exponential function in  $n$  and  $m$ .*

**Proof.** Consider what happens to  $F_{u,v}$  as we increase  $x$  from low values to high values. In general,  $F_{u,v}$  is a constant for low values of  $x$ , in which case  $u$  has a path to a cycle not containing  $v$ , so we would have  $F_{v,s}(x) = C_{v,s}$  or it has a path to  $v$  with slope zero. As  $x$  is increased, at some point it may become an increasing function if a value function corresponding to a path to  $v$  becomes higher valued than  $C_{u,v}$ . Path value functions are linear, and each time the value function corresponding to another path becomes higher valued, a *breakpoint* (change of slope) occurs and  $F_{u,v}$  gets a linear segment with higher slope. It follows that that  $F_{u,v}$  is a piecewise linear, nondecreasing, convex function of  $x$  (Fig. 5.3), and each segment, except the first, corresponds to a different simple  $u$ - $v$  path of which there can be exponentially many at most.  $\square$

Let  $B(n, m)$  denote the maximum number of slope changes, or *breakpoints*, in  $F_{u,v}$  over all graphs of  $n$  vertices and  $m$  edges and possible pair of vertices  $u$  and  $v$ . We show  $B(n, m) = mn^{O(\log n)}$  by identifying a property that constrains the sequence of  $u$ - $v$  paths that correspond to the sequence of break points of  $F_{u,v}$ , as  $x$  is increased.

**Definition 5** (*Path Reappearance Constraint*) Consider a directed graph  $G$ , and a sequence

of  $u$ - $v$  paths  $\{p^{(i)}\}$ , where each path  $p^{(i)}$  is a possible simple  $u$ - $v$  path ( $u$  and  $v$  not necessarily distinct) in  $G$ . We say the path sequence  $\{p^{(i)}\}$  satisfies the path-reappearance constraint, if  $p^{(j)} \neq p^{(k)}$  when  $j \neq k$ , and for any two paths  $p^{(j)}$  and  $p^{(k)}$ , where  $k > j$ , if for any two vertices  $w$  and  $w'$ ,  $w$ - $w'$  path  $q$  appears in  $p^{(j)}$  and  $w$ - $w'$  path  $q'$  appears in  $p^{(k)}$ , with  $q \neq q'$ , then for any path  $p^{(i)}$ ,  $i \geq k$ ,  $q$  does not appear in path  $p^{(i)}$ .

**Lemma 5.3.2** *Let  $G = (V, E)$  be a directed graph with  $n$  vertices, and  $m$  edges. Any sequence of  $u$ - $v$  paths that satisfies the path-reappearance constraint has no more than  $mn^{(2 \log n)}$  paths.*

**Proof.** We show inductively that, between any ordered pair of vertices, there can be at most  $m(2n)^k$  paths of length not more than  $2^k$ ,  $k \geq 0$ , in the sequence.

There can be at most  $m$  paths of length 1 from one vertex to another in the sequence due to the reappearance constraint. Let us assume there are at most  $m(2n)^{k-1}$  paths of length  $2^{k-1}$ , for some  $k \geq 1$ , for any ordered pair of vertices. A path of length  $2^k$  or less is composed of two paths of length  $2^{k-1}$  or less, and there are  $n$  possibilities for the middle vertex. For each such middle vertex  $w$ , there can appear  $m(2n)^{k-1}$  paths of length  $2^{k-1}$  by the induction hypothesis. Order the  $u$ - $s$  paths and  $s$ - $v$  paths, each having length at most  $2^{k-1}$ , in the order that they appear as a subpath of a  $u$ - $v$  path with  $s$  in the middle. Let there be  $k_1$   $u$ - $s$  paths and  $k_2$   $s$ - $v$  paths. Then it is not hard to see that there can be at most  $k_1 + k_2 - 1$  possible composed paths: Consider the first  $u$ - $s$  path and the first  $s$ - $v$  path. At least one of the two appears exactly once due to the path-reappearance constraint, and we can use this argument in an induction on  $k_1 + k_2$ . As  $k_1 \leq m(2n)^{k-1}$  and  $k_2 \leq m(2n)^{k-1}$ , and sequence of  $u$ - $v$  paths having  $s$  in the middle is a subset of such a matching, we get at most  $m(2n)^{k-1} + m(2n)^{k-1} - 1 < 2m(2n)^{k-1}$  such paths. Thus we obtain a total of  $(2n)m(2n)^{k-1}$  from all the  $n$  possible middle vertices. This gives a bound of  $m(2n)^{1+\log n} = mn^{O(\log n)}$  for paths of all possible length.  $\square$

**Theorem 5.3.3** *For  $MDP(2)$ ,  $B(n, m) = mn^{O(\log n)}$ .*

**Proof.** Note that each linear segment of  $F_{u,v}$ , except possibly the first, is increasing (this is when  $F_{u,v}(x) > C_{u,v}$ ). An increasing linear segment of  $F_{u,v}$ , at a point  $x$ , implies that

there exists a simple  $u$ - $v$  path  $p$ , with the following property:  $F_{u,v}(x) = f_p(x)$ , and each  $s$ - $s'$  subpath of path  $p$  is a highest valued  $s$ - $s'$  path for the value of  $s'$ , otherwise we could replace such a path with a better one, and whether a cycle occurs, or otherwise, we contradict the assumption on  $F_{u,v}$ .

Consider what happens as we increase  $x$ . If a  $s$ - $s'$  path  $p_1$  is highest valued, and later, at a higher  $x$ , another  $s$ - $s'$  path  $p_2$  becomes highest valued,  $p_1$  is never highest valued again, as the value of  $s'$  is an increasing function of the value of  $v$ . Thus the sequence of different optimal  $u$ - $v$  paths seen corresponding to the increasing line segments in  $F_{v,s}$  has the path reappearance constraint. Thus, from Lemma 5.3.2 it follows that  $B(n, m) = mn^{O(\log n)}$ .  $\square$

One might expect that the path reappearance constraint gives a better bound than the super-polynomial one given above. However, the same reappearance constraint appears in a parametric version of the single source shortest path problem which we describe next, and the work of Carstensen in [16, 15] shows a matching lower bound of  $mn^{\Omega(\log n)}$  for that problem. However, Carstensen [16] also shows that the number of break points is polynomial in  $n$  and the magnitude of the largest number used. A translation from the number of breakpoints in the parametric shortest path problem to an MDP(2) problem that preserves the asymptotics should be feasible, but we haven't attempted it.

The parametric shortest path problem considered by Carstensen [16, 15] (see also [1], exercise 5.54) is as follows: Consider a directed acyclic graph where all paths run from a vertex  $u$  to a vertex  $v$  and each edge  $e$  is weighted by  $w_e + \alpha_e x$ , where  $x$  is the parameter that changes, and  $w_e$  and  $\alpha_e$  are characteristics of edge  $e$  (*i.e.* they can differ among different edges). It is shown that as  $x$  increases, there can be on the order of  $mn^{\Omega(\log n)}$  changes in the shortest path. It is not hard to see that path reappearance holds here too: For any shortest path from vertex  $u$  to  $v$ , any subpath from  $s$  to  $s'$  must be a shortest  $s$ - $s'$  path and when one path between any two vertices becomes better than another as  $x$  is changed monotonically, it remains better.

### *Deterministic MDPs*

In the special case of the deterministic (discounted) *MDP* problem,  $F_{u,v}$  does have a polynomial and in fact a linear number of breakpoints. Recall from Section 4.4.2 that the deterministic problem is a special *MDP(2)* problem where all edges have the same slope  $\beta$ . In this case, the different functions  $f_p$  and  $f_q$  for two  $u$ - $v$  paths  $p$  and  $q$  of equal length  $k$  have equal slope  $\beta^k$ , thus the two functions are parallel and one dominates the other at all points. It follows that for each  $1 \leq k \leq n$ , there is at most one highest valued  $u$ - $v$  path of length  $k$  for all values of  $v$ . Value functions corresponding to shorter paths have higher slope, thus later linear segments in  $F_{(u,v)}$  correspond to decreasing  $u$ - $v$  path lengths. The following lemma holds:

**Lemma 5.3.4** *Consider the *MDP(2)* problem where all edges have equal slope. Then  $F_{u,v}$  for any pair of vertices  $u$  and  $v$  has no more than  $n$  break points.*

### *5.3.2 Analysis of Policy Iteration on Adags Using the Parametric Bounds*

Consider an *MDP(2)* problem where all paths lead to a vertex  $s$ , and vertex  $s$  has a single  $s$ - $s$  edge  $e$  only. Thus the graph is basically acyclic. Then policy iteration would take one iteration to find the value of  $s$ , and it is not hard to see that each vertex having an optimal path of length  $i$  to  $s$  would find its best edge in  $i$  or fewer iterations. Therefore in  $O(n)$  iterations, policy iteration would find optimal edges and values for all vertices.

In this section we look at a more general graph where the restriction that  $s$  have only a single edge choice back to itself is removed, but all paths go through a *bottleneck* vertex  $s$ , or all cycles share vertex  $s$ . We will refer to such a graph as an *adag* (for almost dag). Fig. 5.4a shows an adag, with the bottleneck vertex  $s$  duplicated to better see the structure. The fact that all paths end in  $s$  makes analysis relatively easy. Our analysis in this section also motivates the fast convergence analysis of the next section. In Section 5.3.5 we show that when the edge choice of a vertex is fixed (frozen) and the resulting problem with effectively one less vertex, is solved (recursively), what remains is a problem similar to the adag case, and our analyses apply with not much change, yielding polynomial *MDP(2)* algorithms.

As policy iteration runs on an adag, the value of each vertex, including the bottleneck  $s$

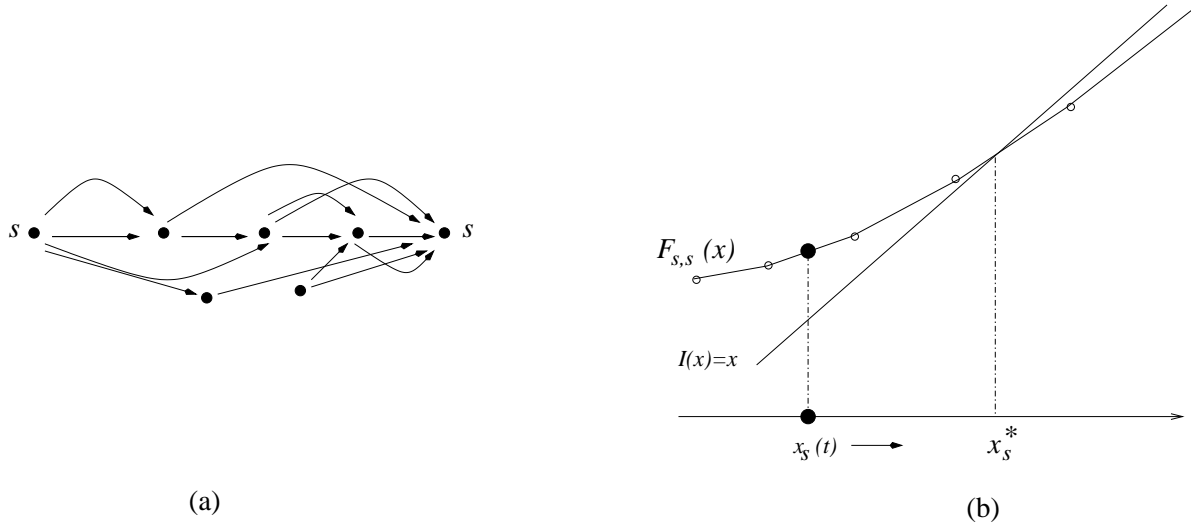


Figure 5.4: (a) An example adag with the bottleneck vertex  $s$  duplicated (one with only in-edges and the other with out-edges). (b) Progress of  $x_s(t)$  on the adag.

increases from time to time. In fact, it is not hard to see that if the value of  $s$  doesn't change in  $n$  iterations, then policy iteration has found an optimal policy. Consider the progress of the value  $x$  of  $s$  and its image  $F_{s,s}(x)$  (the value of the highest valued  $s$ - $s$  path). We show that either the optimal policy is found, or the image  $F_{s,s}(x)$  surpasses at least one point of  $F_{s,s}$  in  $O(n)$  iterations, by showing that the highest valued  $s$ - $s$  path is found in that time. As there are  $B(n, m) = mn^{\log n}$ , this gives a quasi-polynomial bound on the run-time of policy iteration.

We use the sequence notation introduced in the preliminaries Section 4.1 to refer to values at a given time during policy iteration.

**Lemma 5.3.5** *Consider policy iteration on any MDP(2) problem. Assume a vertex  $u$  has path  $p$  to a vertex  $v$  at time  $t, t \geq 2$  ( $u = v$  is a possibility). Then  $f_p(x_v^{(t-1)}) > x_u^{(t-1)}$  if  $p$  did not exist at time  $t - 1$ .*

**Proof.** The proof is by induction on  $|p|$ . Vertex  $u$  or some vertex on path  $p$  besides  $v$  has switched in iteration  $t$ . If  $|p| = 1$ , the inequality follows from the definition of switching to an edge. For  $|p| \geq 2$ , let vertex  $w$  be the vertex appearing after  $u$  on path  $p$ , and let  $p = eq$ ,

where  $e$  is the  $u$ - $w$  edge on  $p$  and  $q$  is the rest of the path to  $v$ . Then we have  $x_u^{(t-1)} \leq f_e(x_w^{(t-1)})$  and  $x_w^{(t-1)} \leq f_q(x_v^{(t-1)})$ , both inequalities by induction, and at least one is strict due to the assumption of a switching vertex on  $p$ . Thus  $x_u^{(t-1)} < f_e(f_q(x_v^{(t-1)})) = f_p(x_v^{(t-1)})$ .  $\square$

**Corollary 5.3.6** *Assume a vertex  $u$  has path  $p$  to a vertex  $v$  at time  $t \geq 1$ , and at time  $t + 1$ , assume  $u$  has path  $q \neq p$  to  $v$ . Then  $f_q(x_v^{(t)}) > f_p(x_v^{(t)})$ .*

**Proof.** We have  $x_u^{(t)} = f_p(x_v^{(t)})$ , and from Lemma 5.3.5 we have  $x_u^{(t)} < f_q(x_v^{(t)})$ , as either  $u$  or a vertex on  $q$  must have switched.  $\square$

**Lemma 5.3.7** *Consider an adag with bottleneck  $s$ . Let  $p_0$  be the highest valued path from vertex  $v$  to  $s$  at value  $x_s^{(t_0)}$  for time  $t_0$ , and let  $k = |p_0|$ . If there are multiple such paths, assume  $p_0$  has the smallest slope, and if ties remain, pick one with the smallest length. We say a  $v$ - $s$  path  $p$  is “dominating” at time  $t$  if  $f_p(x_s^{(t)}) \geq f_{p_0}(x_s^{(t)})$  and we say  $x_v^{(t)}$  is “dominating” if  $x_v^{(t)} \geq f_{p_0}(x_s^{(t)})$ . We have*

1. *At any time  $t \geq t_0$ , if a  $v$ - $s$  path  $p$  is dominating (resp.  $x_v^{(t)}$  is dominating), then for any time  $t' \geq t$ ,  $p$  remains dominating (resp.  $x_v^{(t')}$  remains dominating).*
2. *For some time point  $t \leq t_0 + k$ ,  $x_v^{(t)} \geq f_{p_0}(x_s^{(t)})$ .*

**Proof.** (1) For any  $v$ - $s$  path  $p$ , if  $f_p(x_s^{(t)}) \geq f_{p_0}(x_s^{(t)})$ , then  $f_p(x) \geq f_{p_0}(x)$ , for  $x \geq x_s^{(t)}$ , because  $p_0$  is the highest valued path at  $x_s^{(t_0)}$  with lowest possible slope if there are ties, and  $x_s^{(t)} \geq x_s^{(t_0)}$ , for  $t \geq t_0$ , and the value functions are linear. Thus once a path is dominating it remains dominating.

(vertex value domination) Assume at some iteration  $t$ ,  $x_v^{(t)}$  is dominating, and assume  $v$  has path  $p$  to  $s$  at time  $t$ , thus  $f_p(x_s^{(t)}) = x_v^{(t)} \geq f_{p_0}(x_s^{(t)})$ . Thus if path  $p$  remains at end of iteration  $t + 1$ , the inequality holds for  $t + 1$ , by path domination. Otherwise, assume path  $p$  changes to  $p'$ . By Lemma 5.3.6 we must have  $f_{p'}(x_s^{(t)}) > f_p(x_s^{(t)}) \geq f_{p_0}(x_s^{(t)})$ , and path domination applies.

(2) We establish this by induction on  $|p_0| = k \leq n$ . Assume  $|p_0| = 1$  and consider what happens in iteration  $t_0$ . If  $v$  does not switch, it must be that its value before the iteration

ties  $f_{p_0}(x_s^{(t_0)})$ , because  $p_0$  is a single edge long. Vertex domination applies. If  $v$  switches to  $p_0$  we are done, otherwise let  $p$  be the path from  $v$  to  $s$  at the end of iteration  $t_0$ . If  $v$  switches to form path  $p$ , since we assume  $v$  switches to the highest valued edge, we must have  $f_p(x_s^{(t_0)}) \geq f_{p_0}(x_s^{(t_0)})$ .

Proof in the inductive step is similar to the basis: Assume  $|p_0| = k \geq 2$ , and assume  $v$  is adjacent to  $u$  which has value  $x_u^{(t)} \geq q_0(x_s^{(t)})$  at iteration  $t \leq k - 1$ , where  $q_0$  denotes the highest valued path from  $w$  to  $s$  at  $x_s^{(t_0)}$ , and  $p_0 = eq_0$ , *i.e.* the highest valued path  $p_0$  from  $v$  to  $s$  begins with  $e$  and then follows  $q_0$  to end in  $s$ . If  $v$  does not switch, and is using a path  $p$  to  $s$  at iteration  $t$ , it must be that  $f_p(x_s^{(t)}) \geq f_e(x_u^{(t)}) \geq f_{p_0}(x_s^{(t)})$ , and  $x_v^{(t)}$  is already dominating. If  $v$  switches to  $e$ , then after iteration  $t + 1$ , its value is  $f_e(x_u^{(t+1)}) \geq f_e(f_{q_0}(x_s^{(t+1)}))$  because we assumed  $u$  starts the iteration with a dominant value, and  $f_e(f_{q_0}(x_s^{(t+1)})) = f_{p_0}(x_s^{(t+1)})$ , and we are done. Otherwise, if  $v$  switches, but not to  $e$  and creates a path  $p$  to  $s$ , then we must have  $f_p(x_s^{(t)}) \geq f_e(x_v^{(t)})$ , otherwise policy iteration would pick edge  $e$ . Thus  $f_p(x_s^{(t)}) \geq f_e(f_{q_0}(x_s^{(t)})) = f_{p_0}(x_s^{(t)})$ .  $\square$

Note that for length  $k$  of  $p_0$  in the previous lemma,  $k \leq n$ , and  $k$  may equal  $n$  only when  $p_0$  is an  $s$ - $s$  path.

**Corollary 5.3.8** *Let  $p_0$  be the highest valued  $s$ - $s$  path at time  $t_0$ , and let  $y$  be the fixed-point of  $f_{p_0}$ , *i.e.*  $f_{p_0}(y) = y$ . Then, for some  $k \leq |p_0| \leq n$ ,  $x_s^{(t_0+k)} \geq y$ .*

Thus the image  $F_{s,s}(x)$  moves beyond the next break point in  $F_{s,s}$  in  $n + 1$  iterations (Fig. 5.4.(b)), unless the interval contains the optimal cycle.

**Theorem 5.3.9** *Policy iteration on adag MDPs has run time  $m^2 n^{O(\log n)}$ , and on deterministic adags, it has a run time of  $O(mn^2)$ .*

**Proof.** It takes  $O(n)$  to move the value of  $s$  and each iteration takes  $O(m + n)$ . We thus obtain  $O(mnB(n, m))$  for policy iteration on adags, yielding  $m^2 n^{O(\log n)}$  and  $mn^2$  for deterministic adags.  $\square$

The run times, especially on deterministic adags, is probably improvable by making amortized arguments. We note that similar results hold for the scan algorithm on adags.

1.  $i \leftarrow 0$ , and let  $x^{(0)}$  be any point in  $(-\infty, +\infty)$ .
2. Repeat
3. Choose  $l$  such that  $l(x^{(i)})$  is highest among the functions.
4. Solve for the fixed-point  $x_l$  of  $l$ , and let  $x^{(i+1)} \leftarrow x_l$ .
5.  $i \leftarrow i + 1$ .

Figure 5.5: The *Newton* process for finding  $x^*$ .

The fact that policy iteration finds, in  $O(n)$  iterations, a highest valued cycle or one which has a higher fixed-point, and computes and moves beyond the fixed-point, motivates the following convergence analysis.

### 5.3.3 Policy Iteration as Newton's Method

Assume we are given a finite but large collection  $\mathcal{L}$  of nondecreasing linear functions, each with a positive slope less than 1. Thus each function  $l \in \mathcal{L}$  intersects the identity function  $I(x) = x$  at a unique *fixed-point*, denoted by  $x_l$ . Let  $x^*$  denote the largest fixed-point value. We assume we have access to an oracle such that at any point  $x \in (-\infty, +\infty)$ , the oracle gives a function  $l$  from the collection with highest value  $l(x)$ . The procedure *Newton*, shown in Fig 5.3.3, uses calls to this oracle (line 3) in order to get closer to  $x^*$ . It is not hard to see that the iterates in the procedure converge to  $x^*$  from the left side.

There is a connection between the Newton process and the policy iteration algorithm on adags with bottleneck  $s$  as described in the previous section: The value functions corresponding to  $s$ - $s$  paths are the linear functions, of which there can be exponentially many, and the values of vertex  $s$  at time intervals of  $n$  iterations apart, equal or surpass the iterates  $x^{(i)}$ . On the other hand, the process is basically that of picking the tangent-line of the upper-envelope  $F$  of the linear functions ( $F_{s,s}$  in the adag case), which is a piecewise linear nondecreasing convex function guaranteed to intersect the identity function. The point of intersection of the tangent line defines the new iterate. See Fig. 5.6.(a). This is akin to Newton's method for finding the zero of a function.

Consider the sequence of the iterates  $\{x^{(i)}\}$  of the Newton process, which we will refer to as a *Newton sequence*. Newton's method has been shown to have a local rate of convergence at least as fast as quadratic convergence on smooth functions. In our case, this is not hard to see as when close enough to  $x^*$ , the function  $F$  is linear, and one more iteration gives  $x^*$ . However, we need to establish fast global convergence to establish polynomial run time, and a difficulty in showing this in our case is that the distances of consecutive iterates to  $x^*$  may not reduce by a significant factor at various time points in the iterate sequence. Examples exist, at least initially in the sequence, where a relatively small reduction in distance is the case. In such examples, another desirable property that inter-iterate distances increase by relatively large factors, also does not occur.

We present two arguments that establish polynomial convergence. The first is developed by us and shows that the distances of the iterates to the optimal are bounded by an exponentially diminishing upper bound. The second uses an argument by Radzik [107] which shows that one of two measures, to be described below, are reduced geometrically from one iterate to another. Because there exist both upper bounds and lower bounds on the two measures, after a logarithmic number of iterations, the argument establishes polynomial convergence to the optimal. The second argument has the advantage that it is simpler. Radzik [107] also extends the argument to show strongly polynomial bounds for several fractional problems, and we expect similar arguments would show strongly polynomial bounds for the MDP(2) (and TVPI) as well. However, we do not establish such a result in this thesis.

