THE LINEAR PROGRAMMING APPROACH TO APPROXIMATE DYNAMIC PROGRAMMING: THEORY AND APPLICATION

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

Daniela Pucci de Farias
June 2002

Abstract

Dynamic programming offers a unified approach to solving problems of stochastic control. Central to the methodology is the *optimal value function*, which can be obtained via solving Bellman's equation. The domain of the optimal value function is the state space of the system to be controlled, and dynamic programming algorithms compute and store a table consisting of the optimal value function evaluated at each state. Unfortunately, the size of a state space typically grows exponentially in the number of state variables. Known as the *curse of dimensionality*, this phenomenon renders dynamic programming intractable in the face of problems of practical scale.

Approximate dynamic programming aims to alleviate the curse of dimensionality by considering approximations to the optimal value function — *scoring functions* — that can be computed and stored efficiently. For instance, one might consider generating an approximation within a parameterized class of functions, in a spirit similar to that of statistical regression. The focus of this dissertation is on an algorithm for computing parameters for linearly parameterized function classes. We study *approximate linear programming*, an algorithm based on a linear programming formulation that generalizes the linear programming approach to exact dynamic programming.

Over the years, interest in approximate dynamic programming has been fueled by stories of empirical success in application areas spannning from games to dynamic resource allocation to finance . Nevertheless, practical use of the methodology remains limited by a lack of systematic guidelines for implementation and theoretical guarantees, which reflects on the amount of trial and error involved in each of the success stories found in the literature and on the difficulty of duplicating the same success in other applications. The research presented in this dissertation addresses some of these issues. In particular, our analysis leads to theoretically motivated guidelines for implementation of approximate linear programming.

Specific contributions of this dissertation can be described as follows:

- We offer bounds that characterize the quality of approximations produced by the linear programming approach and the quality of the policy ultimately generated. In addition to providing performance guarantees, the error bounds and associated analysis offer new interpretations and insights pertaining to the linear programming approach. Among other things, this understanding to some extent guides selection of basis functions and "state-relevance weights" that influence quality of the approximation.

- Approximate linear programming involves solution of linear programs with relatively few variables but an intractable number of constraints. We propose a constraint sampling scheme that retains and uses only a tractable subset of the constraints. We show that, under certain assumptions, the resulting approximation is comparable to the solution that would be generated if all constraints were taken into consideration.

- Some of the theoretical results are illustrated in applications to queueing problems. We also devote a chapter to case studies that demonstrate practical aspects of implementation of approximate linear programming to queueing problems and web server farm management and demonstrate effectiveness of the methodology in problems of relevance to industry.

Preface

Few problems are more universal than that of sequential decision-making under uncertainty. To shape one's actions to balance present and future consequences is life's ever-standing challenge, and the ability to successfully navigate through the overwhelmingly large tree of possibilities what might be regarded as the art of the wise.

What, then, constitutes wisdom? How does one not become paralyzed in the analysis of all possible scenarios, but make good moves with limited consideration instead? What does good even mean in this setting?

In life, the ability to make good decisions lies to a large extent in the ability to see the forest through the trees. When the mapping from actions to consequences is nontrivial and exhaustive search through all possible scenarios is prohibitively expensive, clarity of thought allows one to filter out unnecessary details and identify the main factors at play, reducing the problem complexity.

Although not quite as fascinating as life's most pressing issues, many problems in engineering, science and business pose similar challenges: a large number of scenarios and options, uncertainty about the future, and nontrivial mapping from actions to consequences. This dissertation is part of an effort toward the understanding of decision-making in such problems; toward the development of a unified theory of control of complex systems. In other words, toward the development of the mathematical and algorithmic equivalent of wisdom.

Acknowledgments

My thesis advisor Prof. Ben Van Roy had immense impact on my research philosophy and style. I am grateful to him for passing on to me strong ethical values, appreciation for elegant work and the willingness to set ambitious research goals. The research presented in this dissertation is the fruit of a close collaboration and would certainly not have been the same without his invaluable contribution.

I am also thankful to my dissertation readers Professors Peter Glynn and Arthur Veinott Jr. and defense committee members Professors Daphne Koller and David Luenberger for valuable comments and suggestions.

Section 6.2 represents joint work with Drs. Alan King and Mark Squillante, as well as Dave Mulford. The fluid model approximation for the problems of scheduling and routing is due to Drs. Zhen Liu, Mark Squillante and Joel Wolf. The model for arrival rate processes was based on discussions with Dr. Ta-Hsin Lee.

My education at Stanford was enriched by interaction with fellow students. I would like to thank David Choi, Eric Cope, Kahn Mason, Dave Mulford, Shikhar Ranjan, Paat Rusmevichientong and Peter Salzman for great discussions on various research subjects.

I thank my family for unconditional support throughout the years. They provided everything I needed to succeed. Eduardo Cheng was also a fundamental source of support in my early years at Stanford. Discussions with my master's thesis advisor Prof. José Cláudio Geromel in my early undergraduate and master's years played a fundamental role in shaping my career path. He has also been a permanent source of support and encouragement.

Through the days and through the nights, little Francisca was there for me. She

never uttered a word of discouragement, and was always cheerful and willing to endure long hours of work with me. Life as a researcher would not be the same without the lovely sight of her lying in the corner of the room, walking on the keyboard, or happily eating my papers.

This research was supported by a Stanford School of Engineering Fellowship, a Wells Fargo Bank Fellowship, a Stoloroff Fellowship, an IBM Research Fellowship, the NSF CAREER Grant ECS-9985229, and the ONR under Grant MURI N00014-00-1-0637.

To the loving memory of Sissi,
and to adorable Francisca.

Contents

Chapter 1

Introduction

Problems of sequential decision-making in complex systems are a recurrent theme in engineering, science and business. Optimal control of such systems typically requires consideration of a large number of scenarios and options and treatment of uncertainty surrounding future events. Some problems exhibit special structure that allows one to circumvent the apparent complexity and achieve optimality efficiently; however, for the vast majority of applications, optimality remains an impossible goal in face of current computing power limitations. How to make good decisions efficiently in such applications is the subject of this dissertation.

1.1 Complex Systems

We defer a formal characterization of complex systems to Chapter 2. Loosely described, the complexity of a system is determined by certain features that may influence how difficult it is to design optimal control strategies. Features that have an impact include the complexity of the system dynamics (linear/nonlinear); the presence or absence of uncertainty; the number of scenarios and options being considered. We now present instances of control problems involving complex systems, illustrating the impact of some of these features on the decision-making process.

Example 1.1 Scheduling in manufacturing systems
Consider a plant used for the manufacturing of several different types of products.

Each product is characterized by a sequence of processing steps. Machines in this plant may perform multiple functions, such as several processing steps for a given product, or processing steps for several different products. Managing such systems requires scheduling each machine to share its time among the several jobs waiting in queue for service. This is done with some objective in mind; common choices are to maximize production rates (throughput) or minimize the number of unfinished units in the system. Since jobs travel from machine to machine, there is usually a high degree of interconnectivity in manufacturing systems, and optimality requires joint consideration of the status of all queues in the system and of the decisions being made for all machines, leading to difficulties as the system dimensions increase.

Example 1.2 Zero-sum games
Zero-sum games involve two agents playing with completely opposed objectives: the gain of one agent is the loss of the other. An example of a zero-sum game is chess. An optimal strategy for playing chess requires investigation of all possible moves for the current step, investigation of all subsequent moves for the adversary, and so on, until completion of the game, so that one can choose the move that maximizes the chance of victory. Given the large number of moves available for each player and the large number of configurations the chess board can take, this approach is infeasible: consideration of even four or five moves into the future leads to a huge number of possibilities.

Example 1.3 Dynamic portfolio allocation
Consider the problem of deciding what fraction of current capital to invest in each of multiple stocks available in the market. A fraction of the money is devoted to consumption, and the objective is to maximize the sum of a utility of consumption over time. Stock prices follow particularly complicated dynamics, being affected by uncertainty and by a large number of factors in the market. Optimality in the long run would require consideration of the joint evolution of all these factors, as they are generally interdependent. This is intractable unless only a very small number of factors is considered in the stock price model.

1.2 Approximate Dynamic Programming

A common approach to decision-making in complex systems is the heuristic approach. Much like what one would do in life, when confronted with such systems one tries to identify ways of simplifying the problem to make it solvable while still capturing its essence. Heuristics exploit the best of human capacity — creativity — and work surprisingly well at times. However, they also have to be reinvented with each new application, as in general little knowledge is reused when one switches between applications which are far apart in scope, for example, from scheduling in manufacturing systems to chess. A strong focus on deriving simple rules of thumb, which is common in problem-specific approaches, may also lead to inefficient use of a valuable resource: computers' power to perform a large number of operations fast and accurately.