#### *Analysis Based on Deriving an Upper Bound on Distance to Optimal*

Recall that  $f^*(x)$  is a function in the collection  $\mathcal{L}$  with highest fixed point  $x^*$ , and  $F(x)$  is the upper envelope of the functions:

$$F(x) = \max_{l \in \mathcal{L}} l(x).$$

We will assume that the initial iterate  $x^{(0)} \leq x^*$ , which is no loss of generality, as starting at any point, because each future iterate is the fixed point of some function, the next iterate is not greater than  $x^*$ . For the  $i$ th iterate  $x^{(i)}$  of the Newton sequence, we show its distance  $\frac{x^* - x^{(i)}}{x^* - x^{(0)}}$  is less than a diminishing function of  $\frac{x^* - x^{(0)}}{f^*(x^{(0)}) - x^{(0)}}$ . We simplify by assuming  $x^{(0)} = 0$ ,

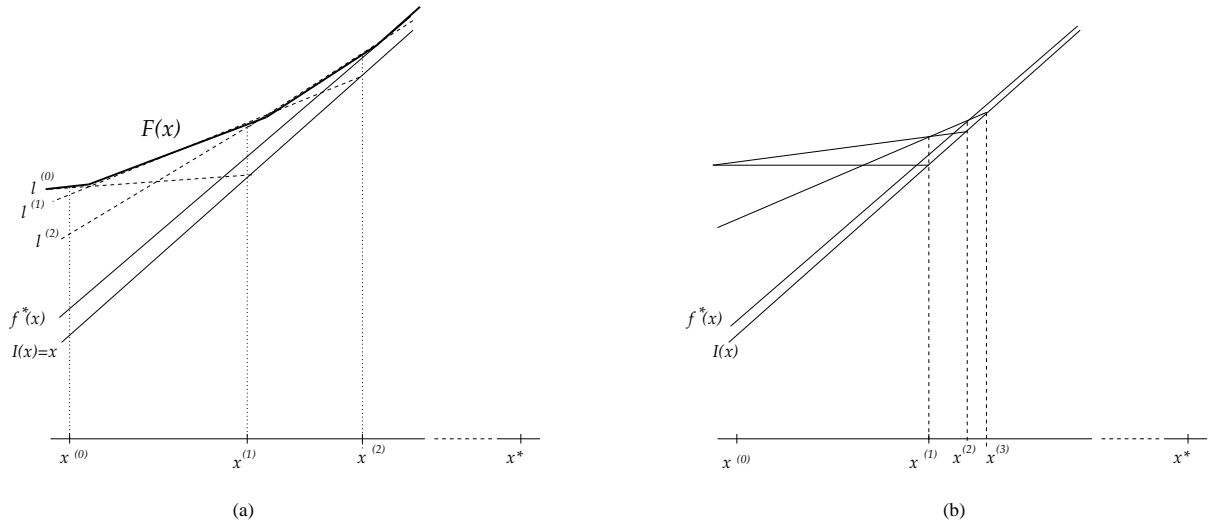


Figure 5.6: (a) The progress of the Newton process in finding  $x^*$ . (b) Among individual iterates, here between  $x^{(1)}$ ,  $x^{(2)}$ , and  $x^{(3)}$ , there may only be relatively small change in distance.

by shifting the origin to  $(x^{(0)}, x^{(0)})$ , and  $x^* = 1.0$ , by normalizing. We now want to bound the distance of the  $i$ th iterate, which is now  $1 - x^{(i)}$ . We let

$$y = f^*(0), \text{ and } h = F(0).$$

Note that  $F(0)$  is the value of the first function picked by the Newton process at  $x^{(0)} = 0$ . We thus have  $0 < y \leq h \leq 1$ , and  $f^*(x) = (1 - y)x + y$ , so  $f^*$  has slope  $1 - y$ .

Our argument is structured in the following way. We show that any Newton sequence “dominates” a corresponding *tight* (Newton) sequence to be defined, in the sense that its  $i$ th iterate is at least as close to the optimal as the  $i$ th iterate of its corresponding tight sequence. We then establish a diminishing distance bound on the iterate distances of tight sequences. A tight sequence is generated by three parameters:  $y = f^*(0) \in (0, 1]$ ,  $h = F(0) \in [y, 1]$ , and a sequence of numbers in  $[0, 1]$  to be defined.

In what follows, it will be convenient to consider the sequence to be an infinite one with the iterates converging only in the limit to 1.0. This possibility has the advantage of showing that the analysis is applicable to slightly different scenarios such as  $\mathcal{L}$  being infinite

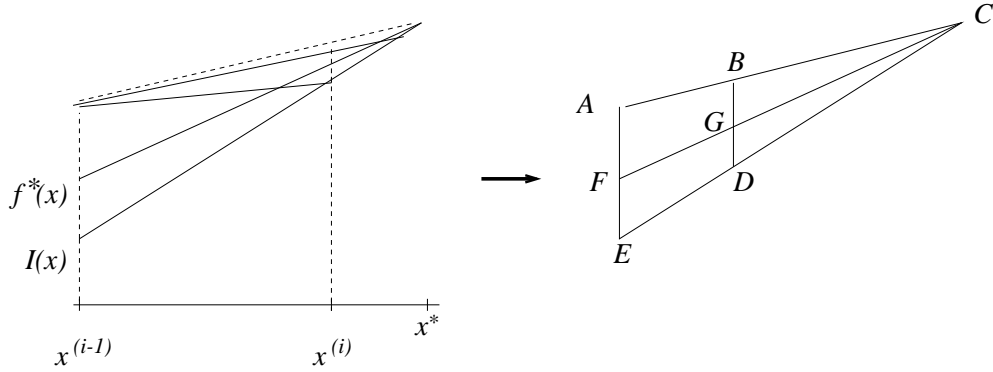


Figure 5.7: That  $r^{(i)} \leq r^{(i-1)}$  can be seen geometrically, as in the triangle  $ACE$ , lines segments  $AE$  and  $BD$  are parallel, so  $\frac{|BG|}{|AF|} = \frac{|CG|}{|CF|} = \frac{|GD|}{|FE|}$ , thus  $\frac{|BG|}{|GD|} = \frac{|AF|}{|FE|}$  ( $|BG|$  denotes the length of segment  $BG$ ). This corresponds to the case  $r^{(i)} = r^{(i-1)}$ , and otherwise we have  $\frac{|BG|}{|GD|} < \frac{|AF|}{|FE|}$ .

or  $F$  being any continuous nondecreasing convex function, with appropriate assumptions<sup>1</sup>. The following analysis applies whether or not the Newton sequence converges to 1 only in the limit.

We now describe a few constraints that Newton sequences satisfy, and from these we characterize tight sequences. Let *distance ratios*  $r^{(i)}$ , when  $x^{(i)} < 1.0$ , be defined as

$$r^{(i)} = \frac{F(x^{(i)}) - f^*(x^{(i)})}{f^*(x^{(i)}) - x^{(i)}}.$$

**Lemma 5.3.10** *For  $i \geq 1$ , such that  $x^{(i)} < 1.0$ ,  $r^{(i)} \leq r^{(i-1)}$ . Furthermore, if  $r^{(i)} = r^{(i-1)}$ , then  $x^{(i+1)} = 1.0$ .*

**Proof.** That  $r^{(i)} \leq r^{(i-1)}$ , and if  $r^{(i)} = r^{(i-1)}$ , then  $x^{(i+1)} = 1.0$  follows using a geometric argument based on the similarity (congruence) of triangles. Fig 5.7 explains.  $\square$

Thus, for any Newton sequence, the parameters  $y$ ,  $h$ , and  $\{r^{(i)}\}$  are well defined, and satisfy constraints such as  $r^{(0)} = \frac{h}{y}$  and  $r^{(i)} \leq r^{(i-1)}$ . However, multiple Newton sequences

<sup>1</sup>In case of  $\mathcal{L}$  being infinite,  $x^*$ , the supremum of the fixed points, should exist. For the quantitative bounds, we assume a limiting function exists that has fixed point  $x^*$  and slope less than 1. Similar properties need to apply to  $F$ .

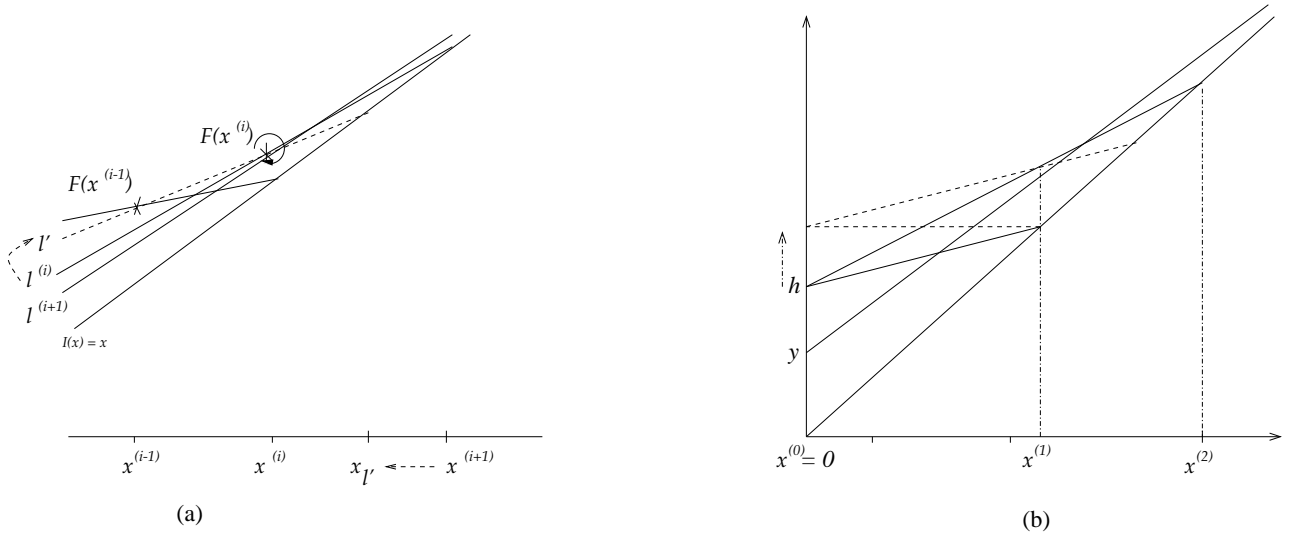


Figure 5.8: (a) Pivoting  $l^{(i)}$  around  $(x^{(i)}, F(x^{(i)}))$  to obtain  $l'$  reduces  $x^{(i+1)}$  to  $x_{l'}$ . (b) Setting  $m^{(0)}$  to zero by moving  $h$  up if necessary yields a tight sequence.

can have the same parameters which stems from the following source of freedom of choice: In each iteration  $i \geq 1$ , the Newton process picks a highest valued line (a linear function)  $l^{(i-1)}$  at  $x^{(i-1)}$ , that is  $l^{(i-1)}(x^{(i-1)}) = F(x^{(i-1)})$ , and the line picked *generates* the  $i$ th iterate, which is its fixed point. We have line  $l^{(i)}$ ,  $i \geq 1$ , going through the points  $(x^{(i)}, l^{(i)}(x^{(i)}))$ , and  $(x^{(i-1)}, l^{(i)}(x^{(i-1)}))$ , where  $l^{(i)}(x^{(i)}) = F(x^{(i)})$  and  $l^{(i)}(x^{(i-1)}) \leq F(x^{(i-1)})$ . Thus the value of  $l^{(i)}$  at  $x^{(i)}$  is determined by  $F(x^{(i)})$ , which in turn is determined by the  $f^*(x^{(i)})$ , and  $r^{(i)}$ , but  $l^{(i)}(x^{(i-1)})$  is only constrained not to be greater than  $F(x^{(i-1)})$ . We next show that by tightening the second inequality, *i.e.* having  $l^{(i)}(x^{(i-1)}) = F(x^{(i-1)})$ , we may only increase the iterate distances, and thus it would be sufficient to consider only such tight sequences in our analysis.

**Lemma 5.3.11** *Consider any Newton sequence  $\{x^{(i)}\}$ . If for some iterate  $x^{(i+1)}$ ,  $i \geq 1$ , we have  $l^{(i)}(x^{(i-1)}) < F(x^{(i-1)})$ , then function  $l'(x)$  going through  $(x^{(i)}, F(x^{(i)}))$  and  $(x^{(i-1)}, F(x^{(i-1)}))$  has a fixed point  $x_{l'}$  smaller than  $x^{(i+1)}$ .*

*If  $x^{(i+1)}$ , is the last iterate of the sequence, we have increased its distance to 1.0. Otherwise, substituting  $x_{l'}$  for  $x^{(i+1)}$ , and leaving the definition of  $l^{(i+1)}$ , so that  $F(x_{l'}) =$*

$l^{(i+1)}(x_{l'})$ , does not affect the position of  $x^{(i+2)}$  or later iterates.

**Proof.** This basically follows from geometry of the lines in the plane. See Figure 5.8.(a). We show  $l^{(i)}(x_{l'}) > l'(x_{l'}) = x_{l'}$ , from which it follows that the fixed point  $x_l$  for  $l$  is greater than that of  $l'$ , as both  $l$  and  $l'$  have slope less than zero. We have  $l'(x^{(i)}) = l(x^{(i)})$ , but  $l'(x^{(i-1)}) > l(x^{(i-1)})$ , and  $x^{(i-1)} < x^{(i)}$ , it follows that  $l(x) > l'(x)$ , for  $x > x^{(i)}$ , and in particular  $l(x_{l'}) > l'(x_{l'}) = x_{l'}$ .

If  $x^{(i+1)}$  is not the last iterate of the sequence, substituting  $l'$  for  $l^{(i)}$  and  $x_{l'}$  for  $x^{(i+1)}$ , need not affect the function  $l^{(i+1)}$ , as we can define  $F(x_{l'}) = l^{(i+1)}(x_{l'})$ , and we already have  $l'(x^{(i)}) = l^{(i)}(x^{(i)}) \geq l^{(i+1)}(x^{(i)})$ . Similarly no change in definition of  $l^{(i+2)}$  is needed. It follows that the magnitude of future iterates need not change.  $\square$

It follows that we need only consider sequences where  $l^{(i)}$  goes through the points  $(x^{(i)}, F(x^{(i)}))$  and  $(x^{(i-1)}, F(x^{(i-1)}))$  in generating the next iterate  $x^{(i+1)}$ . We remove the remaining freedom which is choosing the slope of  $l^{(0)}$ , which we set to 0 in a tight sequence, as otherwise we could increase  $h = F(0)$ , as in Fig. 5.8.(b), so that  $h = x^{(1)}$ . Now  $l^{(1)}(0) < h$ , that is the sequence is not tight, but we can make it tight again by modifying  $l^{(1)}$  so that  $l^{(1)}(0) = h$ , as above, which lowers  $x^{(2)}$ , without affecting later iterates. We call a Newton sequence where  $l^{(0)}$  has slope 0, and  $l^{(i)}$  goes through the points  $(x^{(i)}, F(x^{(i)}))$  and  $(x^{(i-1)}, F(x^{(i-1)}))$  a tight (Newton) sequence.

In our analysis, we have found it more convenient to use the ratios

$$c^{(i)} = \frac{r^{(i)}}{r^{(i-1)}}, i \geq 1, r^{(i-1)} \neq 0,$$

instead of  $\{r^{(i)}\}$ .  $c^{(i)}$  is a measure of reduction in distance ratios in consecutive iterates, and connect the  $i$ th ratio to the 0th ratio: We have  $r^{(0)} = \frac{h-y}{y}$ , and  $r^{(i)} = c^{(i)}r^{(i-1)} = r^{(0)} \prod_{1 \leq j \leq i} c^{(j)}, i \geq 1$ .

We now have that a triple  $(y, h, \{c^{(i)}\})$  where  $y \in (0, 1]$ ,  $h \in [y, 1.0]$ ,  $c^{(i)} \in [0, 1], i \geq 1$ , and for  $i \geq 2$ ,  $c^{(i)} = 0$  whenever  $c^{(i-1)} = 0$ , or  $c^{(i-1)} = 1.0$  (these correspond to the cases where  $x^{(i)} = 1.0$ , and  $r^{(i)} = 0$ ), completely specifies tight sequence. The following then follows from the definition of tight sequences, Lemma 5.3.11, and the surrounding comments:

**Lemma 5.3.12** *Any Newton sequence defining parameters  $y, h$ , and  $\{c^{(i)}\}$  dominates a corresponding tight Newton sequence generated by  $(y, h, \{c^{(i)}\})$ .*

Lemma 5.3.13 establishes an exact expression for  $1 - x^{(i)}$  for tight sequences. Let  $m^{(i)}$  denote the slope of line  $l^{(i)}$ . The product  $\prod_{1 \leq j \leq i} c^{(j)}$  is used numerous times in the expressions of the analysis, and we let the short hand notation  $c^i = \prod_{1 \leq j \leq i} c^{(j)}$ ,  $i \geq 1$ , not to be confused by the  $i$ th sequence element  $c^{(i)}$ . Let  $c^0 = 1$ .

**Lemma 5.3.13** *For any tight sequence  $\{x^{(i)}\}$  generated by  $(y, h, \{c^{(i)}\})$ ,  $\forall i \geq 0$ ,*

$$1 - x^{(i+1)} = \frac{1 - h}{1 - m^{(i)}} \prod_{1 \leq j \leq i} \frac{c^{j-1}(\frac{h}{y} - 1)(1 - c^{(j)})}{1 + c^{j-1}(\frac{h}{y} - 1)}, \quad (5.1)$$

where we have  $0 \leq m^{(i)} \leq 1 - y$ .

**Proof.** The proof proceeds by expressing  $1 - x^{(i+1)}$  first as a product in terms of  $1 - x^{(i)}$ ,  $m^{(i)}$ , and the parameters generating the sequence, and then deriving the expression above by replacing for  $1 - x^{(i)}$  using the inductive expression and further simplifications. See Appendix C.  $\square$

**Corollary 5.3.14** *For any Newton sequence  $\{x^{(i)}\}$  with  $y > 0$ , for any  $\epsilon > 0$ ,  $1 - x^{(k)} < \epsilon$ , for  $k > 2 \log(1/y) + \log(1/\epsilon)$ .*

**Proof.** We derive an upper distance bound  $\delta^{(i)}$  on  $1 - x^{(i)}$  in a tight sequence using expression 5.1 given in Lemma 5.3.13. Then we show that at every subsequent distance bound either we have a “halving” progress step, that is the distance bound is halved or we have a “latent” progress step, in that a quantity in the distance expression, but not the distance bound itself, is halved. We will see that we have no more than  $O(\log \frac{1}{y})$  latent steps, and all the rest of the steps are halving steps. The bounds apply to any Newton sequence by Lemma 5.3.12.

We have  $\frac{1-h}{1-m^{(i)}} < \frac{1}{1-m^{(i)}} \leq \frac{1}{y}$  (as  $m^{(i)} \leq 1 - y$ ), and  $\frac{c^{j-1}(h/y-1)(1-c^{(j)})}{1+c^{j-1}(h/y-1)} = \frac{1-c^{(j)}}{1+\frac{1}{c^{j-1}(h/y-1)}} < \frac{1-c^{(j)}}{1+\frac{1}{c^{j-1}(\frac{1}{y})}}$ , thus

$$1 - x^{(i+1)} < \delta^{(i+1)} = \frac{1}{y} \prod_{1 \leq j \leq i} \frac{1 - c^{(j)}}{1 + \frac{1}{c^{j-1}(\frac{1}{y})}}$$

Consider the ratio of two consecutive distance bounds, where  $i \geq 1$ :

$$\frac{\delta^{(i+1)}}{\delta^{(i)}} = (1 - c^{(i)}) \frac{1}{1 + \frac{1}{c^{i-1}(\frac{1}{y})}}, i \geq 1.$$

Note that both of the factors are less than 1. Now first assume  $c^{i-1}$  is small enough so that  $c^{i-1}(\frac{1}{y}) \leq 1$ . Then we have  $\frac{1}{1 + \frac{1}{c^{i-1}(\frac{1}{y})}} \leq 1/2$ , and thus  $\frac{\delta^{(i+1)}}{\delta^{(i)}} \leq 1/2$ . It follows that we always get a halving step once  $c^{i-1}(\frac{1}{y}) \leq 1$ . Now assume  $c^{i-1}(\frac{1}{y}) > 1$ . If  $1 - c^{(i)} < 1/2$ , we still obtain a halving step, and otherwise we have a latent step where  $c^i \frac{1}{y} \leq 1/2 c^{i-1} \frac{1}{y}$ . It follows that we can have at most  $O(\log \frac{1}{y})$  such latent steps before  $c^{i-1}(\frac{1}{y}) \leq 1$ .  $\square$

### *Analysis Based on a Local Improvement Constraint*

Radzik in [107] analyzes the Newton's method for solving several linear fractional combinatorial optimization problems. These problems include the maximum mean-weight cut and the minimum cost-to-time ratio cycle problem [1]. His basic argument that shows a polynomial bound is simple. Radzik notes that the Newton's method is a common method for solving (parametric versions) of fractional programs [59]. In Chapter 7, we discuss this related work further. We next describe and give intuition behind Radzik's argument, and then establish it for our case in Fig. 5.9b.

Consider Newton's method for finding the zero of a convex function  $F$ , as shown in Fig. 5.9a. We use a similar notation to that of the previous section. In Fig. 5.9a,  $\mu^{(i)}$  denotes the slope of the  $i$ th linear function picked (the tangent to function  $F$ ) whose zero is the  $i + 1$ st iterate  $x^{(i+1)}$ . Here,  $\delta^{(i)} = F(x^{(i)}) - x^{(i)}$ . We have  $0 \leq \delta^{(i+1)} < \delta^{(i)}$  (the function values get closer to zero) and  $\alpha^{(i+1)} < \alpha^{(i)}$  (the angle of the tangent lines go to zero), thus for the slopes we have  $0 \leq -\mu^{(i+1)} < -\mu^{(i)}$ . Therefore,  $0 \leq \frac{\delta^{(i+1)}}{\delta^{(i)}} < 1$  and  $0 \leq \frac{\mu^{(i+1)}}{\mu^{(i)}} < 1$ . The constraints from the way Newton's method operates are used to show that in fact a simultaneous constraint holds on the two fractions:

$$\frac{\delta^{(i+1)}}{\delta^{(i)}} + \frac{\mu^{(i+1)}}{\mu^{(i)}} \leq 1. \quad (5.2)$$

so that in each iteration either  $\frac{\delta^{(i+1)}}{\delta^{(i)}} \leq 1/2$  or  $\frac{\mu^{(i+1)}}{\mu^{(i)}} \leq 1/2$ , and furthermore  $\frac{\delta^{(i+1)}\mu^{(i+1)}}{\delta^{(i)}\mu^{(i)}} \leq 1/4$ . In [107], the geometric reduction in the elements of the sequence  $\{\delta^{(i)}\mu^{(i)}\}$  is used to prove polynomial bounds.

In the MDP(2) case (Fig. 5.9b), we can show similar progress<sup>2</sup> by mapping the problem

---

<sup>2</sup>Considering the *gain* of cycles and policies (see Section 4.4.1) instead of their values would give us the situation in Fig. 5.9a as well.

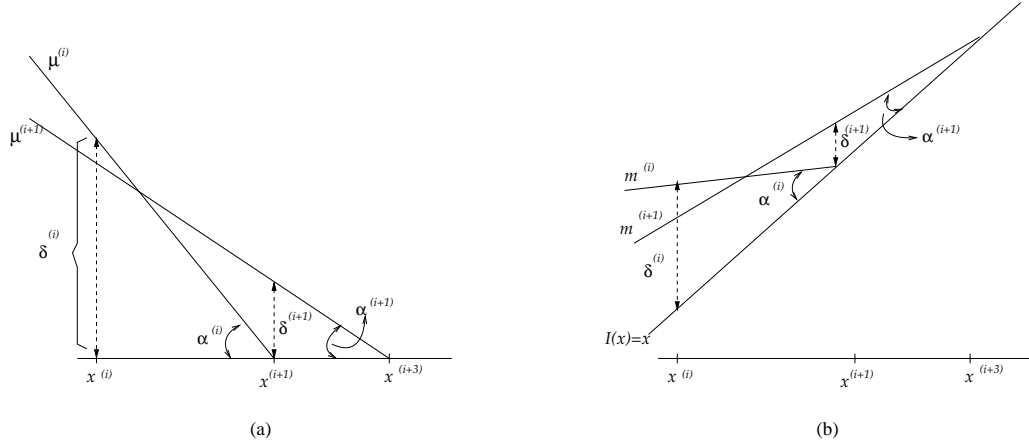


Figure 5.9: (a) Newton's method. We have  $\delta^{(i+1)} < \delta^{(i)}$  and  $\alpha^{(i+1)} < \alpha^{(i)}$ , thus  $(-m^{(i+1)}) < (-m^{(i)})$ . (b) Again  $\delta^{(i+1)} < \delta^{(i)}$  and  $\alpha^{(i+1)} < \alpha^{(i)}$ , thus  $1 - m^{(i+1)} < 1 - m^{(i)}$ .

in Fig. 5.9b to Fig. 5.9a. Here we show a variation of the inequality 5.2 directly for the case of Fig. 5.9b. The observation and the proof is adapted from [107].

**Lemma 5.3.15** *For the Newton method shown in Fig. 5.9b, we have,*

$$\frac{\delta^{(i+1)}}{\delta^{(i)}} + \frac{1 - m^{(i+1)}}{1 - m^{(i)}} \leq 1. \quad (5.3)$$

**Proof.** We have,

$$\begin{aligned} \delta^{(i)} &= F(x^{(i)}) - x^{(i)}, \\ \delta^{(i+1)} &= F(x^{(i+1)}) - x^{(i+1)}, \\ m^{(i)} &= \frac{x^{(i+1)} - F(x^{(i)})}{x^{(i+1)} - x^{(i)}}, \\ m^{(i+1)} &\geq \frac{F(x^{(i+1)}) - F(x^{(i)})}{x^{(i+1)} - x^{(i)}}, \end{aligned}$$

thus  $1 - m^{(i+1)} \leq 1 - \frac{F(x^{(i+1)}) - F(x^{(i)})}{x^{(i+1)} - x^{(i)}}$ , or

$$1 - m^{(i+1)} \leq \frac{x^{(i+1)} - x^{(i)} - F(x^{(i+1)}) + F(x^{(i)})}{x^{(i+1)} - x^{(i)}} = \frac{\delta^{(i)} - \delta^{(i+1)}}{x^{(i+1)} - x^{(i)}}.$$

Similarly,

$$1 - m^{(i)} = \frac{x^{(i+1)} - x^{(i)} - x^{(i+1)} + F(x^{(i)})}{x^{(i+1)} - x^{(i)}} = \frac{\delta^{(i)}}{x^{(i+1)} - x^{(i)}}.$$

we thus have

$$\frac{1 - m^{(i+1)}}{1 - m^{(i)}} \leq \frac{\delta^{(i)} - \delta^{(i+1)}}{\delta^{(i)}} \Rightarrow \frac{\delta^{(i+1)}}{\delta^{(i)}} + \frac{1 - m^{(i+1)}}{1 - m^{(i)}} \leq 1.$$

□

### 5.3.4 Consequences for Policy Iteration on Adags

Recall that with  $W$  denoting the highest integer in magnitude in the input, the value of any vertex under any policy falls in  $[-2nW^2, 2nW^2]$ , and two different values of a vertex under two policies are not closer than  $W^{-2n}$ . Let  $f^*$  denote the value function corresponding to the optimal  $s$ - $s$  cycle, where  $s$  is the bottleneck vertex.

Let us first use the bounds derived from the analysis that upper bounds  $1 - x^{(i)}$ . The quantity corresponding to  $\frac{1}{y}$  (or  $\frac{x^* - x^{(0)}}{f^*(x^{(0)}) - x^{(0)}}$ ) of the previous section is:

$$\delta = \frac{x^* - \lambda}{f^*(\lambda) - \lambda}, \quad (5.4)$$

where  $x^* \in [-2nW^2, 2nW^2]$  denotes the optimal value of  $s$  (the fixed-point of  $f^*$ ), and  $\lambda$  is the value of  $s$  under an initial policy, thus  $\lambda \in [-nW^2, nW^2]$ .

**Lemma 5.3.16** *We have  $\delta \leq W$ .*

**Proof.** Let  $m^*$  be the slope of  $f^*$  and we have  $m^* = \frac{x^* - f^*(\lambda)}{x^* - \lambda}$  or  $x^* = \frac{f^*(\lambda) - m^* \lambda}{1 - m^*}$ . Thus  $\delta = \frac{\frac{f^*(\lambda) - m^* \lambda}{1 - m^*} - \lambda}{f^*(\lambda) - \lambda} = \frac{\frac{f^*(\lambda) - \lambda}{1 - m^*}}{f^*(\lambda) - \lambda} = \frac{1}{1 - m^*}$ . The largest the slope of any cycle can be is  $\frac{W-1}{W}$ , thus  $\delta \leq \frac{1}{1 - \frac{W-1}{W}} = W$ . □

**Corollary 5.3.17** *Policy iteration on an adag comes within  $\epsilon > 0$  of optimal value  $x^*$  for  $s$ , in  $O(n(\log W + \log \frac{1}{\epsilon}))$  iterations. It thus finds all optimal values in  $O(mn^2 \log W)$  time.*

**Proof.** As values of  $s$  corresponding to each iterate  $x^{(i)}$  takes  $O(n)$  iterations to obtain, it follows from Lemma 5.3.16 and Corollary 5.3.14 that in  $O(n(\log W + \log \frac{1}{\epsilon}))$  iterations, value of  $s$  is within an  $\epsilon$  of optimal. Each iteration takes  $O(m + n)$  time, and getting within  $W^{-2n}$  of the optimal policy is enough to ensure obtaining the optimal policy and value, giving the run time  $O(mn^2 \log W)$  □

Similar bounds can be derived using Radzik's analysis. Here we consider the sequence  $\{\delta^{(i)}(1 - m^{(i)})\}$ , where  $\delta^{(i)} = F(x^{(i)}) - x^{(i)}$ , and  $m^{(i)}$  is the slope of  $F$  at  $x^{(i)}$ . The sequence goes down by at least a factor of 4 from one element to next. We have  $1 - m^{(i)} \leq 1$  and  $F(x^{(i)}) - x^{(i)} \leq \frac{nW}{1 - \frac{1}{W}} = nW^2$ , thus  $\delta^{(i)}(1 - m^{(i)}) \leq nW^2$ . We now lower bound  $\{\delta^{(i)}(1 - m^{(i)})\}$ . Let the function tangent to  $F$  at  $x^{(i+1)}$  have y-intercept  $y^{(i)}$ , as well as slope  $m^{(i)}$ , and fixed-point  $x^{(i+1)}$ , so that  $x^{(i+1)} = \frac{y^{(i)}}{1 - m^{(i)}}$ . We have  $\delta^{(i)} = F(x^{(i)}) - x^{(i)} = y^{(i)} + m^{(i)}x^{(i)} - x^{(i)} = y^{(i)} - (1 - m^{(i)})x^{(i)} = (1 - m^{(i)})\left(\frac{y^{(i)}}{1 - m^{(i)}} - x^{(i)}\right) = (1 - m^{(i)})(x^{(i+1)} - x^{(i)}) \geq W^{-1}W^{-2n}$ , as the difference  $x^{(i+1)} - x^{(i)}$  is not less than the closest two distinct values of a vertex under different policies can be. We therefore obtain the same bound of the previous analysis.

We now describe a speed-up that improves the run time by a factor of  $n$ . We will apply a similar speed-up when we discuss extensions of the algorithm to solve MDP(2) problems. In each iteration, we need only compute the best  $s$ - $s$  path for each value of  $s$ , and find its fixed point. This can be done in  $O(m)$  time, rather than  $O(mn)$ , using an algorithm similar to the shortest paths on dags (see for example [25]). At a given value  $\lambda$  of  $s$ , the algorithm replaces  $s$  with copies  $s'$  and  $s''$ ,  $s'$  with no out degree and with value  $\lambda$ , and  $s''$  with no indegree (the same transformation in defining  $F_{s,s}$ ). The graph is now a dag, so the algorithm does a topological sort of the vertices in terms of maximum distance to  $s'$ , and constructs the shortest path to  $s'$  from closest vertices to the farthest to  $s'$ , and once the best  $s'$  to  $s''$  path is found, which corresponds to the highest valued  $s$ - $s$  cycle at  $\lambda$ , computes its fixed-point. All this can be carried out in  $O(m)$  time, giving a run time of  $O(mn \log W)$  for finding all optimal values. This should be contrasted with the  $O(m)$  time shortest path algorithm for the dag problem. Can we improve on  $O(mn \log W)$ ?

### 5.3.5 Extensions to MDP(2)

Here, we show that policy iteration, shown efficient on adags in the previous section, can be used as a subroutine in a *freezing* algorithm for solving the more general MDP(2) problem in time  $nT$ , where  $T$  is the run time of policy iteration on an adag of the same size. Thus we obtain the first policy improvement algorithm efficient on MDP(2). Other

1. Freeze all vertices.
2. Repeat while there exists a frozen vertex  $s$ :
3.     Unfreeze vertex  $s$ .
4.     Apply policy iteration, respecting the fixed choice of edges for frozen vertices.

Figure 5.10: The Freezing algorithm

policy improvement algorithms, efficient on adags, may be used instead of policy iteration, and we show below that a Dijkstra style algorithm we obtain a better asymptotic bound.

The freezing algorithm is shown in Fig 5.10. Freezing a vertex means fixing an edge choice for it: the subroutine call will not change its edge choice. When a vertex is unfrozen it becomes a normal choice vertex, and any improvement in value of the vertices, while other frozen vertices remain frozen, involves a path to the unfrozen vertex. This situation is very similar to the case of the adag, and below we show that analyses of policy iteration on adag applies here. We also show that the speed-up technique of using a fast shortest path algorithm on the adag is applicable as well, but with a small modification: Because the graph has cycles, we cannot use the shortest paths algorithm on dags. However, as any new cycle will be a path from the unfrozen vertex  $s$  to itself, the problem is that of a shortest path problem with no negative cycles and we describe an efficient Dijkstra style algorithm. Consider the point where a frozen vertex  $s$  is just unfrozen and we may assume that  $s$  is the only remaining frozen vertex, and the algorithm has correctly found an optimal policy and the optimal vertex values for the problem with the old edge choice of  $s$ . Thus currently each vertex  $v$ , except vertex  $s$  perhaps, has value not less than  $C_{v,s}$ . In one iteration of policy iteration after  $s$  is unfrozen,  $s$  will also have value not less than  $C_{s,s}$ , as all its neighbors do. We argue that this situation is similar to the case we had with adags: any increase in the value of a vertex  $v$ , as policy iteration continues must involve a  $v$ - $s$  path, otherwise it would have been discovered while  $s$  was frozen. In particular, any new cycle will be an  $s$ - $s$  path. In case  $s$  has an optimal path to an existing cycle, policy iteration would take no more than  $n$  iterations to find all optimal vertex values and an optimal policy. Otherwise, as the

value of  $s$  increases, new  $s$ - $s$  paths become highest valued. Lemma 5.3.7 and its proof still apply: The  $k$ th vertex on the current (with respect to the value of  $s$  at the current iteration) highest valued  $s$ - $s$  path  $p$  will have a value no less than the value the path yields after at most  $k$  iterations. Thus the parametric and the Newton analyses on the adag apply to this portion of the algorithm, and the loop is repeated  $n$  times. Thus we obtain the following.

**Theorem 5.3.18** *The freezing algorithm with policy iteration finds an optimal policy in time  $O(mn^3 \log(nW))$ . In terms of  $m$  and  $n$  only, the run time is at most  $O(m^2 n^{O(\log n)})$ , and on deterministic MDPs, the run time is  $O(mn^3)$ .*

We can use efficient implementations of Dijkstra’s algorithm for single source shortest paths, in order to speed up the computation of the highest valued  $s$ - $s$  path. A subtlety is the choice of vertex “shortest path value” that we use. The shortest path algorithm, as in the adag case, will be used in a loop where a highest valued  $s$ - $s$  path is computed, and then the new value of each vertex under the cycle is computed in another  $O(n)$  operations. By value of a vertex we mean its value before the computation of a new  $s$ - $s$  path. We define the *shortest path* value of a vertex to be the maximum over the value of its edges (defined in Section 4.5) minus its current value. As in Dijkstra’s, a set  $S$  of vertices whose best path is found is maintained. The vertex to add to  $S$  is one with the highest shortest path value. The shortest path algorithm correctly finds an optimal  $s$ - $s$  path because the shortest path values do not create a cycle at this point, and a vertex  $v$  that currently has the highest shortest path value, can not have its shortest path value increased by the addition of another vertex  $w$  to set  $S$ , because if it uses its best path to  $w$ , its shortest path value can be at most that of  $w$  (as slopes of paths are at most 1). Thus adding  $v$  to  $S$  is sound. An efficient implementation of shortest path runs in  $O(m + n \log m)$  for finding the highest valued  $s$ - $s$  path, and we need  $O(n)$  operations for finding the value of the vertices under the new cycle. Thus each time a frozen vertex becomes unfrozen, it takes  $O((m + n \log m) \log W^n)$  iterations to find the optimal values, and the overall cost is a multiple of  $n$  of that, or  $O(mn^2 \log W + n^3 \log m \log W) = O(mn^2 \log(m) \log W)$ .

#### 5.4 Generalization to Feasibility in TVPI

In this section we extend the basic dynamic programming policy improvement method to handle the feasibility problem.

The feasible set in a monotone inequality system forms a semi-lattice with respect to either the min or max operations, depending on the direction of the inequalities, as described in Section 4.3. In case of monotone TVPI, the feasible set forms a lattice [62], thus non-empty and bounded feasible sets have both a glb and an lub. The algorithms that we describe naturally split into two main and symmetric phases of a *glb (computation) phase* and a *lub (computation) phase*. Each phase involves working on a directed multigraph and we call the graph for the glb phase the glb graph and the graph for the lub phase the lub graph. Again the graphs have symmetric structure and properties. In the glb phase, just as in policy improvement, the algorithms work with a vector of infeasible values for the variables, the vectors residing on the left side of the glb, and the sequence of the value vectors converges to the glb point. A similar scenario, convergence from the right side, applies in the lub phase. Fig. 5.12 shows pseudocode for a generic algorithm of this type, which we call *iterative-dual*. We describe the glb graph next.

Assume there are  $n$  indexed variables  $x_i, 1 \leq i \leq n$ . Corresponding to each variable  $x_i$  is a vertex  $v_i$ , and directed edges correspond to inequalities as follows: rewrite each inequality  $ax_i + bx_j \geq c$  involving two variables  $x_i$  and  $x_j$  where  $a > 0$  and  $b < 0$ , as  $x_i \geq m_e x_j + y_e$ , where  $m_e = -\frac{b}{a}, y_e = \frac{c}{a}$ , and for each such inequality put a directed edge from  $v_i$  to  $v_j$ . Lower bound inequalities involving one variable (those with  $m_e = 0$ ) will be handled as follows: We introduce an extra vertex  $v_0$  with no out edges, and for each inequality of type  $x_i \geq c$  is represented by an edge  $e$  from  $v_i$  to  $v_0$  with  $m_e = 0, y_e = c$ . Without loss of generality we will assume at most one edge to  $v_0$  per vertex. We will ignore single variable upper bound inequalities in the glb phase. Thus each edge is characterized by two parameters: its *slope*,  $m_e$ , and its *y-intercept*  $y_e$ . To make describing the algorithm and analysis simpler, for those variables  $x_i$  that don't have a lower bound given in the input ( $v_i$  doesn't yet have an edge to  $v_0$ ), an edge from  $v_i$  to  $v_0$  is added with slope zero, and y-intercept  $-L$ , where the constant  $L$  is defined later. Thus, we now assume that each

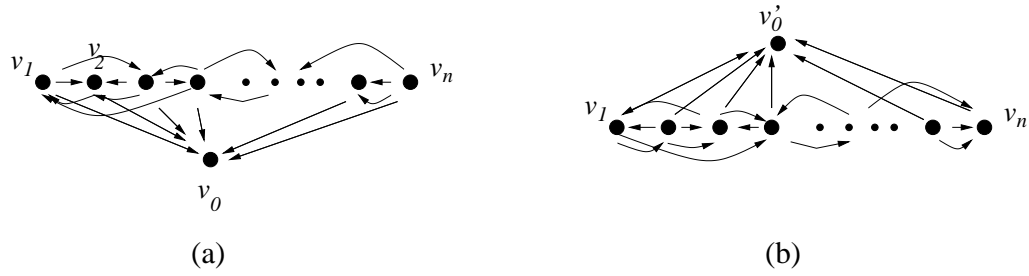


Figure 5.11: (a) The glb graph. (b) The lub graph. All nonzero slope edges in the glb graph are reversed in the lub graph.

- |  |
|--|
| <ol style="list-style-type: none"> <li>1. Construct the glb graph and call a glb finding algorithm.</li> <li>2. If infeasibility detected, return Infeasible.</li> <li>3. Otherwise construct the lub graph, and call an lub finding algorithm.</li> <li>5. If infeasibility detected, return Infeasible.</li> <li>6. Otherwise do final feasibility check: if for any variable, its left endpoint is greater than its right endpoint, report Infeasible, and otherwise report the endpoints.</li> </ol> |
|--|

Figure 5.12: A (generic) iterative-dual algorithm for computing the endpoints.

vertex  $v_i, i > 1$  has exactly one edge to  $v_0$  in the glb graph.

By a lower bound on a variable  $x_i$  or corresponding vertex  $v_i$ , we mean any value  $\lambda$  such that if the system is feasible, no feasible value of  $x_i$  is less than  $\lambda$ . Define an upper bound on  $x_i$  (vertex  $v_i$ ) similarly. We will now describe how lower bounds are derived from the glb graph. To each  $v_i$ - $v_j$  edge  $e$ , corresponds a *bounding* function  $f_e(\lambda) = m_e \lambda + y_e$ . This function has the property that a lower bound value  $\lambda$  for variable  $x_j$  imposes a lower bound of  $f_e(\lambda)$  on variable  $x_i$ . We extend the definition of the bounding function to simple cycles and paths just as we did for the case of the *MDP(2)*: for  $u$ - $v$  path  $p = e_1, \dots, e_k$  we have,  $m_p = \prod_{1 \leq i \leq k}$ , and  $y_p = y_1 + m_1(y_2 + m_2(y_3 + \dots)) \dots = \sum_{1 \leq i \leq k} \left( \prod_{1 \leq j < i} m_j \right) y_i$ , and  $f_p(\lambda) = m_e \lambda + y_e$ , with the semantics that a lower bound  $\lambda$  for  $v$  imposes a lower bound  $f_p(\lambda)$  on  $u$ . Note that for any path  $p$  we have  $m_p \geq 0$ , and that for any cycle  $c$ ,  $m_c > 0$ , because any edge with zero slope ends in  $v_0$  which has out-degree zero. If  $v_i$  has a path  $p$  to

$v_0$ , then  $x_i \geq y_p$ , i.e.  $y_p$  is a lower bound for  $x_i$ . Note that in this case  $f_p$  will be a constant function as  $m_p = 0$ . Cycles are another source of lower and upper bounds. Consider a  $v_i$ - $v_i$  cycle  $c$ , with corresponding bounding function  $f_c(\lambda) = m_c\lambda + y_c$ , for  $x_i$ . There are three possibilities based on  $m_c$ :

1. If  $m_c < 1$ , we say the cycle is *lossy*. For variable  $x_i$  we obtain:  $x_i \geq m_c x_i + y_c \Rightarrow (1 - m_c)x_i \geq y_c \Rightarrow x_i \geq \frac{y_c}{1 - m_c}$ . Thus a lossy cycle imposes a lower bound on the feasible values of each variable whose corresponding vertex is in the cycle, or which has a path to a vertex in the cycle.
2. If  $m_c = 1$ , the cycle is *break-even*, which leads to an inequality of the form  $0 \geq y_c$ . Therefore, the system is infeasible if  $y_c > 0$  (an *infeasible* break-even cycle), and the inequality is degenerate (uninformative) if  $y_c \leq 0$ .
3. If  $m_c > 1$ , the cycle is *gainy*. For a variable  $x_i$  we obtain:  $(1 - m_c)x_i \geq y_c \Rightarrow x_i \leq \frac{y_c}{1 - m_c}$ . Thus, a gainy cycle imposes an upper bound on the feasible values of a variable  $x_i$ , if  $v_i$  is in the cycle, or there is a path from a vertex in the cycle to  $v_i$ .

For a lossy or a gainy  $v_i$ - $v_i$  cycle  $c$ , we can speak of its unique intersection point  $x_c = \frac{y_c}{1 - m_c}$  with the identity line  $I(x) = x$ . We call  $x_c$  the fixed point of  $v_i$ - $v_i$  cycle  $c$ . Thus if  $v_i$  has a path  $p$  to vertex  $v_j$  in a lossy  $v_j$ - $v_j$  cycle  $c$ ,  $i$  and  $j$  not necessarily distinct,  $f_p(x_c)$  is a lower bound for  $x_i$ .

The definition and properties of the lub graph, for the lub phase, is similar to the glb graph. Consider the glb graph, and remove  $v_0$  and its incident edges, and reverse the remaining (nonzero slope) edges. Each inequality  $x_i \geq m_e x_j + y_e$  is rewritten (inversed) as  $x_j \leq \frac{1}{m_e} x_i + -\frac{y_e}{m_e}$ , so that the  $v_j$ - $v_i$  edge has slope  $\frac{1}{m_e}$  and y-intercept  $-\frac{y_e}{m_e}$ , where  $m_e$  and  $y_e$  were the parameters of the unreversed  $v_i$ - $v_j$  edge. Thus if  $f_p$  is the lower bounding function corresponding to a path  $p$  in the glb graph,  $f_p^{-1}$  is the upper bounding function corresponding to the path  $p$  reversed in the lub graph. Single variable upper bound edges are represented by each vertex having an edge to an extra vertex  $v'_0$ , with y-intercept  $L$  if no upper bound inequality for the variable is given in the input. See Fig 5.11(b). Define

bounding functions for edges, paths, and cycles, and lossy, break-even, and gainy cycles for the lub graph in a same way. In this case however, bounding functions from paths to lossy cycles impose upper bounds, while gainy cycles impose lower bounds.

Define the *left endpoint*  $l_i$  of a variable  $x_i$  to be the maximum over all lower bounds derived from paths from  $v_i$  to  $v_0$  or to lossy cycles in the glb graph. Note that there can only be finitely many such lower bounds. Let  $l_i = -\infty$ , if no such path exists. Define the *right endpoint*  $r_i$  for  $x_i$  similarly, using the lub graph. Note that the left and right endpoints of a variable are defined even if the system is infeasible. On the other hand, the endpoints characterize the feasible region when the system is feasible as Lemma 5.4.1 states below. Examples of feasible and infeasible situations are shown in Fig. 5.13. Variants of Lemma 5.4.1 appear in [112], who proved that feasibility in TVPI can be tested by following paths and cycles in a graphical representation of the problem. TVPI algorithms which came later are all based on such graph representations.

**Lemma 5.4.1** *A monotone TVPI system is infeasible if and only if there exists an infeasible break-even cycle in the glb graph (or equivalently in the lub graph), or for some variable  $x_i$ ,  $l_i > r_i$ . If the system is feasible, then the endpoints of the feasibility interval for each variable are its left and right endpoints.*

**Proof.** That infeasible break-even cycle lead to infeasibility is immediate. Any value less than  $l_i$  is by definition less than a lower bound derived from a path to a lossy cycle or inequality given in the input. Similar constraints holds for  $r_i$ . Thus if for any variable  $x_i$ ,  $l_i < r_i$ , the system is infeasible.