Dynamic programming offers a unified treatment of a wide variety of problems involving sequential decision-making in systems with nonlinear, stochastic dynamics. Systems in this setting are described by a collection of variables evolving over time — the *state variables*. State variables take values in the *state space* of the system, the set of all possible *states* the system can be in. The central idea in dynamic programming is that, for a variety of optimality criteria, optimal decisions can be derived from a score assigned to each of the different states in the system. The optimal scoring function prescribed by dynamic programming is referred to as the *optimal value function*, and it captures the advantage of being in a given state relative to being in others.

Unfortunately, the applicability of dynamic programming is severely limited by the *curse of dimensionality*: computing and storing the optimal value function over the entire state space requires time and space exponential in the number of state variables. Hence for most problems of practical interest, dynamic programming remains computationally infeasible.

Approximate dynamic programming tries to combine the best elements of the heuristic and the dynamic programming approaches. It draws on traditional dynamic programming concepts and techniques to derive appropriate scoring functions. However, differently from dynamic programming, the emphasis is not on global optimality, but rather on doing well given computational limitations. In approximate dynamic programming, this translates into finding scoring functions that can be computed and stored efficiently. The underlying assumption is that many problems of practical interest exhibit some structure leading to the existence of reasonable scoring functions that can be represented compactly. Algorithms drawing on dynamic programming concepts and simultaneously exploiting problem-specific structure would hopefully combine the generality of dynamic programming and the efficiency of heuristics.

We refer to the general structure one assumes for the scoring function as the *approximation architecture*. Approximation architectures are one of the two central themes of research in approximate dynamic programming. The other central theme relates to algorithms for finding a good scoring function within the architecture under consideration.

1.3 Approximation Architectures

An appropriate choice of approximation architecture is essential for successful use of approximate dynamic programming: a scoring function can only be as good as the approximation architecture that defines its structure. In this dissertation, we consider the use of *linear architectures*. In a spirit reminiscent of linear regression, one chooses a collection of functions mapping the system state space to real numbers — the *basis functions* — and generates a scoring function by finding an appropriate linear combination of these basis functions. Note that the architecture is called linear because we look for scoring functions that are linear in the basis functions; in principle, the basis functions are arbitrary functions of the system state. Figure 1.1 illustrates the use of basis functions in generating a scoring function.

A scoring function representable as a linear combination of basis functions is fully characterized by the weights assigned to each of the basis functions in this linear combination. Hence it suffices to store these weights (one per basis function) as an encoding of the scoring function. In contrast, the optimal value function typically requires representation as a lookup table, with one value per state in the system. The use of a linear approximation architecture will therefore be feasible and advantageous if either we have only a manageable number of basis functions or we have a large

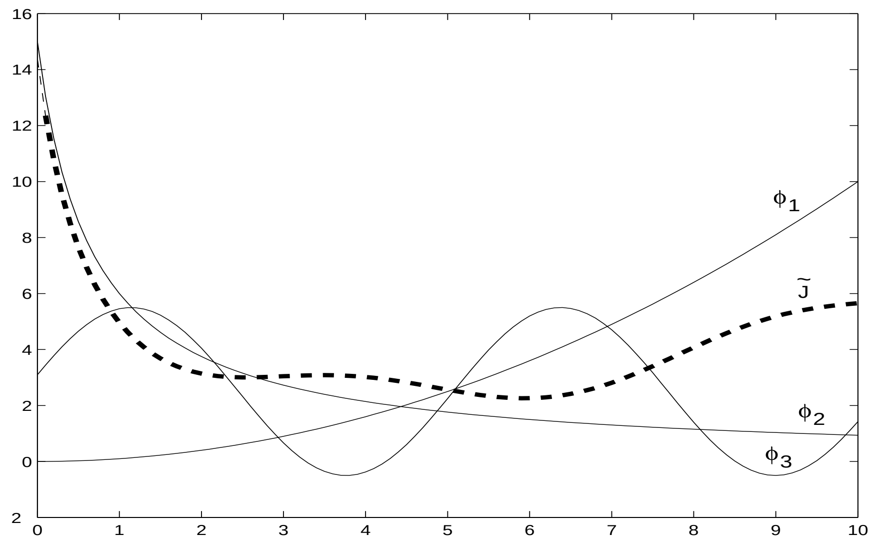


Figure 1.1: A Linear Architecture: Scoring function $\tilde{J}$ is a linear combination of basis functions ϕ_1, ϕ_2 and ϕ_3.

number of basis functions, but restrict choices to scoring functions that are sparse linear combinations of the basis functions. In this dissertation, we take the first approach; the underlying assumption is that many problems of interest will present structure leading to the existence of an appropriate, reasonably sized collection of basis functions.

Note that, in principle, the existence of a relatively small number of basis functions enabling the representation of good basis functions is not such a strong requirement. In fact, for any given problem, if we only have *one* basis function, corresponding to the optimal value function, adding extra basis functions should not improve the approximation architecture. However, another underlying assumption in the use of linear architectures is that one should not have to store the basis functions as lookup tables, but rather they have to be such that they can be computed efficiently on a need basis. Having basis functions that need to be stored as lookup tables would lead to the same problem encountered in trying to store the optimal value function, namely, that storage space requirements are unacceptable: linear in the number of states in the system, or exponential in the number of state variables.

The practical limitations on the complexity of the basis functions also imply that, although in principle linear architectures are rich enough, in practice it may be worth exploring different approximation architectures as well. Common choices are neural networks [7, 25], or splines [50, 11]. A potential drawback in the use of nonlinear architectures is that algorithms for finding appropriate scoring functions with such a structure are usually more complicated.

1.4 Approximate Linear Programming

Given a prespecified linear architecture, we must be able to identify an appropriate scoring function among all functions that are representable as linear combinations of the basis functions. In approximate dynamic programming, it is natural to define the quality of the scoring function in terms of its closeness to the optimal value function. Therefore an ideal algorithm might try to choose the linear combination of basis functions that minimizes some distance to the optimal value function. Note that this is the main idea in traditional regression problems, where one tries to fit a curve based on noisy observations by minimizing, for example, the sum of the squared errors over the samples. Unfortunately, the same idea cannot be applied in the approximate dynamic programming setting because we cannot sample the optimal value function.

Alternatively, we seek inspiration in standard dynamic programming algorithms to derive scoring functions that will hopefully serve as good substitutes to the optimal value function. Approximate dynamic programming algorithms are typically adaptations of standard dynamic programming algorithms that are changed to account for the use of an approximation architecture. Approximate linear programming, the method studied here, falls in that category.

Approximate linear programming is inspired by the more traditional linear programming approach to exact dynamic programming [9, 17, 18, 19, 26, 34]. Perhaps quite surprisingly, even though dynamic programming problems involve optimization of fairly arbitrary functionals, they can be recast as linear programs (LP's). However, these LP's are not immune the curse of dimensionality: they have as many variables as the number of states in the system, and at least the same number of constraints.

Combining the linear programming approach to exact dynamic programming with linear approximation architectures leads to the approximate linear programming algorithm (ALP), originally proposed by Schweitzer and Seidmann [43]. ALP involves the use of an LP with reduced dimensions to generate a scoring function. We call this LP the *approximate LP*. Compared with the exact LP that computes the optimal value function, the approximate LP will typically have a much smaller number of variables: it has as many variables as the number of basis functions. A potential source of concern is that the approximate LP still has a huge number of constraints, in fact as many as those present in the exact LP. However, LP's presenting few variables and a large number of constraints are good candidates to constraint generation techniques. Specifically, we show in the approximate linear programming case how an efficient constraint sampling mechanism can be designed for achieving good approximations to the approximate LP solution in reasonable time.

Compared to other approximate dynamic programming algorithms, approximate linear programming offers some advantages:

- Approximate linear programming capitalizes on decades of research on linear programming. Heavy use of such a standard methodology implies that implementation of the approximate LP may be easier to non-experts than implementation of other methods would be. Combined with the fact that one can take advantage of the several large-scale linear programming algorithms available, this leads to a higher likelihood of approximate linear programming effectively becoming a useful method in industry.

- In the case of minimizing costs (maximizing rewards), approximate linear programming offers a lower (upper) bound on the optimal long-run costs (rewards). Upper (lower) bounds can be generated fairly easily by simulation of any policy. Therefore, there is a possibility of identifying from the approximate linear programming solution whether the chosen linear architecture is satisfactory or further refinements may lead to substantial improvement in performance.