Now assume the necessary conditions for feasibility hold:  $l_i \leq r_i, 1 \leq i \leq n$ , and no infeasible break-even cycle exists. If  $n = 1$ , it follows that any value  $\lambda \geq l_1$ , and  $\lambda \leq r_1$  gives a feasible point. For  $n \geq 2$ , let us use the inductive hypothesis that any monotone TVPI with  $n - 1$  variables and satisfying the conditions is feasible (with feasibility regions bounded by  $l_i$  and  $r_i$ ). Now, for a system with  $n$  variables satisfying the conditions, we'll argue next that replacing  $x_n$  by a value  $\lambda$ ,  $l_1 \leq \lambda \leq r_1$ , in all inequalities that involve  $x_n$ , gives an  $n - 1$  variable system such that the corresponding glb and lub graphs have no infeasible break-even cycle and we still have  $l_i \leq r_i, 1 \leq i \leq n - 1$  for the reduced

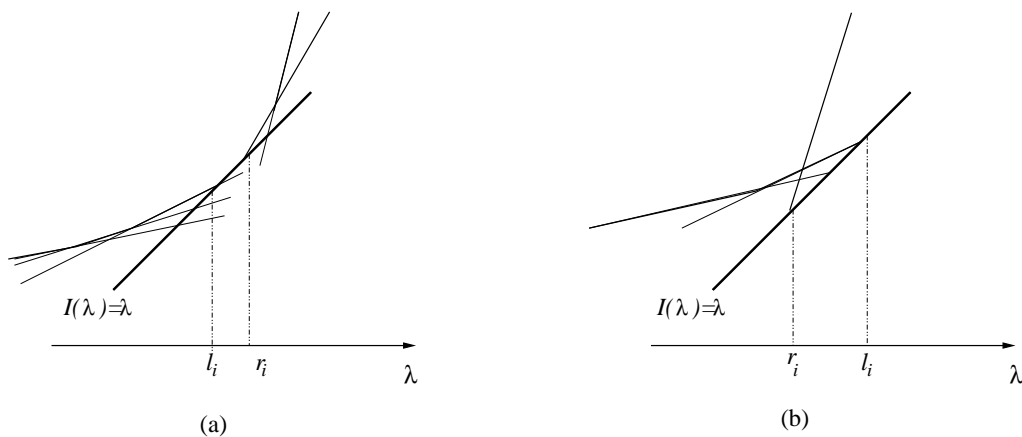


Figure 5.13: (a) When  $l_i \leq r_i$  for a variable  $x_i$ . (b) When  $l_i > r_i$ , which leads to infeasibility.

system/graph, and thus the system is feasible by induction.

To see this, in the new glb (lub) graph, any path that leads to  $x_n$  now leads to  $v_0$  ( $v'_0$ ). Therefore the set of cycles in the new graphs are a proper subset of the original graphs, thus no infeasible break-even cycle can appear, and if for some variable  $x_i$ ,  $l_i > r_i$ , at least one of the bounds must have been derived from a new path, *i.e.* the last edge of the path must correspond to an inequality to  $v_0$  or  $v'_0$  introduced when  $x_n$  was replaced. Assume this is the case for the lower bound  $l_i$ , so that there is a  $v_i - v_n$  path  $p$  such that  $f_p(\lambda) > r_i$  or equivalently  $f_p^{-1}(r_i) < \lambda$ . But then we have, in the original graph, that  $x_n$  has a path to  $x_i$ , and  $x_i$  has an upper bounding path to either a cycle, to  $x_n$ , or to  $v'_0$ , and in all cases this imposes the bound  $f_p^{-1}(r_i) < \lambda \leq r_n$ , contradicting the assumption on  $r_n$ .  $\square$

Figure 5.13 shows the two possibilities for a variable  $x_i$ , one of  $l_i < r_i$  and another, a case of infeasibility, where  $l_i > r_i$ . Those linear functions with slope less than 1 correspond to bounding functions for paths to cycles or  $v_0$  in the glb graph. Those with slope greater than 1 correspond to gainy cycles in the glb graph.

Thus the glb and the lub phases consist of working on the glb and the lub graphs to find the left and right endpoints respectively, as explained in more detail next.

#### 5.4.1 An Extension of Policy Iteration Algorithm for Monotone TVPI

Extensions of policy iteration for finding the left and right endpoints are presented in Fig. 5.14. The differences with policy iteration are the extra checks needed to detect infeasibility. As in policy improvement algorithms, each vertex  $v_i$  (excepting the special indexed zero vertices), carries a value and a choice of edge throughout the algorithm, which may change from one iteration to another. We refer to an assignment of a single (out) edge to each vertex, corresponding to a policy in MDPs, a *configuration*.

We need to deal with two complications that we describe next. Consider for example the glb phase, where we are interested in computing the left endpoints. Unlike MDPs, where computing the values from an arbitrary policy gives lower bounds on optimal values, here there are complications due to presence of break-even and gainy cycles: an arbitrary configuration may contain an uninformative break-even cycle, which yields no bounds, or a gainy cycle, which yields upper bounds rather than the lower bounds we seek. Another problem is that of value assignment to variables (vertices) that are not lower bounded (left point is  $-\infty$ ). The following lemma helps in solving the two problems. Its proof is similar to the proof of Lemma 5.2.4. Similar results appear in [5].

**Lemma 5.4.2** *Let  $W$  be the largest absolute value of an integer in the binary representation of a TVPI system consisting of  $n$  variables. Then the left point of each variable is either  $-\infty$  or greater than  $-2nW^{3n}$  and similarly its right point is either  $+\infty$  or less than  $2nW^{3n}$ .*

Let  $L = 3nW^{4n}$ , and  $L' = 2nW^{3n}$ . We can use  $-L$  and  $L$  as the y-intercept of the edges to  $v_0$  for those vertices which do not have a single variable lower (resp. upper) bound inequality in the input, and  $L'$  is used for detection of variables with  $l_i = -\infty$  or  $r_i = +\infty$ . As  $L$  and  $L'$  are large numbers compared to the input numbers, it may be preferable in practice to run an  $O(mn)$  algorithm to find lower bounds for all vertices that might have one. Those that have a path to a zero slope edge, or to a lossy cycle in the glb have a lower bound. Lossy cycles can be detected by a Bellman-Ford computation on the graph where edge weights are the logarithm of the slopes. Those vertices with no lower bound can then be ignored in the glb phase.

The glb phase begins with each vertex  $v_i, i \geq 1$ , choosing its  $v_i-v_0$  edge and having value

1. All vertices choose edge  $e$  to  $v_0$  ( $v'_0$ ) and take its value  $y_e$ .
2. Repeat until No-Change or Infeasibility
3. Each vertex selects a highest (*resp.* lowest) valued edge.
4. If no change in choice of edge for each vertex, then No-Change.
5. Otherwise, if any cycle is break-even or gainy, then Infeasible.
6. Otherwise lower (upper) bounds from the new configuration are evaluated for each vertex.
7. If Infeasible, return Infeasible.
8. Otherwise, for each vertex, if the value is less than  $-L'$  (greater than  $L'$ ) then report  $-\infty$  ( $+\infty$ ), and otherwise report its current value as the left (right) endpoint.

Figure 5.14: Iterative algorithms for the glb and, in the parenthesis, the lub phases, which are an extension of policy iteration.

$y_e$ . We show that in this way, no break-even cycles or gainy cycles are picked up during the algorithm, unless the system is infeasible, and when the algorithm stops without detecting infeasibility, it is easy to detect vertices with no lower bounds. As before, in describing the algorithm, it is convenient to define the value of an edge  $e$ , which is simply  $f_e(x)$ , where  $x$  denotes the value of the vertex the edge ends in at the beginning of an iteration. In the glb phase, each vertex chooses its highest valued edge, and to avoid degeneracies when ties occur, if the choice edge from previous iteration is in the tie, that edge is chosen and otherwise one with highest y-intercept among the highest valued edge choices is picked. The rest of the algorithm consists of processing the new cycles created and if none are break-even or gainy, updating the (candidate) lower bound values.

The correctness of the algorithm is established using the following lemmas. Our focus will be on the glb phase as the lub is symmetric. As before, let  $x_v^{(t)}$  denote the lower bound value of vertex  $v$  at time point  $t$ , at beginning of iteration  $t$ , thus  $t = 0$  is the initial time point. Take value of  $v_0$  to be zero at all  $t$ . A  $v$ - $v$  cycle  $c$  is said to be a positive gain cycle (not to be confused with a gainy cycle) at a value point  $x$  if  $f_c(x) > x$ , and it is said to be positive gain for  $v$  at time  $t$ , if it is positive gain at value  $x_v^{(t)}$ . By a new path or cycle (at a given time point  $t$ ), we mean a path or cycle that did not exist in the configuration at the previous time point  $t - 1$ . The proof of the next lemma is the same as that of Lemma

5.3.5, as the only difference now from the MDP(2) is that the edge slopes can be greater than one, and the proof of 5.3.5 does not use that assumption.

**Lemma 5.4.3** *Consider the glb phase. Assume a vertex  $u$  has a new path  $p$  to a vertex  $v$  at time  $t, t \geq 1$ . Then  $f_p(x_v^{(t-1)}) > x_u^{(t-1)}$ .*

**Corollary 5.4.4** *Any new cycle created during the glb phase has positive gain. At any iteration  $t \geq 1, \forall v \in V, x_v^{(t)} \geq x_v^{(t-1)}$ , and for at least one vertex,  $x_v^{(t)} > x_v^{(t-1)}$  until the algorithm stops. It follows that no configuration is repeated. Some new cycle is created in every  $n$  iterations until the algorithm stops.*

**Proof.** That new cycles have positive gain follows from Lemma 5.4.3, as a cycle is a  $u$ - $u$  path for every vertex  $u$  in it. If a cycle is break-even or gainy the algorithm stops, otherwise because its slope is less than 1, the evaluation of the cycle increases the values of the vertices in the cycle. It follows that a vertex increases its value if it gets a new path to  $v_0$ , a new path to an old cycle, or a new or old path to a new cycle. Otherwise, the value of the vertex doesn't change. Because in each iteration some vertex switches, it follows that at least one vertex increases value in each iteration, and no configuration repeats.  $\square$

**Lemma 5.4.5** *For any vertex  $v$  at time  $t \geq 1$ , we have  $x_v^{(t)} \leq -L'$  if and only if  $v$  has a path (in the configuration) to  $v_0$  that includes an edge (the last edge of the path) with  $y$ -intercept  $-L$ . No cycle has positive gain while any of its vertex values are less than  $-L'$ .*

**Proof.** From Lemma 5.4.2, if a vertex  $v$  has a path to a cycle or  $v_0$  where the edge is not labeled  $L$ , then  $x_v^{(t)} \geq -L'$ . If a vertex has a path to  $v_0$  with the last edge labeled  $-L$ , then its value is not greater than  $W + W^2 + W^3 + \dots + W^n + \frac{1}{W}(-L) \leq nW^n - \frac{3nW^{4n}}{W} \leq -2nW^{3n} = -L'$ .

It follows from Lemma 5.4.2 that a cycle has positive gain only if the value of each of its vertices is greater than  $-L'$ , and Lemma 5.4.4 states that a cycle is picked only if it has positive gain for all its vertices at the corresponding vertex values. Thus when a cycle is picked, it must be that all its vertices have values greater than  $-L'$ .  $\square$

**Lemma 5.4.6** *The glb phase returns infeasible only if the system is infeasible. Furthermore, if there exists an infeasible break-even cycle, the algorithm reports infeasible. The algorithm always stops and, if it doesn't report infeasible, it computes the left endpoints for all variables.*

**Proof.** If the algorithm reports infeasible, it is because either it has found a break-even cycle or a gainy cycle. A break-even  $v$ - $v$  cycle is positive gain at any value of  $v$  if and only if it is infeasible, because for any  $x$ ,  $f_c(x) = x + y_c > x$  if and only if  $y_c > 0$ . It follows from Lemma 5.4.4, that if an infeasible break-even cycle exists, the algorithm reports infeasibility in a finite number of steps.

On the other hand, if a created  $v$ - $v$  cycle  $c$  at iteration  $t$  is gainy, it imposes an upper bound on  $v$  less than  $v^{(t)}$ , as  $f_c$  has slope greater than 1 and  $f_c(x_v^{(t)}) > x_v^{(t)}$ , so  $f_c$  must intersect the identity function at a point less than  $x_v^{(t)}$ . It follows from Lemma 5.4.5 that all the vertex values of the cycle must be lower bounds, and the upper bound contradicts them, thus the system must be infeasible.  $\square$

If no break-even infeasible cycle is found in the first glb phase, no such cycle is found in the lub phase. However, there are other possibilities for infeasibility that might be discovered in the lub phase or afterwards: just consider a vertex  $v_i$  outside any cycle but still with  $r_i < l_i$  due to it having paths to lossy cycles both in the glb and lub graphs with the lower and upper bounds of each contradicting each other. However, it follows from our arguments on the glb phase, that the lub phase, if it doesn't stop due to infeasibility, computes the  $r_i$  for all variables correctly, and the last check of comparisons of  $l_i$  and  $r_i$  discovers any remaining infeasibility.

**Theorem 5.4.7** *The iterative-dual algorithm reports infeasible iff the system is infeasible, and otherwise computes the left and right endpoints of the feasibility interval for all variables.*

If a feasible point is desired when feasible, if the feasible set is bounded from below (when the left point of no variable is  $-\infty$ ) or from above, respectively the glb or the lub can be reported as a feasible value assignment. Otherwise, each variable can be initialized by its left endpoint if not  $-\infty$  and otherwise  $-L'$ , and after  $n$  value propagation phases where in each phase the value of the highest valued edge is picked, thus in  $O(mn)$  time, the resulting values can be reported.

It is not hard to see that correctness applies to other iterative value propagating and cycle and path evaluating glb or lub finding algorithms as well, as long as the properties in Lemma 5.4.4 are satisfied. Then starting the algorithms with lower or upper bounds as above, and checking cycles for break-even or gainy cycle varieties guarantees detection of infeasibility or finding the glb or lub in a finite number of steps. In the next section, the run time of a freezing algorithm is analyzed.

#### 5.4.2 Analysis of a Freezing Algorithm

We will describe a freezing algorithm for the glb phase and analyze its run time briefly. The freezing algorithm we describe is similar to the one for MDP(2), except for the check of infeasibility, and the initial choice of edge to fix for each vertex. The initial edge choice is not arbitrary, in order to avoid discovering a gainy or an uninformative break-even cycle, the same problems faced with in starting the algorithms of the previous section. And the solution is similar: Each vertex is frozen with the choice of edge  $e$  to  $v_0$  initially, and its value will be  $y_e$ . The remainder of the algorithm is straight forward: Repeatedly a frozen vertex is unfrozen and a glb algorithm such as the one of Fig. 5.14 is called, where the algorithm respects the current frozen vertices. It is not hard to see that the algorithm has the correctness properties of the unfreezing version: When vertices are frozen, fewer inequalities are considered, and if the algorithm reports infeasible during the glb phase, the system is infeasible. Eventually, all inequalities (edges) are considered, so infeasibility is detected or the algorithm correctly computes the endpoints.

The run time analysis is also similar to the ones in Sections 5.3.3 and 5.3.5. Here, the bounds on values we may obtain are slightly higher: we have that right end left endpoints can be in the order of  $nW^{O(n)}$  in magnitude, versus  $O(nW^2)$  in case of MDP(2). However the time to find the right and left endpoints is the same as in the case of the MDP(2) because two different lower or upper bounds for a variable are not closer than  $W^{-O(n)}$  as argued in [5] (proof similar to that of Lemma 5.2.4), and the largest slope less than 1 is still not greater than  $\frac{W-1}{W}$  which bounds the distance measure  $\delta$  in equation 5.4 in Section 5.3.4. We obtain:

**Theorem 5.4.8** *The freezing versions of the algorithms for finding the glb and lub solves the TVPI problem in  $O(mn^3 \log W)$  time.*

It is interesting to note that the Dijkstra style algorithm applicable in the MDP(2) case, for the asymptotic reduction in the run time of the basic algorithm, is not applicable here. In that case, slopes of all paths were not more than 1, while in case of TVPI they can be. This is similar to the case of shortest path problems with or without negative weighted edges. Assume a vertex  $s$  is unfrozen. The vertex  $u$  with the highest increase in value may still obtain a better value if another vertex  $v$  with a lower increase in value is first added to the list of those for which the best path to  $s$  is found, and a  $u$ - $v$  path with slope greater than 1 exists.

### 5.5 Summary and Discussion

We showed that versions of policy iteration are polynomial on MDP(2) and related problems such as deterministic discounted MDPs and TVPI problems. The key to establishing the polynomial run times were considering a parametric variant of the problem and a establishing a connection between policy iteration and Newton's method.

There are many open problems in both extending the analysis of policy improvement algorithms to more general classes of problems and in the design of policy improvement algorithms with better asymptotic or empirical properties. We list a few of the open problems and ways of approaching them below.

1. Does the Newton convergence analysis generalize to MDP(3), and to solving the feasibility problem in monotone LPs? Our analyses are in single dimensions, and we expect that a generalization to multiple dimensions will prove policy iteration polynomial on MDP(3).
2. An important direction is to determine tight bounds on the run time of the policy iteration algorithm and variants on MDP(2) and similar problems. We expect that the analysis of Section 5.3.4 can be improved to lower the number of iterations of policy iteration (and the speed-up variants) by a factor of  $n$  on adags (*e.g.*, down to roughly

$\tilde{O}(m)$  for the run-time of the speed-ups on adags), and that similar improvements can be made to the more general problem variants.

We also conjecture that policy iteration runs in strongly polynomial time on MDP(2). Radzik shows a strongly polynomial bound for Newton's method for several similar linear fractional programming problems [107]. It would be very interesting to determine whether a strongly polynomial bound holds for the MDP(3) problem.

3. Is the number of break points in  $F_{s,s}$  super polynomial? We expect that it is. This can probably be established by constructing a mapping to the shortest paths parametric problem of Carstensen [16]. The question of the number of break points in the MDP(3) problem is also an interesting one.
4. Several policy improvement algorithms have been shown to take exponential time on MDP(3) [89, 88]. How do these algorithms perform on MDP(2)? We remark that an example due to Edmonds appearing in [27] shows that a few simplex algorithms (which can be thought as policy improvement) do poorly even on the shortest path problems, which are special MDP(2) problems.
5. We have not shown (the classic) policy iteration (Fig 4.4b) polynomial for MDP(2). This would be established if the  $n$ -dimensional extension described in bullet 1 shows policy iteration polynomial on MDP(3). However we expect there are other potentially simpler ways, which may also give better bounds for policy iteration on MDP(2) or provide insight, for example on a proof of polynomial run time on MDP(3) without resorting to more sophisticated (potentially multi-dimensional) arguments.

The freezing algorithms do not depend on the order that vertices are frozen. This provides strong evidence that policy iteration, which considers all vertices and edges simultaneously in each iteration, is polynomial as well. A plausible proof strategy would show policy iteration dominates the freezing algorithm on some permutation of the order of the unfreezing of the vertices (or the order of adding new edges to vertices).

However, this seems hard and we haven't been able to find a simple argument based on this strategy.

Another approach is to consider generalizations of adags. What if all cycles went through at least one of two, and in general  $k$ , *bottleneck* vertices? We may think that the  $k$  bottleneck vertices are treated as sink (fixed-valued) vertices, and value iteration is performed for  $n$  iterations, where in the last iteration we allow the bottleneck vertices to switch. The resulting policy is evaluated, and again the bottleneck vertices are treated as sinks (frozen) and the computation sequence is repeated. How many iterations does this take? We have shown that for  $k = 1$  (the adag case) this is polynomial, for example this takes no more than  $n$  iterations for deterministic discounted MDPs. We conjecture that the number of iterations for  $k \geq 1$  is at most  $k$  times the number of iterations required for  $k = 1$  case.

Finally we may show that “quality” cycles are created and evaluated during the progress of policy iteration, just as is the case for adag graphs. The next chapter establishes such a result in the context of the average reward deterministic MDP.

## Chapter 6

**POLYNOMIAL VALUE ITERATION ALGORITHMS FOR AVERAGE  
REWARD DETERMINISTIC MDPS**

In this chapter, we show that value iteration finds the cycle with the highest average reward in a deterministic MDP (DMDP) in  $O(n^2)$  iterations. The analysis leads to a number of algorithms for solving the DMDP problem, all based on value iteration. We show that some of these algorithms have the best current strongly polynomial time of  $O(mn)$  on the problem. We report on empirical results on random graphs that show that several algorithms within this family should be competitive empirically as well. We close with a discussion of extensions of the results to related problems such as the deterministic discounted MDP and the MDP(2) problem.

### **6.1 Introduction**

In the previous chapter we argued that if we show a highest valued or equivalently a highest gain cycle is found and evaluated repeatedly as policy iteration progresses, we can show policy iteration takes polynomial time on the MDP(2) problem. Here it will be more convenient to consider the gain rather than the value of the cycles. For a vertex  $v$  with value  $x_v$  (at a given time point during policy iteration) in a cycle  $c$ , the gain of the cycle is  $f_c(x_v) - x_v$  for  $v$ . In the adag case, where all cycles share a vertex  $s$ , we showed that policy iteration and the simpler value iteration algorithms find a highest-gain cycle for  $s$  in at most  $n$  iterations for any fixed value of  $s$ . In a general graph however, because cycles are restricted to have no repeated vertices, the problem of computing a highest-gain cycle becomes equivalent to the NP-hard problem of finding a Hamiltonian cycle. Not surprisingly, the highest-gain cycle may be the optimal cycle and may not appear in a policy until the entire problem is solved. For example, we can construct a two-vertex MDP(2) example

where, during policy iteration, the highest-gain cycle which is a two-edge cycle, may be found only after many iterations, until all other single-edge cycles have nonpositive gain.

Therefore, in order to find a better measure of algorithm progress, we relax the problem. There are a number of possible relaxations of the problem and one which turned out to be fruitful is as follows. We ask whether a cycle with the highest *average* gain is created during policy iteration in a small polynomial number of iterations. The average gain of a cycle is the cycle gain divided by the number of edges in the cycle. This is sufficient to show polynomial run time, because a highest average gain cycle is the highest gain cycle in the set of cycles that have the same length. There are  $n$  possible cycle lengths, and showing effective progress on each possible length would show policy iteration polynomial.

To establish such a result, we simplify the problem to the deterministic MDP under the average reward criterion, which we call the DMDP problem for short. In a DMDP, the average gain of a cycle is the sum of the rewards of the edges of the cycle divided by the number of edges, and therefore it is independent of which vertex it is computed for or the current values of the vertices. In the more general MDP(2), there are complications as the gain of a cycle is a function of the value of the vertex, and during policy iteration vertex values change. We also simplify the policy iteration algorithm to value iteration, *i.e.* we drop the policy evaluation step. Similar to the adag case, we expect that value iteration suffices in finding good cycles for each fixed value. On the other hand, value iteration is known to be only pseudo-polynomial on a general MDP, and examples demonstrating its slow convergence to an optimal policy are easily constructed for MDP(2). Nevertheless, we show that in a deterministic MDP under the average reward criterion, an optimal cycle (a highest average reward cycle) is found in a polynomial number of iterations. Furthermore, the convergence to an optimal cycle is independent of the initial values assigned to the vertices. However it is interesting to note that it still takes value iteration pseudo-polynomial time to find an optimal *policy*: it may take a long time for a vertex outside any optimal cycle to “choose” its optimal out-edge.

We begin by formally defining the DMDP problem, and then present our analysis of the value iteration algorithm on the problem. In each iteration, value iteration visits a policy defined by the choice of the edge for each vertex. We show that value iteration converges to

an optimal cycle in  $\theta(n^2)$  iterations. Here, convergence means that all subsequent policies visited during value iteration will contain the optimal cycle. We expect that the first time an optimal cycle is created in a visited policy, however, is likely bounded by  $O(n)$  iterations only. In any case, the analysis suggests two algorithms that use value iteration and *histories* of edge choices to find the optimal cycle in  $O(n)$  iterations.

We next report on experiments on graphs where every vertex has two edges with neighbors chosen uniformly at random and rewards chosen uniformly in  $[0, 1]$ . The experiments suggest that the expected lengths of the optimal cycles grow only as a logarithmic factor of graph size  $n$ . The value iteration algorithms are modified to take advantage of this, and may have expected run time of only  $O(n \log n)$  on such graphs as the experiments suggest. Therefore, we expect that algorithms based on value iteration will be very efficient on DMDPs arising in practice as well.

We conclude with a summary and a discussion of algorithmic variations for better empirical performance and open problems in algorithm analysis.