- The inherent simplicity of linear programming implies that approximate linear programming is relatively easier to analyze than some other approximate dynamic programming algorithms. A great obstacle in making approximate dynamic programming popular is that, apart from anecdotal success, it has been difficult to demonstrate strong guarantees for such methods, and to fully understand their behavior. A characteristic common to all approximate dynamic programming methods is that their behavior depends on many different algorithm parameters. With little understanding of how each of the parameters affects the overall quality of the method, appropriately setting them typically requires a large amount of trial and error, and there is a high probability of failure in identifying adequate values. While this problem is not fully solved in approximate linear programming, there is a deeper understanding of the impact of its parameters in the ultimate scoring function being generated. Analysis such as that presented in this dissertation provides valuable guidance in the choice of algorithm parameters and increases the probability of successful implementation of the method.

1.5 Some History

Perhaps one of the earliest developments in the area of automated decision-making in complex systems is due to Shannon, in a paper that proposes an algorithm for playing chess [44]. The paper suggests the use of scoring functions for assessing the many different board configurations a move can lead to. The scoring function should represent an assessment of long-run rewards associated with each board configuration. Scores were based on a linear combination of certain features of the board configuration empirically known to be relevant to the game. Both the features and the linear combination thereof used as a scoring function were chosen heuristically. The idea of using scoring functions for assessing board configuration forms the basis for modern chess playing projects.

Concern with the curse of dimensionality and research on approximate dynamic programming was a part of early research on dynamic programming. Belmann referred to such algorithms, aiming to the design of scoring functions to serve as approximations to the optimal value function, as approximations in the value space [3].

However, despite early efforts, relatively modest progress was made until the recent renaissance of the field in the artificial intelligence domain. Successful implementation of reinforcement learning algorithms, most notably the development of a world-class automatic backgammon player [48], generated great interest in the methodology and inspired further research in the area. Later analysis of reinforcement learning revealed strong connections with dynamic programming: the algorithm corresponds to a stochastic, approximate version of value and policy iteration, traditional dynamic programming algorithms. Unfortunately, reinforcement learning algorithms such as temporal-difference learning remain largely unexplained, with the strongest guarantees being limited to the cases of autonomous systems and optimal stopping problems [51, 53, 52].

An alternative to approximate dynamic programming algorithms such as temporal-difference learning or approximate linear programming is what are called approximations in the policy space. In such algorithms, instead of trying to derive a good policy indirectly via generation of a scoring function, one aims to generating a sequence of improving policies. Searching directly in the policy space typically requires consideration of parametric policies. If the dependence of the optimality criterion (e.g., average cost) on the policy parameter is differentiable, one may be able to use gradient descent methods to identify a policy corresponding to a local optimum within the parametric class under consideration [1, 2, 35, 37, 58].

A more recent development combines approximation in the state space with approximation in the policy space. Actor-critic algorithms involve the use of both parametric policies and value function approximation to generate a stochastic version of policy iteration [46, 28, 27].

1.6 Chapter Organization and Contributions

In Chapter 2, we formally define complex systems, in the framework of Markov decision processes. We describe how dynamic programming offers a solution to the problem of minimizing infinite-horizon, discounted costs, and discuss how the curse of dimensionality affects these algorithms. We then present some of the main ideas in approximate dynamic programming and introduce approximate linear programming, the algorithm studied in the remaining chapters.

In Chapter 3, we address the issue of how to balance accuracy of approximation of the optimal value function over different portions of the state space. Original contributions include:

- showing how approximate linear programming provides the opportunity for assigning different weights — *state-relevance weights* — to approximation errors over different portions of the state space, therefore allowing for user-designed emphasis of regions of greater importance;

- development of a bound on the performance loss resulting from use of a scoring function generated by approximate linear programming instead of the optimal value function;

- two examples motivating heuristics for identifying what regions of the state space should be assigned higher state-relevance weights.

In Chapter 4, we develop bounds on the error yielded by approximate linear programming in trying to approximate the optimal value function. For the approximate LP to be useful, it should deliver good approximations when the linear architecture being used is good. We develop bounds that ensure desirable results of this kind. Original contributions in this chapter include:

- bounds on the error yielded by approximate linear programming in the optimal value function approximation, compared against the "best possible" error achievable by the prespecified linear architecture. This is the first such error bound for any algorithm that approximates optimal value functions of general stochastic control problems by computing weights for arbitrary collections of basis functions;

- analysis of queueing problems demonstrating scaling properties of the approximate linear programming error bounds. In particular, we demonstrate for classes of problems in queueing that the error is uniformly bounded on the problem dimensions;

- identification of the dynamic programming concepts enabling development of the approximate linear programming error bounds, with analysis of connections to standard dynamic programming results.

In Chapter 5, we develop a constraint sampling algorithm for approximate linear programming. ALP typically involves LP's with a relatively small number of variables, but an intractable number of constraints. We study a constraint sampling scheme that retains and uses only a tractable subset of the constraints. Original contributions of this chapter include:

- development of bounds on the constraint sampling complexity for ensuring *near-feasibility* for generic collections of linear constraints with relatively few variables but a large number of constraints;

- development of bounds on the constraint sampling complexity for approximate linear programming for ensuring a value function approximation error comparable to what would be obtained with full consideration of all constraints.

The constraint sampling algorithm presented in Chapter 5 is the first addressing the problem of implementing approximate linear programming for general MDP's.

In Chapter 6, we develop applications of approximate linear programming to problems in queueing control and web server farm management. The applications illustrate some practical aspects in the implementation of approximate linear programming and demonstrate its competitiveness relative to some simple heuristics for tackling the problems being considered.

In Chapter 7 we provide closing remarks and discuss directions for future research on decision-making in complex systems.

Results in Chapters 3 and 4, as well as the case studies on queueing networks presented in Chapter 6 appeared previously in [15]. Results in Chapter 5 previously appeared in [16].

Chapter 2

Approximate Linear Programming

Markov decision processes (MDP's) provide a unified framework for the treatment of problems of sequential decision-making under uncertainty. For a variety of optimality criteria, these problems can be solved by dynamic programming. The main strength of this approach is that fairly general stochastic and nonlinear dynamics can be considered. However, the same generality leads to poor use of problem-specific information to guide the derivation of optimal policies, leading to efficiencies that are ultimately unbearable — dynamic programming algorithms are computationally infeasible for all but very small problems, or problems exhibiting very special structure.

In this chapter, we formally define complex systems, in the MDP framework. We describe how dynamic programming offers a solution to the problem of minimizing discounted costs over an infinite horizon, and discuss how the curse of dimensionality affects dynamic programming algorithms. We then present some of the main ideas in approximate dynamic programming and introduce approximate linear programming, the algorithm studied in the remaining chapters.

2.1 Markov Decision Processes

In the MDP framework, systems are characterized by collections of variables evolving over time — the *state variables*. State variables summarize the history of the system, containing all information that is relevant to predicting future events. In formulating a problem as an MDP, state variables must be designed to satisfy the Markov property: conditioned on the current state of the system being known, its future is independent from its past. The Markov property has important implications in the search for optimal policies, as we shall later see.

The evolution of state variables is partially dependent on decisions or controls exogenous to the system. We consider the problem of determining such decisions so as to minimize a discounted sum of costs incurred as the system runs ad infinitum. We consider systems running in discrete time. Costs accrue at each time step and depend on the state of the system and action being taken at that time.

We now provide a formal description of Markov decision processes. A Markov decision process in discrete-time is characterized by a tuple

$$(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot,\cdot), g_\cdot(\cdot), \alpha),$$

with the following interpretation. We consider stochastic control problems involving a finite state space $\mathcal{S}$ of cardinality $|\mathcal{S}| = N$. For each state $x \in \mathcal{S}$, there is a finite set of available actions $\mathcal{A}_x$. When the current state is x and action $a \in \mathcal{A}_x$ is taken, a cost $g_a(x)$ is incurred. State transition probabilities $P_a(x,y)$ represent, for each pair (x,y) of states and each action $a \in \mathcal{A}_x$, the probability that the next state will be y conditioned on the current state being x and the current action being $a \in \mathcal{A}_x$. The *discount factor* α is a scalar between zero and one and represents inter-temporal preferences, indicating how costs incurred at different time steps are combined in a single optimality criterion.

A *policy* is a mapping from states to actions. Given a policy u, the dynamics of the system follow a Markov chain with transition probabilities $P_{u(x)}(x,y)$. For each policy u, we define a transition matrix P_u whose (x,y)th entry is $P_{u(x)}(x,y)$.

For concreteness, let us consider an example.

Example 2.1 A queueing problem

Consider the queueing network in Figure 2.1. We have three servers and two different kinds of jobs, traveling on distinct, fixed routes $\{1,2,3\}$ and $\{4,5,6,7,8\}$, forming a total of 8 queues of jobs at distinct processing stages. We assume that service times

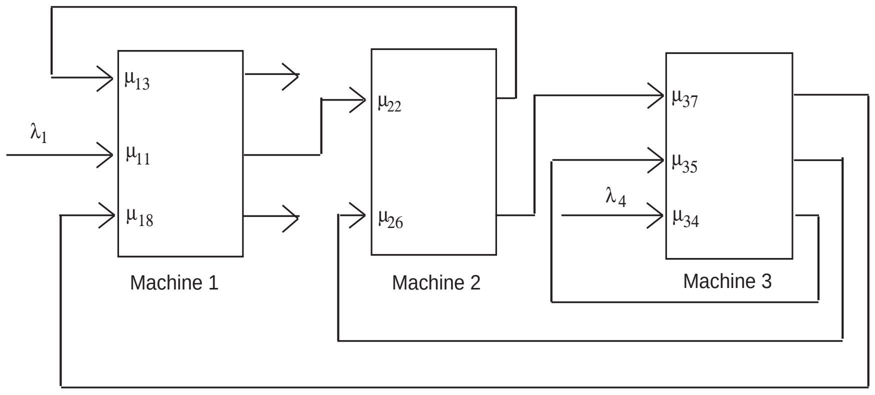


Figure 2.1: A queueing problem

are distributed according to geometric random variables: When a server i devotes a time step to serving a unit from queue j, there is a probability μ_{ij} that it will finish processing the unit in that time step, independent of past work done on the unit. Upon completion of that processing step, the unit is moved to the next queue in its route, or out of the system if all processing steps have been completed. New units arrive at the system in queues $j = 1, 4$ with probability λ_j in any time step, independent of previous arrivals.

A common choice for the state of this system is an 8-dimensional vector x containing the queue lengths. Since each server serves multiple queues, in each time step it is necessary to decide which queue each of the different servers is going to serve. A decision of this type may be encoded as an 8-dimensional vector a indicating which queues are being served, satisfying the constraint that no more than one queue associated with each server is being served; i.e., $a_i \in \{0, 1\}$, and $a_1 + a_3 + a_8 \leq 1,\ a_2 + a_6 \leq 1,\ a_4 + a_5 + a_7 \leq 1$. We can impose additional constraints on the choice of a as desired, for instance considering only non-idling policies.

Policies are described by a mapping u returning an allocation of server effort a as a function of system state x. We represent the evolution of the queue lengths in terms of transition probabilities — the conditional probabilities for the next state $x(t+1)$

given that the current state is $x(t)$ and the current action is $a(t)$. For instance:

$$\begin{aligned}
&Prob\,(x_1(t+1) = x_1(t) + 1|x(t), a(t)) = \lambda_1,\\
&Prob\,(x_3(t+1) = x_3(t) + 1, x_2(t+1) = x_2(t) - 1|x(t), a(t))\\
&\qquad = \mu_{22} I(x_2(t) > 0, a_2(t) = 1),\\
&Prob\,(x_3(t+1) = x_3(t) - 1, |x(t), a(t)) = \mu_{13} I(x_3(t) > 0, a_3(t) = 1),
\end{aligned}$$

corresponding to an arrival to queue 1, a departure from queue 2 and an arrival to queue 3, and a departure from queue 3. $I(.)$ is the indicator function. Transition probabilities related to other events are defined similarly.

We consider costs of the form $g(x) = \sum_i x_i$, the total number of unfinished units in the system. For instance, this is a reasonably common choice of cost for manufacturing systems, which are often modeled as queueing networks.

The problem of stochastic control amounts to selection of a policy that optimizes a given criterion. We employ as an optimality criterion infinite-horizon discounted cost of the form

$$J_u(x) = \mathrm{E}\left[\sum_{t=0}^{\infty} \alpha^t g_u(x_t) \Big| x_0 = x\right], \tag{2.1}$$

where $g_u(x)$ is used as shorthand for $g_{u(x)}(x)$ and the discount factor $\alpha \in (0,1)$ reflects inter-temporal preferences. In finance problems, α has a concrete interpretation: the same nominal value is worth less in the future than in the present, since in the latter case it can be invested for a risk-free return. Discount factors are also useful in problems where the system parameters are slowly changing over time, so that costs predicted for the near future are more certain. Note that using a discount factor close to one yields an approximation to the problem of minimizing average costs; in fact, for any MDP with finite state and action spaces, there is a large enough discount factor strictly less than one such that any discount factor larger than that is optimal for the average-cost criterion [4, 8, 55].

It is well known that there exists a single policy u that minimizes $J_u(x)$ simultaneously for all x, and the goal is to identify that policy. The Markov property, establishing that, conditioned on the current state, the future of an MDP is independent of its past, implies that it suffices to consider policies of the type being used here — mapping from states to actions; extending consideration to policies depending on the history of the system does not improve performance.

In the next section, we describe how an optimal policy may be found via dynamic programming, how the curse of dimensionality makes application of DP algorithms intractable, and how approximate dynamic programming addresses the issue.

2.2 Approximate Dynamic Programming

For any given scoring function, an associated policy can be generated as follows. Let $J : \mathcal{S} \mapsto \Re$ be a scoring function. Then we generate a policy u_J by taking

$$u_J(x) = \operatorname*{argmin}_a \left\{ g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y) J(y) \right\}.$$

Policy u_J is called *greedy* with respect to J. Consider the scoring function J^*, given by

$$J^* = \min_u J_u.$$

This function is denominated the *optimal value function*, and maps each state x to the minimal expected discounted cost attainable by any policy, conditioned on the initial state being x. A standard result in dynamic programming is that a policy is optimal if and only if it is greedy with respect to the optimal value function. Hence the problem of finding an optimal policy can be converted into the problem of computing the optimal value function.

Let us define dynamic programming operators T_u and T by

$$T_u J = g_u + \alpha P_u J \quad \text{and} \quad TJ = \min_u \left(g_u + \alpha P_u J \right),$$

where the minimization is carried out component-wise. A standard dynamic programming result establishes that J^* is the unique solution to Bellman's equation

$$J = TJ.$$

For any function v, let the maximum norm $\|\cdot\|_\infty$ be given by

$$\|v\|_\infty = \max_i |v(i)|.$$

The dynamic programming operator T satisfies the following properties.

Theorem 2.1. *[4] Let J and $\bar{J}$ be arbitrary functions on the state space. Then*

1. [**Maximum-Norm Contraction**] $\|TJ - T\bar{J}\|_\infty \leq \alpha \|J - \bar{J}\|_\infty$.
2. [**Monotonicity**] *If $J \geq \bar{J}$, we have $TJ \geq T\bar{J}$.*

A number of key results in dynamic programming follow from the maximum-norm contraction and monotonicity properties. Of particular relevance to the present study are the results listed in the following corollary.

Corollary 2.1. *[4]*

1. *The operator T has a unique fixed point (given by J^*).*
2. *For any J, $T^\infty J = J^*$.*
3. *For any J, if $TJ \geq J$, then $J^* \geq T^t J$, for all $t \in \{0, 1, 2, ...\}$.*

Remark 2.1 Since T_u corresponds to the dynamic operator T for a system with a single policy, Theorem 2.1 and Corollary 2.1 also hold for T_u, with J_u replacing J^*.

There are several approaches to solving Bellman's equation. However, dynamic programming techniques suffer from the *curse of dimensionality*: the number of states in the system grows exponentially in the number of state variables, rendering computation and storage of the optimal value function intractable in the face of problems of practical scale.

One approach to dealing with the curse of dimensionality is to generate scoring functions within a parameterized class of functions, in a spirit similar to that of statistical regression. In particular, to approximate the optimal value function J^*, one would design a parameterized class of functions $\tilde{J} : \mathcal{S} \times \Re^K \mapsto \Re$, and then compute a parameter vector $r \in \Re^K$ to "fit" the optimal value function; i.e., so that

$$\tilde{J}(\cdot, r) \approx J^*.$$

We consider a parameterized class of functions known as a *linear architecture.* Given a collection of basis functions $\phi_i : \mathcal{S} \mapsto \Re, i = 1, \ldots, K$, we consider scoring functions representable as linear combinations of the basis functions:

$$\tilde{J}(x, r) = \sum_{i=1}^{K} \phi_i(x) r_i.$$

We define a matrix $\Phi \in \Re^{|\mathcal{S}| \times K}$ given by

$$\Phi = \begin{bmatrix} | & & | \\ \phi_1 & \vdots & \phi_K \\ | & & | \end{bmatrix},$$

i.e., the basis functions are stored as columns of matrix Φ, and each row corresponds to the basis functions evaluated at a different state x. In matrix notation, we would like to find r such that $J^* \approx \Phi r$.