## 6.2 Value Iteration on the Deterministic MDP Problem

Most of notation and terminology used in this section is defined in Section 4.1. Unlike the MDP(2) problem, here we deal with a problem on directed graphs where edges have a single parameter (the reward) only. Let  $G = (V, E)$  be a directed graph with  $n$  vertices and  $m$  edges. Let  $r$  be a function from  $E$  into real numbers such that for an edge  $e$  in  $E$ ,  $r(e)$  is the reward of  $e$ . For a walk  $w = e_1 e_2 \cdots e_k$ , let  $r(p) = \sum_{1 \leq i \leq k} r(e_i)$  be its *total reward*, and let  $\bar{r}(p) = \frac{r(p)}{k}$  be its *average reward*. Let  $\mu^* = \max_c \bar{r}(c)$ , where  $c$  ranges over all cycles in  $G$ . We call  $\mu^*$  the *maximum mean* and we define the DMDP problem as the problem of computing  $\mu^*$  or equivalently of finding an *optimal cycle* with average reward  $\mu^*$ . Note that an optimal policy here consists of a set of edge choices such that each vertex has a path to the maximum average reward cycle reachable. Once optimal cycles for each component are located, constructing an optimal policy is not hard. The DMDP problem, also known as the maximum (equivalently the minimum) mean cycle problem, was first shown to be solvable in  $O(mn)$  time by Karp [66, 25, 1]. There are several other algorithms for solving

the problem and a recent computational comparison of these algorithms on a variety of graphs families appears in [30]. Further background and related work on the problem is given in Section 6.4.1 and Chapter 7.

The basic value iteration procedure was described in Chapter 4. Here, the dynamic programming step is simpler as edges have only one parameter. Each vertex  $u$  at each iteration  $t$  performs the following computation to obtain its new value  $x_u^{(t)}$  :

$$x_u^{(t)} \leftarrow \max_{u-v \text{ edge } e} r(e) + x_v^{(t-1)}, \quad t \geq 1. \quad (6.1)$$

In each iteration, a vertex also “chooses” the out-edge that gives it the highest value. The choice of an out-edge for each vertex in each iteration of value iteration defines a policy which we say value iteration *visits*. We will assume that if the chosen edge from the previous iteration ties in the best value, then that edge is chosen again. In other words, value iteration is “lazy” in changing the policy from one iteration to another. This specification of value iteration comes in play in the proof of Lemma 6.2.7. Ties in the first iteration or when the previous edge choice is not a highest-valued edge may be broken arbitrarily.

### 6.2.1 Polynomial Convergence of Value Iteration On DMDPs

Here, we first show that analyzing value iteration on DMDP problems where  $\mu^* = 0$ , or the *mean-zero* case, suffices in showing polynomial convergence, then we analyze value iteration on the mean-zero case.

Consider value iteration as it progresses on a graph  $G = (V, E)$ , where vertices are initialized with arbitrary values. The *value* of a  $u$ - $v$  walk  $w$  at an iteration  $t \geq |w|$  is  $r(w) + x_v^{(t-|w|)}$ , in other words it is the value vertex  $u$  obtains if the value of  $v$  at time  $t - |w|$  is “propagated” (modified using computation 6.1) along the walk  $w$  till it reaches  $u$ . The next lemma relates the value of walks and the value of vertices as value iteration progresses. It is shown by induction on time  $t$  (or length of walks).

**Lemma 6.2.1**  $x_v^{(t)}$  is the maximum value over values of all walks of length  $t$  with start vertex  $v$ .

**Proof.** We have

$$x_v^{(t)} = \max_{e=(v,u) \in E} r(e) + x_u^{(t-1)}, t \geq 1.$$

On the other hand, if we let  $\mathcal{V}_t(v)$  denote the highest value over values of walks of length  $t$  with start vertex  $v$ , we have that  $\mathcal{V}_t(v) \geq \max_{e=(v,u) \in E} r(e) + \mathcal{V}_{t-1}(u)$ ,  $t \geq 1$ , where  $\mathcal{V}_0(v) = x_v^{(0)}$ , and an argument by induction shows that equality holds:  $\mathcal{V}_t(v) = \max_{e=(v,u) \in E_v} r(e) + \mathcal{V}_{t-1}(u)$ . It follows that  $\mathcal{V}_t(v) = x_v^{(t)}$ .  $\square$

The *history walk* of a vertex  $v$  at a given iteration is, informally, the walk formed following the sequence of edge selections of value iteration starting at  $v$  and going back in time. More precisely, at iteration  $t$ , if vertex  $v$  has edge  $e$  to  $u$ , then its history walk of one time step consists of edge  $e$ . Its history walk of  $k > 1$  steps is  $e$  concatenated with the history walk of length  $k - 1$  steps for vertex  $u$  at iteration  $t - 1$ . The history walk is necessarily a maximum valued walk by Lemma 6.2.1.

A pair of DMDP problems is called *parallel* if the problems have identical graphs  $G = (V, E)$ , but the reward function  $r$  in one problem is a constant offset of the reward function  $r'$  for the other, or, for some constant  $\mu$ ,  $\forall e \in E, r'(e) = r(e) - \mu$ . Observe that an optimal cycle in one problem is also an optimal cycle in the other as the average reward of any cycle in one problem is the constant offset  $\mu$  away from the average reward of the corresponding cycle in the parallel problem. Furthermore, as a consequence of the next lemma, value iteration behaves the same on parallel problems, meaning that each vertex selects the same edge in both problems at all iterations (subject to ties being broken identically).

**Lemma 6.2.2** *Consider value iteration started with equal initial values on a pair of parallel DMDP problems  $\mathcal{P}$  and  $\mathcal{P}'$ , with reward functions  $r$  and  $r'$  respectively and  $r'(e) = r(e) - \mu$ , for some constant  $\mu$ . Then if  $x_v^{(t)}$  is the value of vertex  $v$  at time  $t$  in problem  $\mathcal{P}$ ,  $x_v^{(t)} - \mu t$  is the value of vertex  $v$  in problem  $\mathcal{P}'$  at time  $t$ .*

**Proof.** Consider any  $u$ - $v$  walk  $w$  of length  $t$ . If walk  $w$  gives value  $x$  to  $u$  in problem  $\mathcal{P}$ , given value  $y$  for  $v$ , then it gives value  $x - \mu t$  to  $u$  given value  $y$  for  $v$  in  $\mathcal{P}'$ . It follows from Lemma 6.2.1 that if  $v$  has value  $x_v^{(t)}$  in  $\mathcal{P}$  at time  $t$ , then its value in  $\mathcal{P}'$ ,  $x'_v(t)$ , is not less than  $x_v(t) - \mu t$ . Symmetry establishes that  $x'_v(t) = x_v^{(t)} - \mu t$ .  $\square$

Imagine value iteration is started on any two parallel problems with identical initial vertex values. Then the policies visited are identical. Now, given a graph with maximum mean  $\mu^*$ , the parallel problem with offset  $\mu^*$  has maximum mean 0. In this case, if we see a visited policy contains the (optimal) mean-zero cycle in the mean-zero case at some iteration  $t$ , in the original parallel problem the policy visited at iteration  $t$  contains the (identical) optimal cycle. Similar observation can be made for history walks. Below, we show several properties hold on the visited policies and history walks made by value iteration on mean-zero problems. It follows that the same properties hold in the general case.

### 6.2.2 Analysis of Value Iteration on the Mean-Zero DMDP and its Generalizations

In this section we characterize aspects of the behavior of value iteration on the mean-zero case, and from it deduce the general case. The rough argument of convergence of value iteration on the mean-zero case goes as follows. In the mean-zero case, no cycle has a positive total reward. So when a value gets propagated once along any cycle, it does not increase (Lemma 6.2.3). Consider any vertex and the sequence of values it obtains over time as value iteration progresses. We show that the sequence is bounded and in fact the maximum is obtained in the first  $n$  iterations as a consequence of cycles having nonpositive total reward. Similarly, the maximum value the vertex obtains over the even (or odd) iterations or time points is also obtained in  $2n$  iterations, and in general (Lemma 6.2.4), for any  $k$ , the maximum value obtained over all iterations  $t \equiv 0 \pmod{k}$ , (and for any  $i$ , all iterations  $t \equiv i \pmod{k}$ ) is obtained in at most  $kn$  iterations. For those vertices that are on the mean-zero (optimal) cycle, these maximum values are “preserved” along the cycle, and, once all  $|c|$  maximum values “arrive” at the cycle, which happens in at most  $|c|n$  iterations, the vertices of the cycle form the cycle forever and will not switch any more.

It is best to think of a walk as a sequence of vertices, so for example walk  $w = v_2v_5v_1v_3v_1v_5v_4$  (unlike MDP(2), here edges are uniquely determined by their start and end vertices). A loop is just a walk with identical first and last vertices, so it may have multiple cycles. Removing a loop from a walk is simply removing the vertices of the loop from the walk, so if we remove the  $v_5$ - $v_5$  loop from walk  $v_2v_5v_1v_3v_1v_5v_4$  we obtain the walk

$v_2v_5v_4$ .

**Lemma 6.2.3** *In the mean-zero case, if a loop is removed from a walk, the resulting walk has equal or higher total reward. Therefore, if a vertex  $v$  has a history walk of length  $j$  to the same vertex  $v$  at an iteration  $t$ , then  $x_v^{(t)} \leq x_v^{(t-j)}$ .*

**Proof.** Removing a loop is equivalent to removing one or more cycles, and because cycles have nonpositive total reward, the whole loop has nonpositive total reward.  $\square$

Now, consider the sequence of values of a vertex  $v$ ,  $x_v^{(t)}$ , as value iteration proceeds on a mean-zero DMDP problem. The next lemma bounds the latest time an increase in the maximum value can occur over the sequence  $\{x_v^{(t)}\}$  and also its subsequences. Thinking of the  $k = 1$  case helps in understanding the lemma and its proof. For the  $k = 1$  case, the lemma says that for any vertex, an increase in its maximum value—over its values seen so far—can occur only in the first  $n$  iterations.

**Lemma 6.2.4** *For any  $k \geq 1$ , at any time  $l \geq k$ , if*

$$x_v^{(l)} > x_v^{(i)}, \forall i < l, \text{ such that } i \equiv l \pmod{k}, \quad (6.2)$$

*then we have  $l \leq kn$ .*

**Proof.** Follow the history walk for  $v$  at time  $l$  for  $k$  steps, and assume we arrive at a vertex  $u$ . By Lemma 6.2.1 and property 6.2, the property must hold for  $u : x_u^{(l-k)} > x_u^{(i)}$  for  $i < l - k$ , and such that  $i \equiv l \pmod{k}$ . Also we have  $u \neq v$  as otherwise we would have  $x_v^{(l)} \leq x_v^{(l-k)}$  by 6.2.3 contradicting the property. Similar constraints apply if we follow  $u$  at  $l - k$  for  $k$  steps, and so on. Therefore the sequence of vertices visited every  $k$  steps following the history walk of  $v$  has no vertex repetition. It follows that  $l \leq kn$ .  $\square$

Lemma 6.2.4 has many consequences. Let  $x_v^*$  be the highest value  $v$  ever obtains. As one consequence of Lemma 6.2.1,  $x_v^*$  exists, and  $v$  must obtain value  $x_v^*$  in the first  $n$  iterations.

Now, consider a vertex  $v$  in the mean zero cycle  $c$ , and let  $k = |c|$ . Let  $x_v[j], 0 \leq j < k$ , be the highest value vertex  $v$  gets in any iteration  $t \equiv j \pmod{k}$ . These  $k$  values are well defined (bounded) by Lemma 6.2.4. Since  $c$  has mean zero, it follows from Lemma 6.2.1 that once  $v$  obtains such a value, it gets it back every  $|c|$  iterations. We call the  $k$  values the highest  $k$  values of  $v$ .

**Lemma 6.2.5** *For any vertex  $v$  in an optimal cycle of length  $k$ , once  $v$  obtains its highest value  $x_v[j]$ ,  $0 \leq j < k$ , it obtains it again in every  $k$  iterations.*

Vertex  $v$  obtains its highest  $k$  values in no more than  $kn$  iterations by Lemma 6.2.4. The next result is used both in showing convergence to an optimal cycle and in developing fast algorithms in Section 6.2.4. The proof is based on the observation that once  $v$  obtains  $x_v^*$ , some vertex in  $c$  obtains its highest value in every subsequent iteration, and the history walk of a vertex in  $c$  at an iteration when the vertex obtains its highest value cannot include a suboptimal cycle by 6.2.3.

**Lemma 6.2.6** *For any DMDP problem, at any iteration  $t \geq n$ , the history walk of some vertex on an optimal cycle includes only optimal cycles.*

**Lemma 6.2.7** *In a mean-zero DMDP problem, when all vertices in all optimal cycles have obtained their highest values, after at most  $n$  iterations, some optimal cycle appears in all subsequent policies.*

**Proof.** When vertices of an optimal cycle find all their highest values, if some vertex switches to its neighbor in the cycle, it will not switch again from our assumption on “lazy” policy change. When all vertices of optimal cycles find their highest values, whenever a vertex  $v$  obtains  $x_v^*$ , its history walk can only contain mean-zero cycles by 6.2.6, therefore the vertices in such cycle will not switch again.  $\square$

We need the assumption of “lazy” change in policy for Lemma 6.2.7 to hold. As Fig. 6.1a shows, if ties are broken arbitrarily or based on an ordering of edges or their end vertices, then the optimal cycle may never form or may repeatedly appear and disappear. Also, examples exist where some vertices may repeatedly switch edges forever (Fig. 6.1b), but it can be shown that eventually all vertices have a path to some optimal cycle in any visited policy. The worst-case number of iterations to the formation of such (optimal) policies is, however, pseudo-polynomial [125].

Fig. 6.2a shows an example where it takes  $\Omega(n^2)$  time for an optimal cycle to become permanent in the visited policies. Assume  $n$  is even, the vertices are initialized with  $-1$ , and all edges have reward 0, unless indicated. Therefore, the cycle on the left of vertex  $v$  is

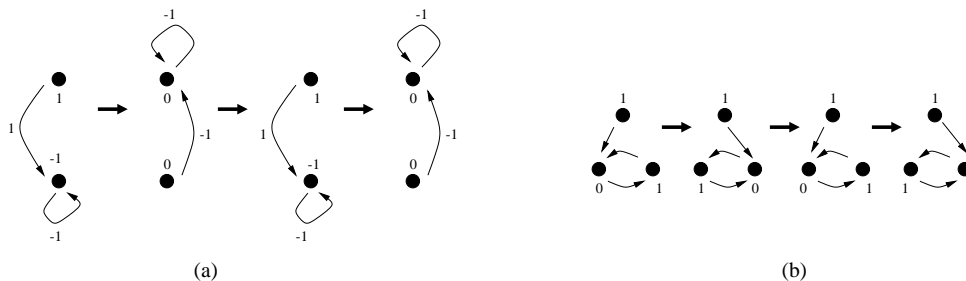


Figure 6.1: (a) The optimal cycle (with mean zero here) may not form if ties are broken arbitrarily or if each vertex uses a local ordering of edges. Here, the values labeling the vertices are the ones obtained at the end of the iteration (*i.e.*  $x_v^{(t)}$  for iteration  $t$ ) and used for the next iteration. (b) An example where a vertex may switch forever. In this example, all edges have reward 0, and the vertices on the optimal cycle are initialized with (or somehow have obtained) values 1 and 0.

optimal with mean 0 and length  $\frac{n}{2} - 1$ . The cycle on the right has an edge with reward  $-1$  and therefore has a slight negative mean with length  $\frac{n}{2}$ . It is not hard to verify that vertex  $v$  switches to form the suboptimal cycle periodically for  $\frac{n}{2} \times (\frac{n}{2} - 1)$  iterations.

As a corollary of Lemmas 6.2.7, 6.2.4, and 6.2.2, we obtain the following theorem on value iteration on DMDPs. It follows from the comments following Lemma 6.2.2 that the result holds for value iteration on any DMDP problem.

**Theorem 6.2.8** *Value iteration converges to an optimal cycle in a DMDP problem in  $\theta(n^2)$  iterations.*

We expect that the first time formation of an optimal cycle is earlier:

**Conjecture 6.2.9** *An optimal cycle is first formed in a policy in  $O(n)$  iterations.*

Simple examples requiring  $n - 1$  iterations exist. If the conjecture is true, an algorithm that alternates value iteration with a routine that updates the average reward computed from the cycles in the current policy would find an optimal cycle in  $O(n)$  iterations instead of our current best bound of  $O(n^2)$ . A step toward proving the conjecture is to give a linear bound on the sum, over all  $k$  subsequences in Lemma 6.2.4, of the number of increases in the maximum value seen so far.

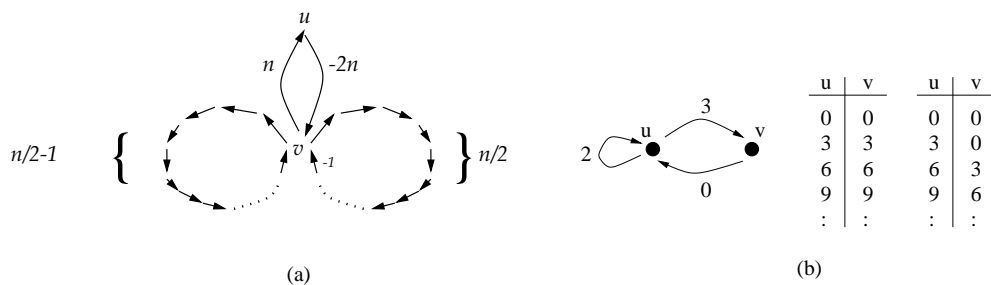


Figure 6.2: (a) An example graph where it takes  $\Omega(n^2)$  iterations for the highest values to reach the optimal cycle. (b) An example where Gauss-Siedel value iteration converges to a cycle (here the two-edge cycle) other than the highest average reward cycle, irrespective of the order of processing the vertices. The tables show the value of the vertices in each iteration with respect to the two possible processing orders.

### 6.2.3 Gauss-Siedel Value Iteration

It is interesting to consider the Gauss-Siedel value iteration, where vertex values are used in computing the edge values as soon as they become available [103]. Imagine scanning vertices in (some) order, and when computing the value  $x_i^{(t)}$  for vertex  $v_i$ , value  $v_j^{(t)}$  computed from current iteration  $t$  is used for any vertex  $v_j, j < i$ , in computing the out-edge values of  $v_i$ . This is the easier implementation of value iteration on a sequential machine.

Gauss-Siedel value iteration does not necessarily converge to the optimal (highest average reward) cycle. Fig. 6.2b shows a counterexample. In this case, Lemma 6.2.1 is not true, as the value of a vertex at iteration  $t$  may be the value of a walk of length greater than  $t$ . Therefore Lemma 6.2.2 does not apply either and the behavior of value iteration on two parallel problems can be different. However, as Fig. 6.2b suggests, we expect that Gauss-Siedel value iteration does converge to some cycle, probably in  $O(n^2)$  iterations, and the cycle has total reward greater than or equal to the total reward of the highest average reward cycle. The above mean-zero results hold for Gauss-Siedel value iteration, when the maximum total reward and average reward cycles are identical. More work in this direction should be insightful.

Due to the incorrectness of Gauss-Siedel value iteration on the DMDP problem, in implementing value iteration for obtaining the optimal cycle, care must be taken to keep two

sets of values for each vertex, one set of values from the previous iteration. In other applications of value iteration, for example in finding optimal total reward (discounted) policies and values, the Gauss-Siedel variation is a preferred method as it is easier to implement and takes less memory, and as discussed in [103] (under *regular splitting* methods), it tends to converge faster. A similar variation exists for policy iteration.

#### 6.2.4 Two Algorithms Based on History Walks

We can still compute the optimal cycle using basic value iteration in  $O(n)$  iterations even if the conjecture on first formation time does not hold. Lemma 6.2.6 shows us how. If we keep track of the edge chosen by each vertex in each iteration for the first  $n$  iterations as value iterations progresses, we can reconstruct the cycles in the history walks, and some cycle must be optimal by Lemma 6.2.6. Searching for the optimal cycle takes  $n^2$  time, thus the algorithm takes  $O(mn)$  time, but also uses  $\theta(n^2)$  extra space.

We next describe a variation that reduces the space back to linear. This algorithm also works in two phases, the first phase being simple value iteration for  $n$  iterations. The second phase also takes  $n$  iterations of value iteration plus an added  $O(1)$  computation per vertex per iteration. In each iteration after iteration  $n$ , each vertex keeps track, not only of its current value and chosen edge, but updates parameters characterizing its *super edge* as well. Super edges summarize history walks, and may be viewed as packets sent along edges of the network (in the reverse direction of the edges, just as values are). When a vertex discovers that a super edge was sent by itself, it computes the average value of the cycle corresponding to the super edge from the parameters of the super edge, and updates the current highest mean  $\mu$  found so far. The algorithm has the desirable property that just like value iteration it is distributed: each vertex performs a simple local computation until the optimal mean is found.

A super edge has three parameters denoted by the ordered triple  $(v, l, r_s)$ , where  $v$  is the vertex it ends in,  $l$  is the number of edges in the super edge, and  $r_s$  is its total reward. At any iteration, such as the beginning of the second phase, the super edge of a vertex may be undefined. In the second phase, each vertex  $u$  performs a value iteration step and

computes its current super edge as follows. Assume  $u$  chooses the  $u$ - $v$  edge  $e$  in iteration  $t$ . In case  $u \neq v$ , if the super edge for  $v$  of iteration  $t - 1$  is undefined, the super edge for  $u$  is defined to be  $(v, 1, r(e))$ . If  $v$  has super edge  $(z, l, r_s)$ , and  $u \neq z$ , the super edge for  $u$  is  $(z, l + 1, r(e) + r_s)$ . Otherwise, when  $u = v$  or  $u = z$ , vertex  $u$  has obtained a *loopy* super edge, and its average reward is respectively  $r(e)$  or  $\frac{r(e)+r_s}{l+1}$ . In this case, the running estimate  $\mu$  is updated if necessary, and vertex  $u$  marks its current super edge undefined.

The algorithm takes  $2n$  iterations and each iteration takes constant time per edge, thus the run time is  $O(mn)$ , with only  $O(n)$  extra space. Correspondence made between super edges and history walks, and Lemma 6.2.6 establish the correctness. A subtlety is when there are multiple optimal cycles and ties in edge values occur. In this case we assume a vertex chooses the edge whose end-vertex has a super edge with lowest numbered vertex. In the beginning of the second iteration where no vertex has a super edge we assume ties are broken based on the lower numbered end-vertex. We call this rule the “lowest-index” rule.

To establish the lemma, we first make several properties of super edges explicit. The next lemma can be verified from the definition of history walks and super edges, and by using induction on the length of super edges.

**Lemma 6.2.10** *The super edge of length  $l$  for a vertex  $v$  at any time point corresponds to the history walk of length  $l$  for vertex  $v$  at that time. A loopy super edge corresponds to a history walk that is a loop (the start and end vertices are identical).*

The next lemma refers to the running estimate  $\mu$  on  $\mu^*$  kept in the second phase of the algorithm.

**Lemma 6.2.11** *No loopy super edge has average reward greater than the optimal mean value  $\mu^*$ . Therefore  $\mu \leq \mu^*$  throughout the algorithm.*

**Proof.** The history walk corresponding to a loopy super edge is a loop, possibly with multiple cycles inside. Each cycle in the loop has average reward not greater than  $\mu^*$ , thus the maximum average reward cycle in the walk is not greater than  $\mu^*$ , and if there is any cycle with average reward less than  $\mu^*$ , the average reward of the whole walk is less than

$\mu^*$ . It follows that the average reward of the whole walk is not greater than  $\mu^*$ , and it is  $\mu^*$  if and only if all cycles of the loop are optimal cycles.  $\square$

Now we show that some loopy super edge computed in the second phase has mean  $\mu^*$ . This is not hard to see when the optimal cycle is unique, as the highest value in the mean-zero case is created and traces the cycle after the first  $n$  iterations. In case of multiple optimal cycles, ties in edges may not be broken arbitrarily, otherwise we can give examples where no loopy super edge corresponding to an optimal cycle is created in the second  $n$  iterations. We will prove the result for the “lowest-index” rule. We expect other easier rules—for example if each vertex breaks ties consistently locally—also give correct algorithms.

**Lemma 6.2.12** *Assume the lowest-index rule is used in breaking ties. Then some vertex obtains a loopy super edge corresponding to an optimal cycle in the second  $n$  iterations.*

**Proof.** Without loss of generality, take the mean-zero case. Consider the set  $S$  of vertices that are in some optimal cycle, and consider their highest value as it gets propagated along edges. We say  $u \in S$  can propagate to  $v \in S$  if there is a path from  $v$  to  $u$  such the highest value can propagate from  $u$  to  $v$  on the path (in the reverse of the edge directions).