It is clear that the choice of basis functions imposes an upper bound on how good an approximation to the optimal value function we can get. Unfortunately, choosing basis functions remains an empirical task. A suitable choice requires some practical experience or theoretical analysis that provides rough information on the shape of the function to be approximated. "Regularities" associated with the function, for example, can guide the choice of representation.

The focus of this dissertation is on an algorithm for computing an appropriate parameter vector $r \in \Re^k$, given a pre-specified collection of basis functions. Despite the similarities with statistical regression, approximating the optimal value function poses extra difficulties, as one cannot simply sample pairs $(x, J^*(x))$ and choose r to minimize, for instance, the squared error over the samples. Instead, approximate dynamic programming algorithms are generally inspired by exact dynamic programming algorithms, which are adapted to account for the use of approximation architectures. In the next section, we introduce approximate linear programming, the approximate dynamic programming algorithm that is the main focus of this dissertation.

2.3 Approximate Linear Programming

Approximate dynamic programming algorithms are typically inspired by exact dynamic programming algorithms. An algorithm of particular relevance to this work makes use of linear programming, as we will now discuss.

Consider the problem

$$\begin{array}{cl} \max_{J \in \Re^{|\mathcal{S}|}} & c^T J \\ \text{s.t.} & TJ \geq J, \end{array} \tag{2.2}$$

where c is a vector with positive components, which we will refer to as *state-relevance weights*. It can be shown that any feasible J satisfies $J \leq J^*$. It follows that, for any set of positive weights c, J^* is the unique solution to (2.2). This linear programming approach to exact dynamic programming has been extensively studied in the literature [9, 17, 18, 19, 26, 34].

Note that T is a nonlinear operator, and therefore the constrained optimization problem written above is not a linear program. However, it is easy to reformulate the constraints to transform the problem into a linear program. In particular, noting that each constraint

$$(TJ)(x) \geq J(x)$$

is equivalent to a set of constraints

$$g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x, y) J(y) \geq J(x), \qquad \forall a \in \mathcal{A}_x,$$

we can rewrite the problem as

$$\begin{aligned} \max \quad & c^T J \\ \text{s.t.} \quad & g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y) J(y) \geq J(x), \quad \forall x \in S, a \in \mathcal{A}_x. \end{aligned}$$

We will refer to this problem as the *exact LP*.

The curse of dimensionality has tremendous impact on the exact LP. This problem involves prohibitively large numbers of variables and constraints: as many variables as the number of states in the system, and as many constraints as the number of state-action pairs. The approximation algorithm we study reduces dramatically the number of variables.

With an aim of computing a weight vector $\tilde{r} \in \Re^K$ such that $\Phi\tilde{r}$ is a close approximation to J^*, one might pose the following optimization problem

$$\begin{aligned} \max_{r \in \Re^K} \quad & c^T \Phi r \\ \text{s.t.} \quad & T\Phi r \geq \Phi r. \end{aligned} \tag{2.3}$$

Given a solution $\tilde{r}$, one might then hope to generate near-optimal decisions according to the scoring function $\Phi\tilde{r}$.

As with the case of exact dynamic programming, the optimization problem (2.3) can be recast as a linear program

$$\begin{aligned} \max \quad & c^T \Phi r \\ \text{s.t.} \quad & g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y) \geq (\Phi r)(x), \quad \forall x \in S, a \in \mathcal{A}_x. \end{aligned}$$

We will refer to this problem as the *approximate LP*. Note that, although the number of variables is reduced to K, the number of constraints remains as large as in the exact LP. Fortunately, most of the constraints become inactive, and solutions to the approximate LP can be approximated efficiently. Linear programs involving few variables and a large number of constraints are often tractable via constraint generation. In the specific case of the approximate LP, we show in Chapter 5 how

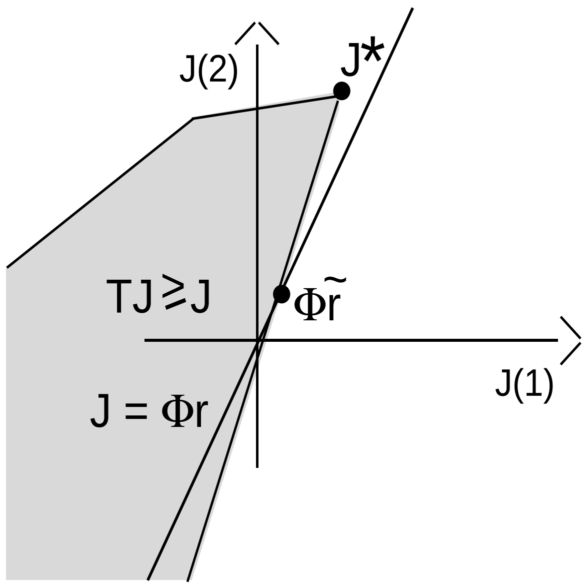


Figure 2.2: Graphical interpretation of approximate linear programming

the special structure of dynamic programming can be exploited in an efficient constraint sampling algorithm that leads to good approximations to the approximate LP solution.

Figure 2.2 offers an interpretation of exact and approximate linear programming. We have a system with two states, and the plane represents the space of all functions of the states. The shaded area represents the feasible region of the exact LP. J^* is the pointwise maximum over the feasible region. The approximate LP corresponds to the addition of a new constraint to the exact LP: we constrain solutions to the hyperplane $J = \Phi r, r \in \Re^k$. The approximate LP finds a solution $\tilde{J} = \Phi \tilde{r}$, over the intersection of the feasible region of the exact LP with $J = \Phi r$.

In the next three chapters we study different aspects of the approximate LP. We start in Chapter 3 by analyzing the role of state-relevance weights in the approximation algorithm. In Chapter 4 we provide performance guarantees for the approximate LP regarding the distance from the scoring function $\Phi \tilde{r}$ to the optimal value function. We finally address the issue of how to deal with the huge number of constraints involved in the approximate LP in Chapter 5.

Chapter 3

State-Relevance Weights

Optimizing multiple objectives simultaneously imposes that certain tradeoffs must be made: unless all objectives are completely aligned, there will be conflict and the need to compromise on some of them. Approximating the optimal value function over a large domain such as the state space poses the same problem. With only limited approximation capacity, we cannot expect to obtain an approximation that is uniformly good throughout the state space; in particular, the maximum error over all states can become arbitrarily large as the problem dimensions increase. Therefore we face the question of how to balance the accuracy of the approximation over different portions of the state space. This chapter is dedicated to addressing this issue.

We first show how approximate linear programming provides the opportunity for assigning different weights — *state-relevance weights* — to approximation errors over different portions of the state space, therefore allowing for emphasis regions of greater importance. We then develop a bound on the performance losses resulting from use of a scoring function generated by approximate linear programming instead of the optimal value function. We finally provide two examples motivating heuristics for identifying what regions of the state space should be assigned higher state-relevance weights.

3.1 State-Relevance Weights in the Approximate LP

In the exact LP, for any vector c with positive components, maximizing $c^T J$ yields J^*. In other words, the choice of state-relevance weights does not influence the solution. The same statement does not hold for the approximate LP. In fact, as we will demonstrate in this chapter, the choice of state-relevance weights bears a significant impact on the quality of the resulting approximation.

To motivate the role of state-relevance weights, let us start with a lemma that offers an interpretation of their function in the approximate LP. This lemma makes use of a norm $\|\cdot\|_{1,c}$, defined by

$$\|J\|_{1,c} = \sum_{x \in \mathcal{S}} c(x)|J(x)|.$$

Lemma 3.1. *A vector $\tilde{r}$ solves*

$$\begin{array}{ll} \max & c^T \Phi r \\ \text{s.t.} & T\Phi r \geq \Phi r, \end{array}$$

if and only if it solves

$$\begin{array}{ll} \min & \|J^* - \Phi r\|_{1,c} \\ \text{s.t.} & T\Phi r \geq \Phi r. \end{array}$$

Proof: It is well known that the dynamic programming operator T is monotonic. From this and the fact that T is a contraction with fixed point J^*, it follows that, for any J with $J \leq TJ$, we have

$$J \leq TJ \leq T^2 J \leq ... \leq J^*.$$

Hence, any r that is a feasible solution to the optimization problems under consideration satisfies $\Phi r \leq J^*$. It follows that

$$\|J^* - \Phi r\|_{1,c} = \sum_{x \in \mathcal{S}} c(x)|J^*(x) - (\Phi r)(x)| = c^T J^* - c^T \Phi r,$$

and maximizing $c^T \Phi r$ is therefore equivalent to minimizing $\|J^* - \Phi r\|_{1,c}$. □

The preceding lemma points to an interpretation of the approximate LP as the minimization of a certain weighted norm of the approximation error, with weights equal to the state-relevance weights. This suggests that c specifies the tradeoff in the quality of the approximation across different states, and we can lead the algorithm to generate better approximations in a region of the state space by assigning relatively larger weight to that region. In the next sections, we identify states that should be weighted heavily to improve performance of the policy generated by the approximate LP.

3.2 On The Quality of Policies Generated by ALP

In deriving a scoring function as a substitute to the optimal value function, a central question is how to compare different scoring functions. A possible measure of quality is the distance to the optimal value function; intuitively, we expect that the better the scoring function captures the real long-run advantage of being in a given state, the better the policy it generates. A more direct measure is a comparison between the actual costs incurred by using the greedy policy associated with the scoring function and those incurred by an optimal policy. In this section, we provide a bound on the cost increase incurred by using scoring functions generated by approximated linear programming.

Recall that we are interested in minimizing discounted costs over infinite horizon. Complete information about the costs associated with a policy u is provided the function J_u (2.1), which provides the expected infinite-horizon discounted cost incurred by using policy u as a function of the initial state in the system.

Comparing costs associated with two different policies requires comparison of two functions on the state space. In turn, comparison of functions involves choice of a metric defined on the space of these functions. We consider as a measure of the quality of policy u the expected increase in the infinite-horizon discounted cost, conditioned on the initial state of the system being distributed according to a probability distribution ν; i.e.,

$$\mathrm{E}_{X\sim\nu}\left[J_u(X) - J^*(X)\right] = \|J_u - J^*\|_{1,\nu}.$$

It will be useful to define a measure $\mu_{u,\nu}$ over the state space associated with each policy u and probability distribution ν, given by

$$\mu_{u,\nu}^T = (1-\alpha)\nu^T \sum_{t=0}^{\infty} \alpha^t P_u^t. \tag{3.1}$$

Note that, since $\sum_{t=0}^{\infty} \alpha^t P_u^t = (I - \alpha P_u)^{-1}$, we also have

$$\mu_{u,\nu}^T = (1-\alpha)\nu^T (I - \alpha P_u)^{-1}.$$

The measure $\mu_{u,\nu}$ captures the expected frequency of visits to each state when the system runs under policy u, conditioned on the initial state being distributed according to ν. Future visits are discounted according to the discount factor α.

Lemma 3.2. *$\mu_{u,\nu}$ is a probability distribution.*

Proof: Let e be the vector of all ones. Then we have

$$\begin{aligned}
\sum_{x\in\mathcal{S}} \mu_{u,\nu}(x) &= (1-\alpha)\nu^T \sum_{t=0}^{\infty} \alpha^t P_u^t e \\
&= (1-\alpha)\nu^T \sum_{t=0}^{\infty} \alpha^t e \\
&= (1-\alpha)\nu^T (1-\alpha)^{-1} e \\
&= 1,
\end{aligned}$$

and the claim follows. □

We are now poised to prove the following bound on the expected cost increase associated with policies generated by approximate linear programming.

Theorem 3.1. *Let $J : \mathcal{S} \mapsto \Re$ be such that $TJ \geq J$. Then*

$$\|J_{u_J} - J^*\|_{1,\nu} \leq \frac{1}{1-\alpha}\|J - J^*\|_{1,\mu_{u_J,\nu}}. \tag{3.2}$$

Proof: We have

$$\begin{aligned}
\|J_{u_J} - J\|_{1,\mu_{u_J,\nu}} &\leq \|J_{u_J} - T_{u_J}J\|_{1,\mu_{u_J,\nu}} + \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
&= \|T_{u_J}J_{u_J} - T_{u_J}J\|_{1,\mu_{u_J,\nu}} + \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
&= \alpha\|P_u(J_{u_J} - J)\|_{1,\mu_{u_J,\nu}} + \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}}. \qquad (3.3)
\end{aligned}$$

where we first applied the triangle inequality and then $J_{u_J} = T_{u_J}J_{u_J}$.

By Corollary 2.1, since $J \leq TJ$, we have $J \leq J^* \leq J_{u_J}$. It follows that

$$\|P_u(J_{u_J} - J)\|_{1,\mu_{u_J,\nu}} = \mu_{u_J,\nu}^T P_u(J_{u_J} - J). \tag{3.4}$$

Combining (3.3) and (3.4), we get

$$\begin{aligned}
\|J_{u_J} - J\|_{1,\mu_{u_J,\nu}} &\leq \alpha\mu_{u_J,\nu}^T P_u(J_{u_J} - J) + \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
\mu_{u_J,\nu}^T(J_{u_J} - J) &\leq \alpha\mu_{u_J,\nu}^T P_u(J_{u_J} - J) + \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
\mu_{u_J,\nu}^T(I - \alpha P_u)(J_{u_J} - J) &\leq \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
(1-\alpha)\nu^T(I - \alpha P_u)^{-1}(I - \alpha P_u)(J_{u_J} - J) &\leq \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
(1-\alpha)\nu^T(J_{u_J} - J) &\leq \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \\
(1-\alpha)\|J_{u_J} - J\|_{1,\nu} &\leq \|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}}. \qquad (3.5)
\end{aligned}$$

Finally, we have $J \leq TJ = T_{u_J}J \leq J^*$, so that

$$\|T_{u_J}J - J\|_{1,\mu_{u_J,\nu}} \leq \|J^* - J\|_{1,\mu_{u_J,\nu}}, \tag{3.6}$$

and

$$\|J_{u_J} - J^*\|_{1,\nu} \leq \|J_{u_J} - J\|_{1,\nu}. \tag{3.7}$$

Combining (3.5), (3.6) and (3.7), we get

$$\begin{aligned}
\|J_{u_J} - J^*\|_{1,\nu} &\leq \|J_{u_J} - J\|_{1,\nu} \\
&\leq \frac{1}{1-\alpha}\|T_{u_J} J - J\|_{1,\mu_{u_J,\nu}} \\
&\leq \frac{1}{1-\alpha}\|J^* - J\|_{1,\mu_{u_J,\nu}},
\end{aligned}$$

and the claim follows. □

Theorem 3.1 has some interesting implications. Recall from Lemma 3.1 that the approximate LP generates a scoring function $\Phi\tilde{r}$ minimizing $\|\Phi r - J^*\|_{1,c}$ over the feasible region; contrasting this result with the bound on the increase in costs (3.2), we might want to choose state-relevance weights c that capture the (discounted) frequency with which different states are expected to be visited. The theorem also sheds light on how beliefs about the initial state of the system can be factored into the approximate LP.

Note that the frequency with which different states are visited in general depends on the policy being used. One possibility is to have an iterative scheme, where the approximate LP is solved multiple times with state-relevance weights adjusted according to the intermediate policies being generated. The next section explores a different approach geared towards problems exhibiting special structure.

3.3 Heuristic Choices of State-Relevance Weights

Theorem 3.1 suggests the use of state-relevance weights that emphasize states visited often by reasonable policies. A plausible conjecture is that some problems will exhibit structure making it relatively easy to take guesses about which states are desirable and therefore more likely to be visited often by reasonable policies, and which ones are typically avoided and rarely visited. We present two examples illustrating these ideas.

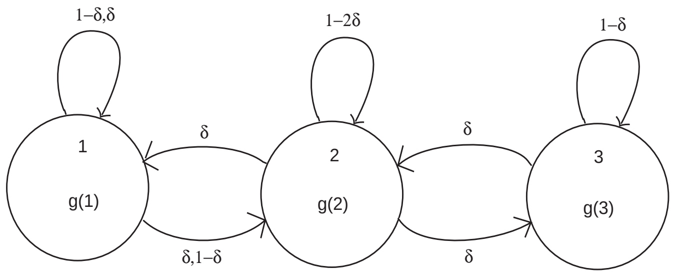


Figure 3.1: Markov decision process for the example studied in Section 3.3.1.

The first example provides some insight into the behavior of the approximate LP when cost-to-go values are much higher in some regions of the state space than in others. Such a situation is likely to arise when the state space is large.