The “can-propagate-to” relation is transitive: if  $u$  can propagate (the highest value) to  $v$  and  $v$  to  $w$  then  $u$  can propagate to  $w$ . Consider the transitive closure  $\mathcal{T}$  of the can-propagate-to relation.  $\mathcal{T}$  is composed of one or more fully connected components where in each component, each vertex can propagate to all vertices in the same component. Therefore each component contains all the vertices of at least one optimal cycle of the graph. Consider the partial ordering imposed by  $\mathcal{T}$  on the components, and take a component  $C$  with no predecessors (no vertex outside the component has an edge into the component).

After iteration  $n$ , if a vertex in  $S$  receives a highest value, it must have been propagated from another vertex in  $S$  (a vertex in some optimal cycle). This can be verified by examining the history walks: at least one optimal cycle and only optimal cycles exist in the history walk of length  $n$  or more whenever a vertex obtains its highest value. Thus vertices in  $C$  can obtain their highest values only from propagation of the highest value from vertices in  $C$ ; assuming otherwise contradicts the assumption that  $C$  has no predecessor components.

Component  $C$  has at least one vertex with the highest value in iteration  $n$ , as  $C$  has

at least one optimal cycle. Consider the lowest numbered vertex  $v \in C$  from such highest valued vertices. Now consider any optimal cycle  $c$  with smallest length from the optimal cycles containing  $v$ , and let  $v_0 = v$ , and  $v_i, i \leq |c| - 1$  denote the  $i^{\text{th}}$  vertex that has a path of  $i$  edges in cycle  $c$  to  $v$ . It follows from the tie breaker rule and properties of  $c$  that  $v_1$  will choose its edge in  $c$  to  $v$  in iteration  $n + 1$ , and will form a super edge ending in  $c$ , and in general  $v_i$  will have a history path of length  $i$  to  $v$  in the end of iteration  $n + i$  (not necessarily corresponding to  $c$  but with same total reward and length), and obtain a super edge ending in  $v$  with total reward equal to  $v_i$ 's path to  $v$  in  $c$ . Thus in iteration  $n + |c|$ ,  $v$  obtains a loopy super edge from at least one neighboring vertex in component  $C$ , corresponding to an optimal cycle, containing  $v$ , of shortest length.  $\square$

We have described an algorithm that has to run for  $2n$  iterations on any graph. A natural question is whether vertices may begin keeping track of super edges before iteration  $n$ . The answer is negative (in the worst case). An example exists where the optimal cycle is a single edge long (a self-edge) and if vertices start keeping track of super edges in any iteration before  $n$ , no super edge will form the loop corresponding to the optimal cycle. However, in the next section we give an algorithm where vertices start keeping track of super edges early and repeatedly discard their super edges and start over until the optimal mean  $\mu^*$  is found.

### 6.3 The Behavior of Value Iteration on Random Graphs

The algorithms we have developed require at least a linear number of iterations. This situation is similar to the situation for Karp's algorithm where it is observed that the unaltered algorithm basically takes  $O(mn)$  time irrespective of underlying graphs [52]. However we suspected that typically, relatively a few iterations of value iteration would suffice in finding the maximum mean. It is noted in previous work (see for example the references in [103]) that the value and policy iteration algorithms converge to optimal solutions in relatively "few" iterations.

We explored these questions on random graphs where every vertex has two out-edges, the end-vertex of each edge is chosen uniformly at random, and the reward of edges were

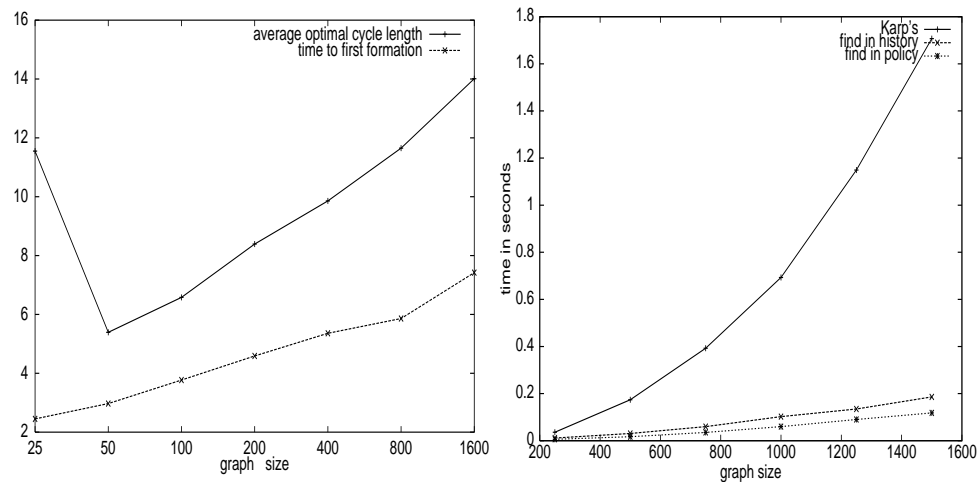


Figure 6.3: (a) Averages length of optimal cycles and the first iteration until an optimal cycle is found. (b) A comparison of the run times of Karp's algorithm versus value iteration using history walks and value iteration using visited policies.

chosen uniformly at random in  $[0, 1]$ . We tested on sparse graphs only, as the number of actions (edges) per vertex is usually small in MDP problems (a linear function of the number of vertices), which leads to problems on sparse graphs. The results for graphs of size  $n$  were obtained over samples of size maximum of 500 and  $n$ . In the plots shown, self-edges are not allowed (the neighboring vertex is chosen from the remaining  $n - 1$  vertices), but results are similar to the case where self-edges are allowed. Fig. 6.3a gives strong evidence that the growth of both the expected length of the optimal cycle and the expected number of iterations until an optimal cycle is first formed in a policy increase as a logarithmic function of graph size<sup>1</sup>. The distribution of the two random variables seems to decay exponentially. It would be fruitful to prove the observed logarithmic growth pattern of the expectations and the distributions under the given graph distributions. This would verify the observations, develop proof techniques, and would likely lead to obtaining other insights on properties that might lead to better algorithms.

The plots of Fig. 6.3a suggest alternative value iteration algorithms that attempt to

---

<sup>1</sup>The initial high average optimal length for size 25 is due to the small size of the graphs. With larger graph sizes, the limiting distribution of the random variable seems to kick in.

find an optimal cycle early. One way to compute the optimal mean quickly is to test the current visited policy periodically and compute whether the average reward of the cycle(s) in the policy is optimal. An efficient way<sup>2</sup> to test this is to subtract the candidate mean from all edge rewards, and use a Bellman-Ford (single source) shortest paths algorithm to detect the presence of positive cycles [1, 25]. If there are no positive cycles, the candidate mean is the maximum. While the shortest path detection algorithm also has  $O(mn)$  run time, empirically it is very efficient and may have linear time [1]. Fig. 6.3b verifies our expectations of fast run times. It shows the run times for two algorithms that use the shortest paths test periodically after  $\log n$  initial value iterations. These timings were performed on a PentiumIII/500 with 128 megabytes of RAM. One algorithm uses super edges and each time that the test fails in finding an optimal, it doubles the length of the cycle sought after and vertices reconstruct super edges from scratch, but continue with value iteration with their current values. Another algorithm, after  $\log n$  initial simple value iterations, simply checks the cycles formed in the policies visited. The expected run times of both algorithms are close to linear time, probably  $O(n \log n)$ , as expected.

Recently, a number of algorithms for the DMDP have been compared on a number of different sparse graphs in [30], and the authors observe that Howard's algorithm (a policy iteration algorithm for the DMDP problem) has the fastest run time. We discuss some of the algorithms and possibilities for improvements in the next section.

#### 6.4 Summary and Discussion

We showed that value iteration converges to an optimal cycle in a DMDP in  $\theta(n^2)$  iterations, regardless of the initial vertex values<sup>3</sup>. We also showed two algorithms, based on history walk, that solve the problem in  $O(mn)$  time. Previously, the known  $O(mn)$  algorithms were based on Karp's algorithms. Conjecture 6.2.9 gives another sense in which value iteration finds an optimal cycle, and is an important conjecture to verify to give a very simple  $O(mn)$  algorithm for solving the problem. We remark that in general, we have found it easier to

---

<sup>2</sup>See also the next section for another method of stopping the algorithm.

<sup>3</sup>Propagation of values in the forward direction of edges (just like the Forward-Propagate algorithm of Chapter 5) also results in algorithms with similar properties.

show that the cycles formed in history walks, versus cycles formed in the visited policies, are of good quality. We can show this not only for deterministic MDPs, but also for the more general problems<sup>4</sup> such as MDP(2). Consequently methods based on evaluating cycles found in the history of vertex edge choices during value iteration also give polynomial iterative algorithms for MDP(2) (and TVPI) problems, just as they do for the DMDP. Similarly, one may design algorithms based on super edges, parametrized by transition probabilities (slopes), expected rewards, and end vertices, for MDP(2) problems. We do not give the algorithms here, but the proofs of polynomial time are based on the quality of cycles found in the history walks and the analysis of the Newton’s method given in Chapter 5.

We also showed that the value iteration algorithms, with small modifications, find optimal cycles very quickly on a class of random graphs, and provided evidence why: the experiments suggest that the expected length of optimal cycles and first-time formation of optimal cycles grow only logarithmically as a function of graph size. It is useful to verify these observed properties rigorously. It would also be interesting to have similar concise statements on the fast convergence of value iteration and policy iteration on other similarly generated random MDP problems, such as MDP(2) and MDP(3).

Algorithms based on value iteration may be fast on practical problems as well. We will next explore algorithmic possibilities for fast empirical performance on DMDPs and then describe some of the theoretical open problems.

#### 6.4.1 Policy and Value Iteration in Practice

There are a number of algorithms that solve the DMDP algorithm. Dasdan et. al. [30] describe many such algorithms and compare their empirical performance on several graph types. Gaubert et. al. [19] give a policy iteration algorithm designed for the DMDP and more generally the minimum cost-to-time ratio cycle problem<sup>5</sup> and observe that the algorithm performs very well. The algorithm is a specializations of policy iterations for

---

<sup>4</sup>For the more general problems, the statements on cycle quality take into account the change in the highest-gain cycle as a function of the changing vertex values. We omit the precise statements.

<sup>5</sup>This problem is simply a generalization where edges have both rewards (or costs) and time costs or weights. The average reward of a cycle is the sum of the edge rewards of the cycle divided by the sum of the edge time costs.

average reward (stochastic) MDPs [33, 103]. See also Fox [41] for a similar algorithm for the minimum cost-to-time ratio cycle problem. In [30], it is observed that this policy iteration algorithm is fastest among the many algorithms tested on many graph types, and a pseudo-polynomial bound on its run time is given.

The policy iteration algorithm of Gaubert et. al. [19] works as follows for the DMDP problem. In each iteration, the mean of each cycle in the policy is computed. The vertices use two types of values in choosing their next out-edge (which creates the next improved policy). In each policy, each vertex has a path to some cycle and is assigned the average reward of the cycle it has a path to. If a vertex has a neighbor with higher such value (having a path to a better cycle in the current policy), it chooses the edge to the highest valued such neighbor. Let  $S$  denote the set of vertices that switch edges this way. If  $S$  is not empty, vertices in  $V - S$  keep their old edge and the next iteration begins. When  $S$  is empty, each vertex has a path to the best cycle among the cycles in the current policy that it can reach, and a second set of values is computed to improve the policy. Let  $x_v$  denote this second type of value for  $v$ .  $x_v$  is computed as follows. For each cycle  $c$  in the current policy, choose an arbitrary vertex  $s$  on  $c$ . Value  $x_s$  of  $s$  is assigned zero (or any other value). Any vertex having a path  $p$  in the current policy to  $s$  gets value  $r(p) - \lambda|p|$ , where<sup>6</sup>  $\lambda$  is the average reward of cycle  $c$  and  $r(p)$  is the total reward of  $p$ . Now, each vertex choose the edge with the highest value, where the value of a  $u-v$  edge is  $r(e) + x_v - \lambda$  (where  $\lambda$  is the average value of the cycle that  $v$  has a path to in the current policy). A new policy is created in this way and policy iteration is repeated until no change in policy.

We note two other variations for policy and value iteration on DMDPs. We know that DMDPs reduce to discounted MDPs with a discount factor sufficiently close to 1, see for example Zwick [125]. Policy iteration on discounted MDPs is simpler and policy evaluation still takes  $O(n)$ . This might speed up the basic loop above, but a drawback in practice may be numerical difficulties as the discount factor should be picked to be very close to 1. Value iteration on the discounted problem may be used too for faster empirical performance.

Another variation comes from our study of value iteration on the DMDP. If we add to

---

<sup>6</sup>This computation, sometimes called a “reduced-cost computation” in network flow algorithms, causes only positive gain cycles to appear in future visited policies.

each iteration of value iteration a step that computes the best cycle mean  $\mu$  found in the current policy (taking  $O(n)$  time), and adds to each vertex a self-edge (discards the old self-edge) with such reward  $\mu$ , then we can show that the cycles formed in the policies visited stay the same or improve from one iteration to another, as long as vertices always choose their self-edge in case of ties. We still need a quick test for when to stop the algorithm. Using a policy improvement test as described above or adding a discount factor and checking the switchability may provide a quicker test than using a shortest path negative/positive weight cycle detection algorithm.

It would be interesting to see why the policy improvement algorithm performs the fastest [30], and whether improvements can be made. One reason is perhaps due to its relatively low overhead. We also expect that the expected number of iterations of the algorithm to find the optimal policy is  $O(n \log n)$  on sparse random graphs, but can it be faster? How many strictly improving cycles are seen on average on the way to the optimal cycle? Compared to simple value iteration, policy improvement algorithms do  $O(n)$  extra computation in each iteration and simple value iteration or hybrids of value iteration and policy iteration (sometimes called modified policy iteration), should be faster empirically. The better performance of modified policy iteration is noted elsewhere in the context of solving general MDPs [103]. A hybrid would do value iteration for a number of iterations (perhaps started with randomly chosen initial vertex values), and in a second phase start some type of policy iteration. Again more experimentation is needed to tell whether such algorithmic variations make a significant difference.

A challenge is to design an algorithm that has expected time  $O(n)$  (or  $O(m)$  when dense) on random graphs, just as some shortest paths algorithms seem to take linear time in practice [1]. Bringing down the run time to expected  $O(n)$  may be hard, at least if we use value iteration on random graphs, due to the observed logarithmic growth of the expected length of the optimal cycle. However, policy evaluation may speed this up.

### 6.4.2 *Some Open Theoretical Problems*

We began this chapter with questions on formation of cycles as policy iteration progresses on MDP(2) problems. Do similar properties that we showed hold for the DMDP hold for MDP(2) or more generally TVPI, where edges have two parameters, one additive (the reward or the y-intercept) and one multiplicative (the transition probability or the slope)? Interestingly, if we assign positive multipliers to the edges and assign zero to the rewards (summands), and apply value iteration with some set of positive initial vertex values the arguments of this chapter work to show that value iteration converges to a cycle with highest geometric mean. What happens when each edge has both a multiplier and a summand?

A promising next step is to study value iteration on the discounted deterministic MDP. We may think of the (average reward) DMDP problem as an MDP(2) problem where edges all have slope equal to 1. In a discounted deterministic MDP, the slopes are equal but less than 1. We can prove that value iteration converges, in polynomial time, to the optimal cycles for value of the discount factor close to 1 (by making an argument based on small perturbation on the discount factor and using continuity), but can similar guarantees be given for other smaller values of the discount factor? Note that when the discount factor is 0, or sufficiently close to 0, the problem is trivial. We expect that close variants of value iteration are polynomial approximation algorithms for the discounted deterministic MDP. We also conjecture that policy iteration runs in  $O(mn)$  time in solving discounted deterministic MDPs.

Finally, another next step is to analyze policy iteration on DMDPs and the minimum cost-to-time ratio cycle problem as given in [19]. We expect insights from our analysis of value iteration and Newton's method developed here and by Radzik [107] will establish the algorithms as polynomial. For starters, we can see that arguments on adags and the freezing policy iteration algorithms of the previous sections apply in this context to give polynomial freezing policy iteration algorithms for the minimum cost-to-time ratio cycle problem.

## Chapter 7

## CONCLUSIONS

*A Hair, they say, divides the False and True,  
 Yes; and a single Alif were the clue,  
 Could you but find it—to the Treasure-House,  
 And peradventure to The Master too!*  
 (The Rubaiyat of Omar Khayyam)

We conclude the thesis, first by reviewing related research, then by summarizing thesis contributions, and finally by suggesting directions for future research.

### 7.1 *Related Work*

The first paper on the computational complexity of solving MDPs is by Papadimitriou and Tsitsiklis [98]. Both of the problems we address in our work, *i.e.* decidability of infinite-horizon POMDPs and algorithms for solving MDPs, are posed in their paper<sup>1</sup>.

#### 7.1.1 *POMDPs*

Early papers on POMDP problem formulations include those by Drake [36], Astrom [6], and Aoki [3]. Sondik and Smallwood [113, 115] are probably the first in addressing computational difficulties associated with solving POMDPs. Key references on applications and algorithms are the surveys [90, 79]. In recent years, many algorithms and heuristics have been developed for solving POMDPs, especially in the artificial intelligence research community.

Analysis of the complexity of solving MDPs began with the work of Papadimitriou and Tsitsiklis [98]. The computational complexity of finite-horizon control problems has received considerable attention recently, see for example [119, 13, 94, 45]. For the infinite-horizon case, two questions, the complexity of goal-state reachability with either nonzero probability or probability one, which

---

<sup>1</sup>On the subject of algorithms for MDPs, the authors comment that: “Let us note in passing that the problem of deriving a ‘clean’ polynomial time algorithm for this [MDP] problem, without using general linear programming or approximate techniques, is an important open question that has not been emphasized in the literature as much as it deserves.”

reduce to reachability computations and are decidable, had been studied by Alur et. al. [2] and Littman [76].

Our work on analyzing the computability of POMDPs was motivated by questions on the decidability of the probabilistic planning problem [74]. Littman also raises similar questions in his thesis [76]. Work on the computational complexity of planning in AI include [14, 38], which consider deterministic problems. Later work, for example [77, 78], considers finite plans (or problems in finite-horizon POMDPs).

Our work builds on the undecidability of the emptiness problem for probabilistic finite automata, first proved by Paz [99]. We found the undecidability proof of Lipton and Condon [24] more useful in our undecidability results, as apparent in Chapter 3. A short version of our work appears in [83].

Recent surveys [45, 11] are valuable sources on the computational complexity of probabilistic planning, MDPs and POMDPs, and other problems in systems and control together with some outstanding open problems and future directions.

### 7.1.2 MDPs

Work on MDPs has roots in the late nineteenth century, but in the 40's and 50's many researchers began working on various problems in the framework and popularized the problems [103]. Bellman coined the name, “Markov decision process,” and collected the common model ingredients [103]. A valuable early reference on dynamic programming is Bellman's book [7].

#### *Algorithms*

Value iteration, similar to the Bellman-Ford label correcting algorithm for shortest paths, has roots in the Jacobi iteration methods for algebraic structures such as closed semi-rings [60]. Its use in MDPs probably originates in Shapley's work [110] demonstrating the existence of solutions in the two-player game versions of MDPs. The policy iteration procedure in a general form is attributed to Bellman, who called it “approximation in policy space” and associated it with dynamic programming (see for example [7]), and to Howard [58], who gave an explicit policy iteration procedure for solving discounted MDPs.

#### *MDPs and Linear Programs*

Formulation of MDPs as linear programs appears as early as in the work of D'Epenoux [34]. It is shown in [98] that, similar to linear programs, MDPs are P-complete, so they are likely inherently sequential (but see the section on TVPI below). The linear programming problem (LP) was first

shown solvable in polynomial time by Khachian, who used an exterior-point ellipsoid method [69]. Books on linear programming problems include [109, 97].

The connection between pivoting in simplex algorithms for solving LPs and policy improvement appears in early work, for example in [31, 35]. A simplex algorithm is a local (or neighborhood) search algorithm for the linear programming (LP) problem. See, for example [97], for a discussion of the close relationship between local search in combinatorial optimization and the simplex algorithm. Denardo [32] has a section titled, “linear programming is policy iteration,” which highlights the connection between policy improvement and the simplex algorithms. The choice of vertex to switch in policy improvement corresponds to a pivot rule in a simplex algorithm. The LP problem is a convex optimization problem and has no local optima, or more precisely a local optimum is also a global optimum. Thus, the convergence of the simplex algorithm to the optimum is not an issue—as long as the pivots (the local moves) make progress—as much as the speed of convergence is.

Several examples that are “bad,” *i.e.* cause an exponential number of iterations for some policy improvement methods are given by Melekopoglou and Condon [89, 88] (see Section 4.5.2 for a related discussion). The policy improvement algorithms studied have a flavor similar to simplex algorithms using “selective” pivoting rules. Interestingly, similar to the case for policy iteration where switching all switchable vertices is a natural and inexpensive move, Zadeh [124] notes that “.. none of the [bad] examples’ pivot rules are typically used in practice, either because of computational requirements or due to lack of even-handed movement through the column set.” He also notes the common property that all the examples “hide” the “good” moves (local improvements that lead to fast convergence) in that in each iteration some other move exists that looks better with respect to the pivot rule used, so that exponentially many pivots, leading to small progress, are made before a good pivot is made. In the context of MDPs, a vertex (or a group of vertices) whose switch would lead to fast progress remains switchable but is not switched for an exponential number of iterations [89, 88]. Even the simpler shortest paths problems have bad examples (due to Edmonds and appearing in [27]).

Recommended algorithmic variations to get around the bad examples all guarantee that all columns (choice of actions or vertices in MDPs) are periodically checked and used if they improve the objective function. We may call these “scan” simplex algorithms. These simplex variations are somewhat similar to successful local search algorithms which make randomized moves or even (locally) bad moves to avoid getting stuck in local optima. Note that in the LP context, we seek only to avoid relatively slow convergence. Kalai [65] and Ludwig [80] give subexponential algorithms that select their moves randomly (in MDPs, the vertex to switch) for LPs and MDPs. It is noted in [124] that given that any of the scan simplex variations are polynomial, proofs of polynomial

run-time might be extremely hard as they would reduce the upper bound on the diameter of convex polytopes [70] (at the time exponential and currently quasi-polynomial [65]) to a polynomial in the number of facets of the polytope and the dimension. However, in the case of the MDP, from any policy one can obtain an optimal policy with at most  $n$  changes of choices of actions in the policy, where each change creates a better policy. In other words, the diameter of corresponding polytope is linearly bounded. The dual of the LP formulation 4.1 in Chapter 4 is a Leontief substitution system, and the corresponding polyhedra satisfy the Hirsch conjecture on the upper bound on the diameter [46, 70]. Furthermore, value and policy iteration algorithms have been shown to be pseudo-polynomial on MDP problems and game variants [119, 125, 76], while this is not known for scanning simplex algorithms on general linear programs. To summarize, we are likely faced with analyzing a promising pivoting simplex algorithm on a simpler problem than a general LP.

### *Applicability of the Iterative Algorithms to Related Problems*

Policy and value iteration methods also solve two-player game versions of MDPs for which no linear programming formulation is known. The games where players alternate making moves, called simple stochastic games, are known to be in P and coNP but not known to be in P [22]. An algorithm similar to policy iteration is given by Karp and Hoffman [56], and Condon [23] explores several algorithms, and shows a few natural algorithms incorrect. Even some deterministic versions of the two-player games (basically the two player deterministic average reward MDP problem) are not known to be in P [125, 100].

Value and policy iteration are used in many other related problems such as POMDPs and factored MDPs [17, 50, 73].

The applicability and the common use of policy iteration, the aforementioned observations on the algorithm and its relation to simplex algorithms, and its simplicity and fit to the problems, has motivated this work and the work in [89, 76, 84] to analyze its run-time.