3.3.1 States with Very Large Costs

Consider the problem illustrated in Figure 3.1. The node labels indicate state indices and associated costs. There are three states, 1, 2 and 3, with costs $g(1) < g(2) \ll g(3)$ which do not depend on the action taken. There is one available action for states 2 and 3, and two actions for state 1. We assume that the transition probability δ is small, for instance,

$$\delta = \frac{1-\alpha}{g(3)}.$$

Clearly, it is optimal to take the first action in state 1, which makes the probability of remaining in that state equal to $1-\delta$ and the probability of going to state 2 equal to δ. We can calculate the optimal value function:

$$J^* = \begin{bmatrix} \frac{g(1)}{(1-\alpha)(1+\alpha/g(3))} + \frac{\alpha}{g(3)+\alpha} J_2^* \\ \frac{(1+\alpha/g(3))g(2)}{(1+3\alpha/g(3))(1-\alpha)} + \frac{\alpha(g(1)+g(3))}{(1-\alpha)(g(3)+3\alpha)} \\ \frac{g(3)}{(1-\alpha)(1+\alpha/g(3))} + \frac{\alpha}{g(3)+\alpha} J_2^* \end{bmatrix} \approx \begin{bmatrix} \frac{g(1)}{1-\alpha} \\ \frac{g(2)+\alpha}{1-\alpha} \\ \frac{g(3)}{1-\alpha} \end{bmatrix}.$$

The optimal value function J^* corresponding to $g(1) = 0.5$, $g(2) = 50$, $g(3) = 1000$ and $\alpha = 0.99$ is plotted in Figure 3.2.

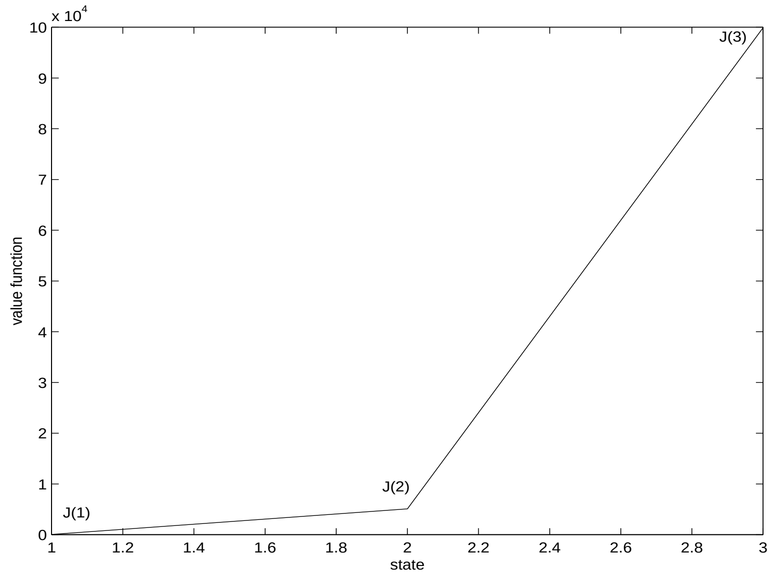


Figure 3.2: Typical optimal value function for $g(1) < g(2) \ll g(3)$.

Let

$$\Phi = \begin{bmatrix} g(1) & 1 \\ g(2) & 1 \\ 2g(2) - g(3) & 1 \end{bmatrix}.$$

The constraints of the approximate LP are given by

$$(1-\alpha) \begin{bmatrix} g(1) + \frac{\alpha(g(1)-g(2))}{g(3)} & 1 \\ \frac{g(1)-\alpha g(2)}{1-\alpha} + \frac{\alpha(g(2)-g(1))}{g(3)} & 1 \\ g(2) + \alpha - \frac{\alpha g(1)}{g(3)} & 1 \\ 2g(2) - g(3) - \alpha + \frac{\alpha g(2)}{g(3)} & 1 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \leq \begin{bmatrix} g(1) \\ g(1) \\ g(2) \\ g(3) \end{bmatrix}.$$

The constraint boundaries intersect the axes at

$$\left(0, \frac{g(1)}{1-\alpha}\right) \text{ and } \left(\frac{g(1)}{(1-\alpha)\left(g(1) + \frac{\alpha(g(1)-g(2))}{g(3)}\right)}\right) \approx \left(\frac{1}{1-\alpha}, 0\right);$$

$$\left(0, \frac{g(1)}{1-\alpha}\right) \text{ and } \left(\frac{g(1)}{g(1) - \alpha g(2) + (1-\alpha)\frac{\alpha(g(2)-g(1))}{g(3)}}, 0\right) \approx (-\epsilon, 0);$$

$$\left(0, \frac{g(2)}{1-\alpha}\right) \text{ and } \left(\frac{1}{(1-\alpha)\left(1 + \frac{\alpha}{g(2)} - \frac{\alpha g(1)}{g(2)g(3)}\right)}, 0\right) \approx \left(\frac{1-\epsilon}{1-\alpha}, 0\right);$$

$$\left(0, \frac{g(3)}{1-\alpha}\right) \text{ and } \left(\frac{g(3)}{(1-\alpha)\left(2g(2) - g(3) - \alpha + \frac{\alpha g(2)}{g(3)}\right)}, 0\right) \approx (-M, 0),$$

where ϵ represents a relatively small positive number and M represents a relatively large positive number. The feasible region for a particular set of parameter values is illustrated in Figure 3.3.

Let us make some observations that are based on the figure but hold true more generally so long as $g(1) < g(2) \ll g(3)$ and δ is small. First, constraint 4 is never active in the positive quadrant, and the only relevant extreme points are at the intersection of constraints 1 and 2 (which is $\eta_1 = (0, \frac{g(1)}{1-\alpha})$) and the intersection of constraints 1 and 3 (which is $\eta_2 \approx (\frac{1}{1-\alpha}, 0)$). Note that the solution is bounded for all positive state-relevance weights c, since $\Phi r \leq J^*$ for all feasible r. Hence, η_1 and η_2 are the only possible solutions. The solution η_2 leads to the best policy. The solution η_1, on the other hand, does not.

It turns out that the choice of state-relevance weights can lead us to either η_1 or η_2. In particular, since

$$c^T \Phi r = (c_1 g(1) + g(2)(c_2 + 2c_3) - c_3 g(3)) r_1 + r_2,$$

when c_3 is relatively large, η_1 is the optimal solution, while η_2 is the optimal solution when c_3 is small. The shape of the optimal value function in Figure 3.2 explains this behavior: the value at state 3 is much higher than that at states 1 and 2, hence there is potentially more opportunity for increasing the objective function of the approximate LP by increasing Φr at state 3 than at states 1 or 2. Therefore it appears that, unless we discount state 3 more heavily than the others, the approximate LP will approximate $J^*(3)$ very closely at the expense of large errors in approximating $J^*(1)$ and $J^*(2)$.

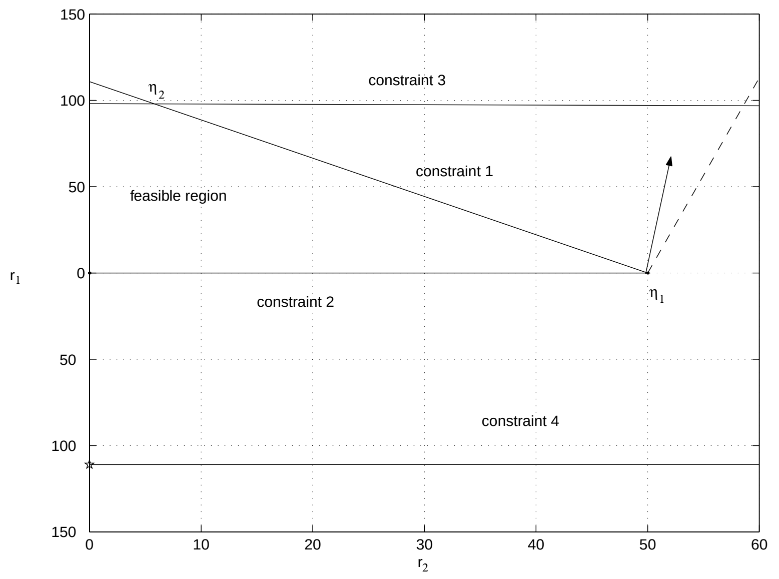


Figure 3.3: Typical feasible region for $g(1) < g(2) << g(3)$. The arrow represents $c^T\Phi$ for a state-relevance weight c which yields a "good" solution for the approximate LP.