#### *7.1.3 MDP(2) and TVPI*

We expect that the MDP(2) problem is simpler than the MDP(3) problem in a number of ways. For example, the partially observable variant of the MDP(2) (actions restricted to be of the MDP(2) type) is decidable while the variant for the MDP(3), the POMDP, is not. The MDP(2) problem is equivalent to the generalized shortest path problem [95] and is a special feasibility problem in linear systems with two variables per inequality (TVPI). It is probably the case that the MDP(2) problem is in NC, meaning that it can be efficiently solved in parallel. In [81], the same open question is

raised about the TVPI problem. Linear optimization over TVPI is P-complete [81]. Deterministic MDP problems (average reward or discounted) are in NC [98].

Algorithms for the TVPI problem were developed partly in the search for polynomial algorithms for the general linear programming problem [5]. It was shown by Megiddo [87] that the TVPI problem could be solved in strongly polynomial time using parametric techniques (see also [86]). The strongly polynomial algorithms for TVPI given in [55, 20] and for the the generalized shortest path problem [95] are all based on the parametric binary search techniques of Megiddo [87].

There is an interesting correspondence between satisfiability problems and linear feasibility problems as pointed out by Hochbaum and Naor [55], and to which we add a little here. The algorithm in [55] is based on the Fourier-Motzkin variable elimination method for solving linear programs, whose analogue in logic is the resolution method. Resolution can be efficiently implemented on the 2-SAT problem and it is shown that the Fourier-Motzkin method can be efficiently implemented, using parametric techniques, to solve TVPI. The monotone feasibility problem described in Chapter 4 is a three-variable feasibility problem, but corresponds to Horn clauses in logic which can also be solved efficiently. We expect that monotone linear feasibility problems are also easier to solve than general feasibility problems. We remark that general linear feasibility problems are as hard as linear programming problems as far as polynomial time computations are concerned.

Monotone TVPI problems find applications in solving generalized network flow problems [21, 121, 95]. We expect that other LPs and problem variants (for instance games) can also be solved by repeatedly solving some monotone sub-problems, therefore studying fast algorithms for the monotone cases should prove fruitful.

#### *7.1.4 Policy Iteration and Newton's Method*

Connections between policy iteration and Newton's method has been noted by Kalaba [64] and Puterman and Brumelle [104] and by Pollatschek and Avi-Itzhak in the context of games [102]. However its computational complexity has not been analyzed, and results such as fast (local) convergence are not sufficient for showing polynomial running time, as pointed out by Littman [76] and Blondel and Tsitsiklis [11] and here for example.

Newton's method, also known as the Newton-Raphson method, is one of the main methods for solving (parametric versions of) fractional programming problems [59, 107, 108]. In this context, it is also known as the Dinkelbach method. Polynomial convergence of Newton's method on fractional linear combinatorial problems include [68, 106], for uniform weights, such as the maximum mean cycle problem, and [107, 111], for non-uniform weights, such as the minimum cost-to-time ratio cycle

problem.

### 7.1.5 *The Deterministic Average Reward MDP Problem*

The deterministic average reward MDP problem is a special fractional linear programming problem. It can be solved in  $O(\min\{nm, n^{1/2}m \log(nC)\})$ . The first bound is due to Karp [66], and the next is due to Orlin and Ahuja [67]. Applications of the problem are numerous and include network flow problems and systems performance analysis. See for example [1, 30, 52].

Several other algorithms have been designed for the DMDP. To the best of our knowledge, all the algorithms with known  $O(mn)$  run time prior to our work, for example the algorithms proposed by Dasdan and Gupta [29] and Hartman and Orlin [52], were based on Karp's algorithm. A substantial empirical examination of many DMDP algorithms is carried out by Dasdan et. al. [30] on many sparse graphs and circuits. It is interesting to note that a multi-chain policy iteration algorithm [32, 103] described in [19] (also referred to as Howard's algorithm [19, 30]) for the minimum cost-to-time ratio cycle problem, turns out to be the fastest on all the graphs tested. Dasdan et. al. [30] give upper bounds on the run time of the policy improvement algorithm that is pseudo-polynomial. We expect that our results on value iteration and the connection between policy iteration and Newton's method will show that these policy improvement algorithms are polynomial for the mean cycle problem and the minimum cost-to-time cycle ratio problem, and hybrids of value and policy iteration may even be faster.

## 7.2 *Thesis Contributions and Future Work*

Our work in this thesis concentrated on determining the computational complexity of several problems in (infinite-horizon) MDPs. Below we summarize our contributions:

- We established that many basic problems in partially observable MDPs (POMDPs) are undecidable. These include determining whether a plan exists with success probability exceeding a desired threshold in a probabilistic planning problem, or whether a policy exists, again with value exceeding a desired threshold, in a discounted infinite-horizon POMDP problem.
- We established that versions of policy iteration are polynomial on a subclass of MDP problems with two variables per inequality when formulated as a linear programming problem. We believe that the simplification to the MDP(2) was an important step in identifying the aspects of policy iteration, particularly its behavior on a parametric version of the problem and its connection to Newton's method, which led to proofs of efficiency. We also showed that

policy iteration algorithms solve related TVPI and deterministic discounted MDP problems efficiently.

- We showed that the basic value iteration procedure takes a polynomial number of iterations in converging to an optimal (average reward) cycle in a DMDP problem, and gave a few algorithms that have  $O(mn)$  time on the problem. We gave evidence that algorithms based on value iteration are very efficient on random graphs.

Many open problems are discussed throughout the thesis. Next, we list a few general future directions on algorithm development.

We believe that policy iteration is polynomial on MDPs, and expect that the promising way to establish this is through its connection to Newton’s method, and thus using the geometric and analytic constraints on the method’s convergence to the optimal. Relevant questions on performance of the algorithm in [56] for stochastic games [23] may also be answered using similar techniques. Considering the simpler deterministic games [125, 100] and designing and analyzing policy improvement algorithms for them can also be fruitful here.

It would also be very interesting to determine whether the analysis of policy iteration on MDPs can help in any way in analyzing “scanning” simplex algorithms on linear programs. A possible next step is to analyze these simplex algorithms on simpler problems such as linear optimization problems over systems of two variables per inequality (*i.e.* linear optimization over TVPI systems) or over Leontief-substitution systems. Recently, Wayne [121] gave polynomial combinatorial algorithms for linear optimization over TVPI systems; thus these problems may provide a simpler more tractable “testbed” to study the simplex algorithms, develop analysis techniques, and verify and narrow hypotheses.

Finally, we believe that the family of policy and value iteration algorithms are a source of many efficient methods for solving MDPs and related problems in practice. Fast convergence of the algorithms have been reported in several computational studies [30, 118, 51, 103]. More studies in this direction and comparison of different algorithms on various problems will be essential in designing algorithms with superior empirical performance.

## BIBLIOGRAPHY

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [2] R. Alur, C. Couroubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proc. of 27th STOC*, pages 363–372, 1995.
- [3] M. Aoki. Optimal control of partially observable Markovian systems. *J. Franklin Inst.*, 280:367–386, 1965.
- [4] B. Aspvall. *Efficient algorithms for certain satisfiability and linear programming problems*. PhD thesis, Stanford University, 1980.
- [5] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 9(4):827–845, November 1980.
- [6] K. J. Astrom. Optimal control of markov processes with incomplete state information. *J. Math. Anal. Appl.*, 10:174–205, 1965.
- [7] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [8] R. E. Bellman. *Eye of the Hurricane, an autobiography*. World Scientific, 1984.
- [9] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice Hall, 1987.
- [10] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.
- [11] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. To appear in *Automatica*, 2000.
- [12] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 2000.
- [13] D. Burago, M. De Rougemont, and A. Slissenko. On the complexity of partially observed Markov decision processes. *Theoretical Computer Science*, pages 161–183, 1996.

- [14] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:161–204, 1994.
- [15] P. J. Carstensen. Parametric cost shortest path problems. Unpublished Bellcore memo, available from the Literaturstelle at the Inst. für Ökonometrie und Operations Research, U. Bonn, 8 1984.
- [16] P. J. Cartstensen. *The complexity of some problems in parametric linear and combinatorial programming*. PhD thesis, Department of Mathematics, University of Michigan, 1983.
- [17] A. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple fast exact algorithm for partially observable markov decision processes. In *Proc. Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 54–61, 1997.
- [18] A. R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, 1998.
- [19] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proc. IFAC Conf. on Systems Structure and Control*, pages 667–74, 1998.
- [20] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23(6):1313–1347, December 1994.
- [21] E. Cohen and N. Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64:325–336, 1994.
- [22] A. Condon. The complexity of simple stochastic games. *Information and Computation*, 96(2):203–224, February 1992.
- [23] A. Condon. On algorithms for simple stochastic games. In *Advances in computational complexity theory*, volume 13 of *DIMACS series in discrete mathematics and theoretical computer science*. American Mathematical Society, 1993.
- [24] A. Condon and R. Lipton. On the complexity of space bounded interactive proofs. In *30th Annual Symposium on Foundations of Computer Science*, 1989.
- [25] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.
- [26] R. W. Cottle and A. F. Veinott, Jr. Polyhedral sets having a least element. *Mathematical Programming*, 3:238–249, 1972.

- [27] W. H. Cunningham. Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 2:196–208, 1979.
- [28] T. Darrell and A. Pentland. Active gesture recognition using partially observable Markov decision processes. In *Proceedings of the Thirteenth IEEE International Conference on Pattern Recognition (ICPR)*, 1996.
- [29] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 17(10):889–99, 1998.
- [30] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimal cycle mean and optimum cost to time ratio problems. In *Proc. 1999 Design and Automation Conference*, pages 37–42, 1999.
- [31] G. DeGhellinck. Les problèmes de décisions séquentielles. *Cahiers du Centre de Recherche Opérationnelle*, 2:161–179, 1960. In French.
- [32] E. V. Denardo. *Dynamic Programming: Models and Applications*. Prentice-Hall, 1982.
- [33] E. V. Denardo and B. Fox. Multichain markov renewal programs. *SIAM J. on Computing*, 16:468–487, 1968.
- [34] F. D’Epenoux. Sur un problème de production et de stockage dans l’aléatoire. *Rev. Fr. Autom. Inf. Recherche Op.*, 14(3), 1963. (Engl trans.) in *Man. Sci.*, 10, 98-108 (1963).
- [35] C. Derman. *Finite State Markov Decision Processes*. Academic, 1970.
- [36] A. W. Drake. *Observation of a Markov Process Through a Noisy Channel*. PhD thesis, Massachusetts Institute of Technology, 1962.
- [37] H. Edelsbrunner, G. Rote, and E. Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theoretical Computer Science*, 66(2):157–180, 20 August 1989.
- [38] K. Erol, D. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.
- [39] H. R. Everett. *Sensors For Mobile Robots*. A. K. Peters, 1995.
- [40] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.
- [41] B. Fox. Finding minimal cost-time ratio circuits. *Operations Research*, 17:546–551, 1969.

- [42] R. Freivalds. Probabilistic two way machines. In *Proc. International Symposium on Mathematical Foundations of Computer Science*, volume 118, pages 33–45. Springer-Verlag, 1981.
- [43] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by cancelling negative cycles. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 388–397, 1988. Full paper in *Journal of ACM* **36**(1989), 873-886.
- [44] A. V. Goldberg and R. E. Tarjan. Solving the minimum-cost flow problem by successive approximations. In *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pages 136–146, 1988. Full paper in *Mathematics of Operations Research* **15**(1990), 430-466.
- [45] J. Goldsmith and M. Mundhenk. Complexity issues in Markov decision processes. In *Proc. IEEE conference on Computational Complexity*, pages 272–80, 1998.
- [46] R. C. Grinold. The Hirsch conjecture in Leontief substitution systems. *SIAM J. Appl. Mathematics*, 21:483–485, 1970.
- [47] D. Gusfield. Parametric combinatorial computing and a problem of program module distribution. *J. ACM*, 30(3):551–563, July 1983.
- [48] D. Gusfield and R.W. Irving. *The Stable Marriage Problem*. MIT Press, 1989.
- [49] E. A. Hansen. *Finite Memory Control of Partially Observable Systems*. PhD thesis, University of Massachusetts, Amherst, 1998.
- [50] E. A. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *Advances in Neural Information Processing systems*, volume 10, 1998.
- [51] R. Hartley, A. C. Lavercombe, and L. C. Thomas. Computational comparison of policy iteration algorithms for discounted markov decision processes. *Comput. Res.*, 13:411–420, 1986.
- [52] M. Hartmann and J. Orlin. Finding minimum cost to time ratio cycles with small integral transit times. *Networks*, 23:567–74, 1993.
- [53] M. Hauskrecht and H. Fraser. Planning medical therapy using partially observable Markov decision processes. In *Proceedings of the ninth international workshop on principles of diagnosis (DX-98)*, 1998.
- [54] D. S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62(1):69–83, October 1993.

- [55] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.
- [56] A. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5), 1966.
- [57] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [58] R. Howard. *Dynamic Programming and Markov Decision Processes*. M.I.T. Press, 1960.
- [59] T. Ibaraki. Parametric approaches to fractional programs. *Math. Programming*, 26:345–362, 1983.
- [60] C. G. J. Jacobi. Über eine neue auflösungsart der bei der methode der kleinsten quadrate vorkommenden linearen gleichungen. *Astronomische Nachrichten*, 22:297–303, 1945. In German.
- [61] M. Jiang. *Partially Observable Markov Decision Process Models for Structural Management Policies and Design*. PhD thesis, John Hopkins University, 1994.
- [62] A. F. Veinott Jr. Representation of general and polyhedral sublattices of product spaces. *Linear Algebra Applications*, 114/115:681–704, 1989.
- [63] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Artificial Intelligence Research*, 4:237–285, 1996.
- [64] R. Kalaba. On nonlinear differential equations, the maximum operation and monotone convergence. *J. Math. Mech.*, 8:519–574, 1959.
- [65] G. Kalai. A subexponential randomized simplex algorithm. In *24th annual ACM STOC*, pages 475–482, 1992.
- [66] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.
- [67] R. M. Karp and J. B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3:37–45, 1981.
- [68] A. V. Karzanov. On minimal mean cuts and circuits in a digraph. In *Methods for Solving Operator Equations*, pages 72–83. Yaroslavl State Univ., 1985. In Russian.
- [69] L. G. Khachian. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, pages 191–194, 1979.

- [70] V. Klee and P. Kleinschmidt. The  $d$ -step conjecture and its relatives. *Mathematics of Operations Research*, 12(4):718–755, 1987.
- [71] S. Koenig, R. Goodwin, and R. Simmons. Robot navigation with Markov models: a framework for path planning and learning with limited computational resources. In *Reasoning with Uncertainty in Robotics*, Lecture Notes in Artificial Intelligence, pages 322–337. Springer Verlag, 1995.
- [72] S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable Markov decision process model. In *Artificial intelligence based Mobile robotics: case studies of successful robot systems*. MIT Press, Cambridge MA, 1997.
- [73] D. Koller and R. E. Parr. Policy iteration for factored MDPs. In *proc. Uncertainty in AI*, 2000. To appear.
- [74] N. Kushmerick, S. Hanks, and D. S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.
- [75] D. E. Lane. A partially observed model of decision making by fisherman. *Operations Research*, 37:240–254, 1989.
- [76] M. L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown, 1996.
- [77] M. L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the 14th National Conference on AI*. AAAI Press, 1997.
- [78] M. L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Artificial Intelligence Research*, 9:1–36, 1998.
- [79] W. Lovejoy. A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, pages 47–66, 1991.
- [80] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and computation*, 117:151–155, 1995.
- [81] G. S. Lueker, N. Megiddo, and V. Ramachandran. Linear programming with two variables per inequality in poly-log time. *SIAM Journal on Computing*, 19(6):1000–1010, December 1990.
- [82] O. Madani. On constraints on the search path of policy iteration. Technical Report UW-CSE-99-04-01, University of Washington, 1998.

- [83] O. Madani, S. Hanks, and A. Condon. On the computability of infinite-horizon partially observable Markov decision processes. In *Proc. of 16th national conference in artificial intelligence*, pages 541–548, 1999.
- [84] Y. Mansour and S. Singh. On the complexity of policy iteration. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.
- [85] R. A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.
- [86] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.*, 30:852–865, 1983.
- [87] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12(2):347–353, May 1983.
- [88] M. Melekopoglou and A. Condon. On the complexity of the policy iteration algorithm. Technical report, University of Wisconsin-Madison, 1990.
- [89] M. Melekopoglou and A. Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing*, 6(2), 1994.
- [90] G. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–15, 1982.
- [91] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [92] M. Mundhenk, J. Goldsmith, and E. Allender. The complexity of policy existence problem for partially-observable finite-horizon Markov decision processes. In *Mathematical Foundations of Computer Science*, pages 129–38, 1997.
- [93] M. Mundhenk, J. Goldsmith, and Eric Allender. The complexity of policy evaluation for finite-horizon partially-observable Markov decision processes. In *Proc. 22nd Mathematical Foundations of Computer Science*, pages 129–138, 1997.
- [94] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity results for finite-horizon markov decision processs problems. *Journal of the ACM*, 2000. To appear.
- [95] J. D. Oldham. Combinatorial algorithms for generalized flow problems. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Mathematics*, pages 704–714, 1999.
- [96] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

- [97] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.
- [98] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, August 1987.
- [99] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.
- [100] N. N. Piasaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–28, 1999.
- [101] W. P. Pierskalla and J. A. Voelker. A survey of maintenance models: The control and surveillance of deteriorating systems. *Naval Research Logistics Quarterly*, 23:353–388, 1976.
- [102] M. Pollatschek and B. Avi-Itzhak. Algorithms for stochastic games with geometrical interpretation. *Management Science*, 15:399–413, 1969.
- [103] M. L. Puterman. *Markov Decision Processes*. Wiley Inter-science, 1994.
- [104] M. L. Puterman and S. L. Brumelle. The analytic theory of policy iteration. In M. L. Puterman, editor, *Dynamic Programming and its Applications*, pages 91–114. Academic, 1978.
- [105] M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
- [106] T. Radzik. Minimizing capacity violations in a transshipment network. In *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, 1992.
- [107] T. Radzik. Newton’s method for fractional combinatorial optimization. In *Proceedings, 33rd Annual Symposium on Foundations of Computer Science*, pages 659–669, 1992.
- [108] T. Radzik. Fractional combinatorial optimization. In D. Du and P. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1. Kluwer Academic, 1998.
- [109] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, 1986.
- [110] L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. U.S.A.*, 39:1095–1100, 1953.
- [111] M. Shigeno, Y. Saruwatari, and T. Matsui. An algorithm for fractional assignment problems. *Discrete-Applied-Mathematics*, 56(2-3):333–43, 1995.
- [112] R. Shostack. Deciding linear inequalities by computing loop residues. *J. ACM*, 28:769–779, 1981.
- [113] R. Smallwood and E. J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

- [114] E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.
- [115] E. J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.
- [116] C. Striebel. Sufficient statistics in the optimum control of stochastic systems. *Journal of Mathematical Analysis And Applications*, 12:576–592, 1965.
- [117] L. C. Thomas, D. P. Gaver, and P. A. Jacobs. Inspection models and their applications. *IMA Journal of Mathematics applied in business and Industry*, 3:283–303, 1991.
- [118] L. C. Thomas, R. Hartley, and A. C. Lavercombe. Computational comparison of value iteration algorithms for discounted markov decision processes. *Operations Research Letters*, 2:72–76, 1983.
- [119] P. Tseng. Solving  $H$ -horizon stationary Markov decision process in time proportional to  $\log(H)$ . *Operations Research Letters*, 9(5):287–297, 1990.
- [120] R. Washington. Uncertainty and real-time therapy planning: Incremental Markov model approaches. In *Proceedings of the AAAI symposium on Artificial Intelligence in Medicine*, 1996.
- [121] K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 11–18, 1999.
- [122] D. J. White. Further real applications of Markov decision processes. *Interfaces*, 18:55–61, September 1988.
- [123] D. J. White. *Markov Decision Processes*. Wiley, 1993.
- [124] N. Zadeh. What is the worst case behavior of the simplex algorithm? Technical Report 27, Stanford, 1980.
- [125] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, May 1996.

## Appendix A

### SOME PROPERTIES OF OPTIMAL VALUE FUNCTIONS AND POLICIES IN UMDPS

Here, we include proofs for a few properties of the optimal value functions  $V^*$  and optimal policies  $p^*$  for POMDPs, mostly under the discounted criterion. These properties include convexity of  $V^*$  and closure of optimal action regions. The proofs are included for purposes of illustration and are standard in POMDP literature. Books such as [9, 123] are valuable references on POMDPs. We also include the notions of contraction mappings and fixed-points, and some of the associated proofs as they are simple and aid in proofs of existence of desired properties, for example existence of optimal policies and the convexity of  $V^*$  under the discounted criterion.

#### **A.1 Contraction Mappings and the Fixed-Point Property**

For a real-valued function  $V$  on a set  $X$ , let  $\|V\| = \sup_{x \in X} |V(x)|$ . Let  $B(X)$  denote the set of bounded real-valued function on  $X$ : for  $V \in B(X)$ ,  $\|V\| < M$  for some  $M \in \mathbb{R}$ .

A mapping  $H : B(X) \rightarrow B(X)$  is said to be a contraction mapping if  $\exists \beta \in \mathbb{R}, 0 \leq \beta < 1$  such that  $\forall V \in B(X), \forall V' \in B(X)$ :

$$\|HV - HV'\| \leq \beta \|V - V'\|.$$

Let  $H^k$  be the mapping  $H$  composed with itself  $k$  times:  $H^0V = V$ , and  $H^kV = H(H^{k-1}V)$ ,  $k \geq 1$ . The following Theorem is a version of Banach fixed-point theorem for contraction mappings. It says that applying a contraction mapping over and over to a function and its images leads to convergence to a unique function.

**Theorem A.1.1** *If  $H : B(X) \rightarrow B(X)$  is a contraction mapping, then  $\exists V^* \in B(X)$  such that  $HV^* = V^*$  and  $\forall V \in B(X), \lim_{k \rightarrow \infty} \|H^kV - V^*\| = 0$ .*

**Proof.** Let  $V \in B(X)$ . For  $k \geq 0$ , let  $d_k = \|H^{k+1}V - H^kV\|$ . We have  $d_0 \in \mathbb{R}$ , and from the contraction assumption,  $\exists \beta \in \mathbb{R}, 0 \leq \beta < 1, d_k \leq \beta^k d_0$ . Therefore  $\lim_{k \rightarrow \infty} d_k = 0$ . We next show that  $\forall x \in X$ , the sequence  $\{(H^kV)(x)\}$  is bounded, from which it follows that  $\forall x \in X$ , the sequence

$\{(H^k V)(x)\}$  is convergent. We have:

$$\begin{aligned}
(\text{from contraction}) \forall k \geq 0, \|H^{k+1}V - HV\| &\leq \beta \|H^k V - V\| \Rightarrow \\
\forall k \geq 1, \|H^{k+1}V\| &\leq \beta (\|H^k V\| + \|V\|) + \|HV\| \Rightarrow \\
\forall k \geq 1, \|H^{k+1}V\| &\leq \left( \sum_{0 \leq i \leq k} \beta^i \right) \|HV\| + \left( \sum_{1 \leq i \leq k} \beta^i \right) \|V\| \Rightarrow \\
\forall k \geq 0, \|H^k V\| &\leq \frac{\|HV\| + \|V\|}{1 - \beta},
\end{aligned}$$

and  $V$  and  $HV$  are bounded.

Next we show that the function  $V^* \in B(X)$ , defined by  $V^*(x) = \lim_{k \rightarrow \infty} (H^k V)(x)$ , is a fixed-point of  $H$ . But first, we have  $\forall \epsilon > 0, \exists N$ , such that:

$$\begin{aligned}
\|H^{N+1}V - H^N V\| &\leq \epsilon \Rightarrow \\
\forall k \geq 0, \|H^{N+k+1}V - H^{N+k}V\| &\leq \beta^k \epsilon \Rightarrow \\
\forall k \geq 0, \forall l \geq 0, \|H^{N+k+l}V - H^{N+k}V\| &\leq \sum_{0 \leq i < l} \beta^i \epsilon \Rightarrow \\
\forall k \geq 0, \lim_{l \rightarrow \infty} \|H^{N+k+l}V - H^{N+k}V\| &\leq \frac{\epsilon}{1 - \beta} \Rightarrow \\
\forall k \geq 0, \|V^* - H^{N+k}V\| &\leq \frac{\epsilon}{1 - \beta},
\end{aligned}$$

which shows that  $\{H^k V\}$  converges uniformly to  $V^*$ . Therefore,  $\forall \epsilon > 0$  we can find an integer  $N$ , such that  $\forall k \geq N, \|V^* - H^k V\| \leq \epsilon$ , and  $\|H^{k+1}V - H^k V\| < \epsilon$ . For such  $k$ ,  $\|H^{k+1}V - HV^*\| \leq \beta \|V^* - H^k V\| \leq \epsilon$ , and we have  $\|HV^* - V^*\| \leq \|H^k V - V^*\| + \|H^{k+1}V - H^k V\| + \|H^{k+1}V - HV^*\| \leq 3\epsilon$ . Therefore  $\forall \epsilon > 0, \|HV^* - V^*\| \leq \epsilon \Rightarrow HV^* = V^*$ .

We have shown that for arbitrary  $V \in B(X)$ ,  $H^k V$  converges uniformly to a fixed-point  $V^*$  of  $H$ . The last statement of the theorem follows if there is a unique fixed-point, which can be shown using the contraction property, again: Let  $V_1^*$  and  $V_2^*$  be two fixed-points of  $H$ . Then  $\|HV_1^* - HV_2^*\| \leq \beta \|V_1^* - V_2^*\|$ , but  $HV_1^* = V_1^*$ , and  $HV_2^* = V_2^*$ , so  $\|HV_1^* - HV_2^*\| = \|V_1^* - V_2^*\| \leq \beta \|V_1^* - V_2^*\| \Rightarrow \|V_1^* - V_2^*\| = 0$ .  $\square$

## A.2 Optimal Policies and Value Functions

Consider the set  $P$  of policies of the form  $^1 p : X \rightarrow \Sigma$ , where  $X$  is the set of probability distributions over the states of a POMDP and  $\Sigma$  is the set of actions. Define the operators  $H_p : B(X) \rightarrow B(X)$ ,

---

<sup>1</sup>It can be shown that an optimal policy need only be a (stationary) function of distribution over states, and need not be randomized.

where  $p \in P$ , and  $H^* : B(X) \rightarrow B(X)$  (the Bellman operator) as:

$$H_p V(x) = xR_{p(x)} + \beta V(xM_{p(x)}) \quad (\text{A.1})$$

$$H^* V(x) = \max_{a \in \Sigma} [xR_a + \beta V(xM_a)]. \quad (\text{A.2})$$

These operators originate from expressing value functions for a horizon of  $k$  (for a policy  $p$  or the optimal policy) in terms of the value functions for the horizon of  $k - 1$ . The operators are a short hand and eliminate writing pointwise equations. They also allow us to make connections with contraction mappings of the previous section.

**Lemma A.2.1 (contraction property)**  $H_p$  and  $H^*$  are contraction mappings.

**Proof.** Let  $V$  and  $V'$  be in  $B(X)$ .  $\forall x \in X, \|H_p V(x) - H_p V'(x)\| = \beta \|V(xM_{p(x)}) - V'(xM_{p(x)})\| \leq \beta \|V - V'\| \Rightarrow \|H_p V - H_p V'\| \leq \beta \|V - V'\|$ . A similar argument works for the case of operator  $H^*$ .  $\square$

It follows from theorem A.1.1 that both  $H_p$  and  $H^*$  have unique fixed-points, which we denote by  $V_p$  and  $V^*$  respectively, and that  $\forall V \in B(X), H_p^k V$  converges to  $V_p$  and  $H^{*k} V$  converges to  $V^*$ .

For  $V, V' \in B(X)$ , we write  $V \leq V'$  when  $\forall x \in X, V(x) \leq V'(x)$ . The next lemma shows that  $\forall p \in P, V_p \leq V^*$ . Hence the  $\leq$  relation induces a partial ordering on the space of fixed-point (and finite horizon) value functions, where  $V^*$  is the top point.

**Lemma A.2.2 (monotonicity)** Let  $V, V' \in B(X), p \in P$ . Then

$$V \leq V' \Rightarrow H_p V \leq H_p V' \text{ and } H^* V \leq H^* V'.$$

**Proof.** Take any  $x \in X$ . Say action  $a$  maximizes the right side of eq. A.2. But  $H^* V(x) = xR(a) + \beta V(xM_a) \leq xR(a) + \beta V'(xM_a)$ , since  $V(xM_a) \leq V'(xM_a)$ , and  $xR(a) + \beta V'(xM_a) \leq \max_{a \in \Sigma} xR(a) + \beta V'(xM_a) = (H^* V')(x)$ . The argument for  $H_p$  is simpler:  $H_p V(x) = xR_{p(x)} + \beta V(xM_{p(x)}) \leq xR_{p(x)} + \beta V'(xM_{p(x)}) = (H_p V')(x)$ .  $\square$

**Corollary A.2.3**  $\forall V \in B(X), \forall p \in P, \forall k > 0, H_p^k V \leq H^{*k} V$ .

**Proof.** Proof by induction. Not hard to see for  $k \leq 1$ . If for some  $k \geq 0, H_p^k V \leq H^{*k} V$  then  $H_p^{k+1} V = H_p(H_p^k V) \leq H_p(H^{*k} V)$  from the monotonicity of  $H_p$ . But by definition of  $H_p$  and  $H^*$ ,  $H_p(H^{*k} V) \leq H^*(H^{*k} V) = H^{*k+1} V$ .  $\square$

**Corollary A.2.4**  $\forall p \in P, V_p \leq V^*$ .

**Proof.** Let  $V \in B(x)$ . The corollary follows from the convergence of  $H_p^k V$  to  $V_p$  and of  $H^k V$  to  $V^*$ : Let  $x \in X$ , then

$$V_p(x) - V^*(x) = [V_p(x) - H_p^k V(x)] + [H_p^k V(x) - H^{*k} V(x)] + [H^{*k} V(x) - V^*(x)], \text{ for any } k \geq 0.$$

But  $\lim_{k \rightarrow \infty} [V_p(x) - H_p^k V(x)] = \lim_{k \rightarrow \infty} [H^{*k} V(x) - V^*(x)] = 0$ , while  $H_p^k V \leq H^{*k} V$ .  $\square$

It follows that  $V^*(x)$  is the best value that a policy can achieve at a point  $x$  over the infinite horizon. Notice that the action that the operator  $H^*$  prescribes at a point (the action maximizing eq. A.2) can change with different functions  $H^{*k} V$  (as  $k$  increases). Can any policy in  $P$  achieve  $V^*$ ? The answer is positive, and the function  $V^*$  tells us how: Any mapping  $p^* \in P$  such  $\forall x \in X, p^*(x) \in \arg \max_{a \in \Sigma} [xR(a) + \beta V^*(xM_a)]$  is well defined and  $H_{p^*} V^*(x) = H^* V^*(x) = V^*(x)$ , *i.e.* the unique fixed-point of  $H_{p^*}$  is  $V^*$ , hence the execution of policy  $p^*$  over the infinite horizon is expected to yield the value function  $V^*$ .  $V^*$  is called the optimal value function for the infinite horizon. We next elaborate on the properties of  $V^*$  and  $p^*$ .

**Lemma A.2.5**  *$H^*$  preserves continuity.*

**Proof.** Let  $V \in B(X)$  be continuous. Then for  $a \in \Sigma$ , the function  $xR_a + \beta V(xM_a)$  is also continuous: the composition of continuous functions and their summation preserves continuity. Therefore,  $H^* V$ , the pointwise maximum of these functions (one functions for each action), is also continuous.  $\square$

A piecewise linear function is a function that is linear over each partition of a finite partitioning of  $X$ .

**Lemma A.2.6** *If  $V$  is piecewise linear, then  $H^* V$  is piecewise linear. If in addition  $V$  is convex, then  $H^* V$  is also convex.*

**Proof.** Let  $V \in B(X)$  be piecewise linear and convex. Then for  $a \in \Sigma$ , the function  $xR_a + \beta V(xM_a)$  is also piecewise linear and convex: the composition of piecewise linear functions and their summation preserves piecewise linearity, and it is not hard to verify that if  $V$  is convex, because  $xM_a$  is a linear function,  $V(xM_a)$  is convex. Therefore,  $H^* V$ , the pointwise maximum of these functions (one functions for each action), is also piecewise linear and convex.  $\square$

Lemma A.2.6 shows that the finite horizon optimal value function is piecewise linear and convex: for a finite horizon of  $k$  stages, the optimal value function is simply  $H^{*k}$  applied to the zero function, and the zero function is piecewise linear, continuous and convex.

**Corollary A.2.7**  *$V^*$  is continuous and convex.*

**Proof.** Let  $V \in B(x)$  be continuous, piecewise linear and convex (*e.g.*, the zero function). Then the functions  $H^{*k}V, \forall k \geq 0$  are continuous, piecewise linear, and convex, and the convexity and continuity of  $V^*$  follows by the uniform convergence of  $H^{*k}V$  to  $V^*$ .  $\square$

The monotonicity and continuity preservation would hold even if  $H^*$  was not a contraction mapper. Using these facts, one can show that  $V^*$  is convex in the absence of discounting as well (for a goal model,  $V^*$  would denote maximum probability of reaching the goal, or we may consider the average-reward criterion). What follows are properties that bring out some differences between the presence and absence of discounting. Recall that (in the discounting case)  $V^*$  tells us how to construct an optimal policy: any mapping  $p^* \in P$  defined by  $\forall x \in X, p^*(x) \in \arg \max_{a \in \Sigma} [xR_a + \beta V^*(xM_a)]$  would do. We are not specific about the choice of actions when the set  $\arg \max_{a \in \Sigma} [xR_a + \beta V^*(xM_a)]$  has more than one action, since any choice would give us an optimal policy. This is due the fact that any such policy  $p^*$  corresponds to an operator  $H_{p^*}$  whose unique fixed-point is  $V^*$  by definition of  $p^*$  (and from the contraction property of  $H_{p^*}$ ).

Properties such as uniqueness of fixed-points and uniform convergence do not hold for operators that are not contraction mappings. A simple example of a goal model (with no discounting) illustrates cases where operators (corresponding to policies) can have more than one fixed-point, and a policy  $p$  for which the optimal value function is a fixed-point may not necessarily be an optimal (stationary) policy. Hence, when there is no discounting, some choices of actions in  $\arg \max_{a \in \Sigma} [xR_a + V^*(xM_a)]$  may not correspond to any optimal policy.

We say that action  $a$  is optimal at a point  $x$ , if there is some optimal stationary policy  $p^*$  that maps  $x$  to  $a$ . In the presence of discounting, we have:

$$a \in \arg \max_{\sigma \in \Sigma} [xR_\sigma + \beta V^*(xM_\sigma)] \Leftrightarrow a \text{ is optimal at } x \quad (\text{A.3})$$

In the undiscounted case, only the left implication holds.

## Appendix B

## THE WEAK EQUALITY TEST

Here we describe the algorithm, carried out by a PFA, for the weak equality test in detail. To see that such an algorithm can be executed by a PFA, it is easier to think of the PFA model as the equivalent one-way TM machine model, where the machine has a read-only input tape and a finite memory, and in addition can flip a fair coin on each step. Given is the input  $a^i b^j$ , and the question is whether  $i = j$ . The outputs of the algorithm (the state the PFA ends in) are *Indecision*, *Suspect* or *Correct* as described in Chapter 3. The PFA performs the following independent computations as it scans the input:

1. For each letter  $a$ , the algorithm flips two coins<sup>1</sup>.
2. For each letter  $b$ , the algorithm flips two coins.
3. For each letter  $a$  and for each letter  $b$ , the algorithm flips one coin.
4. For each letter  $a$  and for each letter  $b$ , the algorithm flips one coin.
5. The algorithm checks whether  $i \equiv j \pmod k$

$k$  is a sufficiently large constant. It can be easily verified that the algorithm can be carried out by a PFA for the operations described. If  $i \not\equiv j \pmod k$ , then the outcome is Suspect. Otherwise, let event  $A$  be true if computations 1 or 2 get only heads, and let event  $B$  be true if computations 3 or 4 get only heads. The algorithm outputs Indecision (the PFA goes into indecision state) if both  $A$  and  $B$  are false (the common case), or both are true. Otherwise, in case of a *Decision* outcome, the algorithm outputs Suspect if  $A$  is true and  $B$  is false, and it outputs Correct if  $A$  is false and  $B$  is true. We will next show that if  $i = j$ , then  $A$  and  $B$  have the same probability of being true, while if  $i \neq j$ , in case of a Decision outcome, and with  $k \geq 2$ , the probability of outputting Suspect is much higher than outputting Correct.

We have  $p_A = Pr(A) = 2^{-2i} + 2^{-2j} - 2^{-2i-2j}$ , and  $p_B = Pr(B) = 2^{-i-j} + 2^{-i-j} - 2^{-2i-2j}$ . Therefore, if  $i = j$ ,  $p_A = p_B$  and in case of a Decision outcome, Suspect and Correct outcomes are equally likely.

---

<sup>1</sup>All coins are assumed to be fair.

If  $i \neq j$ , assume without loss of generality that  $j = i + lk$  for some integer  $l \geq 1$ . First note that with  $k \geq 2$  or  $l \geq 1$ ,  $p_A$  is larger than  $p_B$  as  $p_A > 2^{-2i}$ , while  $p_B < 2^{-2i-2lk+1}$ . The probability of the Suspect outcome given Decision is  $Pr(A|\bar{A}B \text{ or } A\bar{B}) = Pr(A\bar{B}|\bar{A}\bar{B} \text{ or } \bar{A}B) = \frac{Pr(A\bar{B} \text{ and } (\bar{A}B \text{ or } A\bar{B}))}{Pr(\bar{A}B \text{ or } A\bar{B})} = \frac{Pr(A\bar{B})}{Pr(\bar{A}B \text{ or } A\bar{B})} = \frac{p_A(1-p_B)}{Pr(\bar{A}B \text{ or } A\bar{B})}$ . Similarly, the probability of Correct given Decision is  $Pr(B|A\bar{B} \text{ or } \bar{A}B) = \frac{p_B(1-p_A)}{p_A(1-p_B) + (1-p_A)p_B}$ . Consequently, given a Decision outcome, the probability of Suspect is at least  $2^k$  times higher than the probability of Correct:  $\frac{Pr(A|\bar{A}B \text{ or } A\bar{B})}{Pr(B|A\bar{B} \text{ or } \bar{A}B)} = \frac{p_A(1-p_B)}{p_B(1-p_A)} > \frac{p_A}{p_B} > \frac{2^{-2i}}{2^{-2i-2lk+1}} = 2^{2k-1}$ .

## Appendix C

**PROOF OF LEMMA 5.3.13 ON FAST CONVERGENCE OF THE  
NEWTON'S METHOD**

**Lemma C.0.8** For any Newton sequence  $\{x^{(i)}\}$  generated by  $(y, h, \{c^{(i)}\})$ ,  $\forall i \geq 0$ ,

$$1 - x^{(i+1)} = \frac{1-h}{1-m^{(i)}} \prod_{1 \leq j \leq i} \frac{c^{j-1}(\frac{h}{y}-1)(1-c^{(j)})}{1+c^{j-1}(\frac{h}{y}-1)},$$

where  $0 \leq m^{(i)} \leq 1-y$ .

**Proof.** Let  $r = r^{(0)} = \frac{h-y}{y}$ . We have  $\frac{F(x^{(i)})-f^*(x^{(i)})}{f^*(x^{(i)})-x^{(i)}} = r^{(i)} = c^i r$  (recall  $c^i = \prod_{1 \leq j \leq i} c^{(j)}$ ,  $i \geq 1$ , and  $c^0 = 1$ ). We have  $f^*(x^{(i)}) - x^{(i)} = (1-y)x^{(i)} + y - x^{(i)} = (1-x^{(i)})y$ , thus

$$\begin{aligned} F(x^{(i)}) - f^*(x^{(i)}) &= (f^*(x^{(i)}) - x^{(i)})c^i r \\ \Rightarrow F(x^{(i)}) &= f^*(x^{(i)}) + ((1-x^{(i)})y)c^i r \\ &= (1-y)x^{(i)} + y + (1-x^{(i)})c^i r y \\ &= x^{(i)} - yx^{(i)} + y + (1-x^{(i)})c^i r y \\ &= x^{(i)} + (1-x^{(i)})y + (1-x^{(i)})c^i r y \\ \Rightarrow F(x^{(i)}) &= x^{(i)} + (1-x^{(i)})(1+c^i r)y \end{aligned} \tag{C.1}$$

Taking definition of  $c^0 = 1$ , we can check that  $F(x^{(0)}) = F(0) = h$ . We have that  $x^{(i+1)}$ , where  $i \geq 0$ , is the fixed point of  $l^{(i)}$ . Therefore  $l^{(i)}$  goes through the point  $(x^{(i+1)}, x^{(i+1)})$ .  $l^{(i)}$  has slope  $m^{(i)}$  and also goes through the point  $(x^{(i)}, F(x^{(i)}))$ , thus:

$$m^{(i)} = \frac{x^{(i+1)} - F(x^{(i)})}{x^{(i+1)} - x^{(i)}},$$

or

$$\begin{aligned} \forall i \geq 0, x^{(i+1)} &= \frac{F(x^{(i)}) - m^{(i)}x^{(i)}}{1 - m^{(i)}} \\ &= \frac{[x^{(i)} + (1-x^{(i)})(1+c^i r)y] - m^{(i)}x^{(i)}}{1 - m^{(i)}} \quad (\text{using C.1}), \\ &= \frac{x^{(i)} - m^{(i)}x^{(i)}}{1 - m^{(i)}} + \frac{(1-x^{(i)})(1+c^i r)y}{1 - m^{(i)}} \end{aligned}$$

$$\Rightarrow x^{(i+1)} = x^{(i)} + \frac{(1-x^{(i)})(1+c^i r)y}{1-m^{(i)}} \quad (\text{C.2})$$

and therefore,

$$\begin{aligned} 1-x^{(i+1)} &= 1-x^{(i)} - \frac{(1-x^{(i)})(1+c^i r)y}{1-m^{(i)}} \\ &= (1-x^{(i)})\left(1 - \frac{(1+c^i r)y}{1-m^{(i)}}\right) \Rightarrow \\ 1-x^{(i+1)} &= \prod_{0 \leq j \leq i} \left(1 - \frac{(1+c^j r)y}{1-m^{(j)}}\right) \end{aligned} \quad (\text{C.3})$$

We will now derive an expression for  $m^{(i)}$ , which results in further simplification of the expression for  $1-x^{(i+1)}$ . We have  $l^{(i)}$  going through  $(x^{(i)}, F(x^{(i)}))$  and  $(x^{(i-1)}, F(x^{(i-1)}))$ , thus  $m^{(i)} = \frac{F(x^{(i)})-F(x^{(i-1)})}{x^{(i)}-x^{(i-1)}}$ , and,

$$\begin{aligned} m^{(i)} &= \frac{x^{(i)} + (1-x^{(i)})(1+c^i r)y - [x^{(i-1)} + (1-x^{(i-1)})(1+c^{i-1} r)y]}{x^{(i)} - x^{(i-1)}} \\ &= \frac{x^{(i)} - x^{(i-1)}}{x^{(i)} - x^{(i-1)}} + \frac{(1-x^{(i)})c^{(i)} - (1-x^{(i-1)})}{x^{(i)} - x^{(i-1)}} c^{i-1} r y + \frac{(1-x^{(i)}) - (1-x^{(i-1)})}{x^{(i)} - x^{(i-1)}} y \\ &= 1 - y - \frac{c^{i-1} r y [(1-c^{(i)}) - (x^{(i-1)} - c^{(i)} x^{(i)})]}{x^{(i)} - x^{(i-1)}} \end{aligned}$$

We rewrite  $x^{(i)} - x^{(i-1)}$  and  $x^{(i-1)} - c^{(i)} x^{(i)}$ , by replacing for  $x^{(i)}$  using C.2. We have

$$x^{(i)} - x^{(i-1)} = \left[ x^{(i-1)} + \frac{(1-x^{(i-1)})(1+c^{i-1} r)y}{1-m^{(i-1)}} \right] - x^{(i-1)} = \frac{(1-x^{(i-1)})(1+c^{i-1} r)y}{1-m^{(i-1)}},$$

and

$$x^{(i-1)} - c^{(i)} x^{(i)} = (1-c^{(i)})x^{(i-1)} - \frac{c^{(i)}(1-x^{(i-1)})(1+c^{i-1} r)y}{1-m^{(i-1)}}.$$

Thus,

$$\begin{aligned} m^{(i)} &= 1 - y - \frac{c^{i-1} r y [(1-c^{(i)}) - ((1-c^{(i)})x^{(i-1)} - \frac{c^{(i)}(1-x^{(i-1)})(1+c^{i-1} r)y}{1-m^{(i-1)}})](1-m^{(i-1)})}{(1-x^{(i-1)})(1+c^{i-1} r)y} \\ &= 1 - y - \frac{c^{i-1} r [(1-c^{(i)})(1-x^{(i-1)}) + \frac{c^{(i)}(1-x^{(i-1)})(1+c^{i-1} r)y}{1-m^{(i-1)}}] (1-m^{(i-1)})}{(1-x^{(i-1)})(1+c^{i-1} r)} \\ &= 1 - y - \frac{c^{i-1} r [(1-c^{(i)}) + \frac{c^{(i)}(1+c^{i-1} r)y}{1-m^{(i-1)}}] (1-m^{(i-1)})}{1+c^{i-1} r} \\ &= 1 - y - \frac{c^{i-1} r [(1-m^{(i-1)})(1-c^{(i)}) + c^{(i)}(1+c^{i-1} r)y]}{1+c^{i-1} r} \\ &= 1 - y - c^i r y - \frac{c^{i-1} r (1-c^{(i)})}{1+c^{i-1} r} (1-m^{(i-1)}), \end{aligned}$$

thus, for  $i \geq 1$ ,

$$1-m^{(i)} = (1+c^i r)y + \frac{c^{i-1} r (1-c^{(i)})}{1+c^{i-1} r} (1-m^{(i-1)}) \quad (\text{C.4})$$

It follows from C.4 that  $1 - m^{(i)} \geq y$  or  $m^{(i)} \leq 1 - y, \forall i$ . In fact inspecting C.4 shows that as expected, with increasing  $i$ , the slopes  $m^{(i)}$  converge to  $1 - y$  which is the slope of  $f^*$ .

Replacing C.4 in C.3, we get,

$$\begin{aligned}
1 - x^{(i+1)} &= \prod_{0 \leq j \leq i} \frac{1 - m^{(j)} - ((1 + c^j r)y)}{1 - m^{(j)}} \\
&= \frac{1 - m^{(0)} - ((1 + c^0 r)y)}{1 - m^{(0)}} \\
&\quad \prod_{1 \leq j \leq i} \frac{(1 + c^i r)y + \frac{c^{j-1} r(1 - c^{(j)})}{1 + c^{j-1} r} (1 - m^{(j-1)}) - ((1 + c^j r)y)}{1 - m^{(j)}} \\
&= (1 - h) \prod_{1 \leq j \leq i} \frac{c^{j-1} r(1 - c^{(j)})}{1 + c^{j-1} r} \prod_{1 \leq j \leq i} \frac{1 - m^{(j-1)}}{1 - m^{(j)}} \\
\Rightarrow 1 - x^{(i+1)} &= \frac{1 - h}{1 - m^{(i)}} \prod_{1 \leq j \leq i} \frac{c^{j-1} r(1 - c^{(j)})}{1 + c^{j-1} r},
\end{aligned}$$

and recall that  $r = r^{(0)} = \frac{h}{y} - 1$ . □

### Biographical Note

Omid Madani was born in Tehran, Iran, and raised in Bandar Abbas. Later, he and his family moved to Dubai, UAE, where he spent his secondary school years. He then went to the United States and spent two years at Saddleback Community College, in Mission Viejo, California, after which he transferred to the University of Houston, in Houston, Texas. In 1993, he obtained a B.S. in computer science from the University of Houston. He spent the next seven years at the University of Washington in Seattle, where he obtained M.S. and Ph.D. degrees, both in computer science, respectively in the years 1995 and 2000.