3.3.2 Weights Induced by Steady-State Probabilities

The preceding example illustrates how the approximate LP is likely to yield poor approximations for states with relatively low value function unless these states are emphasized by state-relevance weights. From the theoretical bound develop in Theorem 3.1, we know that state-relevance weights should also be chosen so as to emphasize frequently visited states. The next example illustrates how in some cases it may be possible to identify regions of the state space that would be visited often if the system were to be controlled by a "good" policy, and how reasonably good state-relevance weights can be found in that case. We expect structures enabling this kind of procedure to be reasonably common in large-scale problems, in which desirable policies often exhibit some form of "stability," guiding the system to a limited region of the state space and allowing only infrequent excursions from this region.

Consider the problem illustrated in Figure 3.4. The node labels indicate state indices and associated costs. There are four states and costs for each state do not

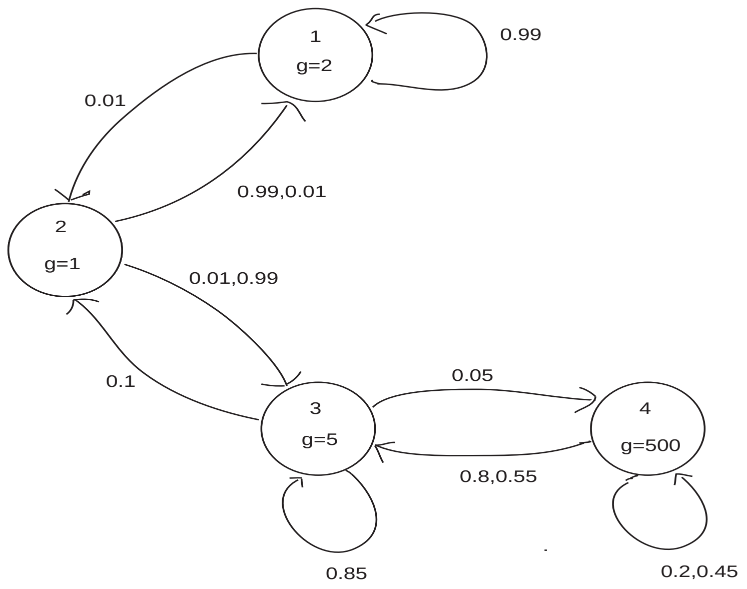


Figure 3.4: Markov decision process for the example studied in Section 3.3.2.

depend on the action taken. There is one available action for states 1 and 3, and two available actions for states 2 and 4. The arc labels indicate transition probabilities. Clearly, it is optimal to take the first action in both states 2 and 4. Let

$$\Phi = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -15 \\ 1 & 100 \end{bmatrix}$$

and $\alpha = 0.99$. One can solve the approximate LP with $c = e$ (e is the vector with every component equal to 1) to obtain an approximation to the optimal value function Φr^e, and then generate a policy u^e that is greedy with respect to Φr^e. This policy takes the right action in state 4 but the wrong action in state 2. The resulting costs-to-go are given by

$$J_{u^e} = \begin{bmatrix} 1560.9 \\ 2935.5 \\ 2978.3 \\ 3565.6 \end{bmatrix}.$$

Let us now discuss how more appropriate state-relevance weights might be chosen and how they can influence the outcome. One possible measure of relevance is given by the stationary distribution induced by an optimal policy. In this simple example, such a distribution is easily calculated and is given by [0.98895 0.00999 0.00100 0.00006]. In general, however, it is difficult to compute such a distribution because the computation typically requires knowledge of an optimal policy.

Is there an alternative approach to generating a good set of weights c? Note that state 4 has a much higher cost than the other states, and we might therefore argue that an optimal policy would try to avoid this state. In addition, based on the pattern of transition probabilities, we can identify the region comprised of states 1, 2 and 3 as "stable," whereas state 4 is "unstable" under any policy; in particular, all actions in state 4 involve a transition with relatively high probability to state 3. This motivates choosing a lower weight for state 4 than for other states. For example, one might try $c = [1\ 1\ 1\ 0.6]$. Denote the solution to the approximate LP by r^c. It turns out that the greedy policy u^c with respect to Φr^c takes the right action in state 2 and the wrong action in state 4, and the value for this policy is

$$J_{u^c} = \begin{bmatrix} 203.2 \\ 206.5 \\ 637.7 \\ 1527.9 \end{bmatrix}$$

The policy u^e bears much higher average cost (28.9066) than u^c (2.0382).

3.4 Closing Remarks

We have seen that the state-relevance weights can influence the quality of an approximation. It is not clear how to find good weights in an efficient manner for larger problems. However, our theoretical results and examples do motivate certain heuristics for selecting weights. Theorem 3.1 suggests the use of an iterative scheme for selection of state-relevance weights, with solution of multiple approximate LP's and weights being designed based on intermediate policies being generated. The examples also suggest that given certain problem structures, certain portions of the state space should be emphasized. First, it seems important to assign low weights to states with high costs, as suggested by Example 3.3.1. Second, if the system when operated by a good policy spends most of its time in a certain subset of the state space, it seems that this subset should be weighted heavily, as suggested by Theorem 3.1 and illustrated by Example 3.3.2. We expect that these heuristics will not generally conflict. In particular, in realistic large-scale problems, states with high costs should avoided, and the system should spend most of its time in a low-cost region of the state space.

Chapter 4

Approximation Error Analysis

It is clear that a scoring function can only be as good as the approximation architecture that defines its representation. Historically, it has been difficult to show the reverse: even if the approximation architecture includes good approximations to the optimal value function, it usually remains uncertain whether approximate dynamic programming algorithms are guaranteed to deliver a reasonable scoring function.

When the optimal value function lies within the span of the basis functions, the approximate LP yields the optimal value function. Unfortunately, it is difficult in practice to select a set of basis functions that contains the optimal value function within its span. Instead, basis functions must be based on heuristics and simplified analysis. One can only hope that the span comes close to the desired value function.

For the approximate LP to be useful, it should deliver good approximations when the optimal value function is near the span of selected basis functions. In this chapter, we develop bounds that ensure desirable results of this kind. We begin in Section 4.2 with a simple bound capturing the fact that, if the vector of all ones is in the span of the basis functions, the error in the result of the approximate LP is proportional to the minimal error given the selected basis functions. Though this result is interesting in its own right, the bound is very loose — perhaps too much so to be useful in practical contexts. In Section 4.3, however, we remedy this situation by providing a tighter bound, which constitutes the main result in this chapter.

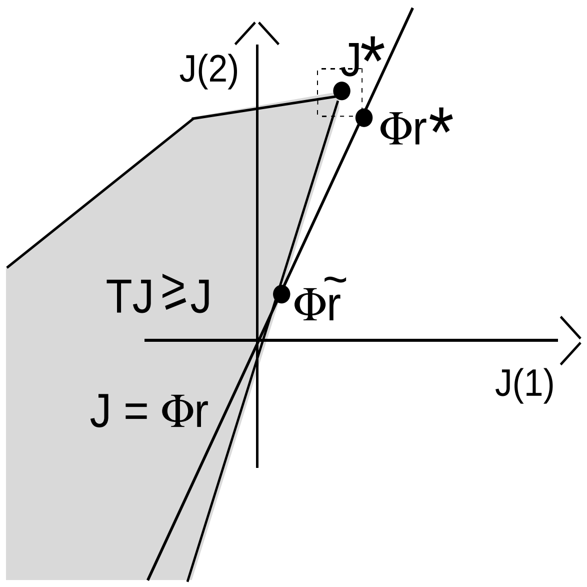


Figure 4.1: Graphical interpretation of approximate linear programming

The central concept enabling the development of the tighter bound is that of a *Lyapunov function*. The definition of a Lyapunov function given here differs slightly from the ones commonly found in the literature. In a closing section, we discuss the intuition behind the use of Lyapunov functions in approximate dynamic programming, and explore connections to standard dynamic programming concepts such as the maximum-norm contraction property of dynamic programming operators and stochastic-shortest-path problems.

4.1 A Graphical Interpretation of ALP

The central question in this chapter is whether having a good selection of basis functions is sufficient for the approximate LP to produce good scoring functions. Figure 4.1 illustrates the issue. Consider an MDP with states 1 and 2. The plane represented in the figure corresponds to the space of all functions over the state space. The shaded area is the feasible region of the exact LP, and J^* is the pointwise maximum over that region. In the approximate LP, we restrict attention to the subspace $J = \Phi r$.

In Figure 4.1, the span of the basis functions comes relatively close to the optimal value function J^*; if we were able to perform, for instance, a maximum-norm projection of J^* onto the subspace $J = \Phi r$, we would obtain the reasonably good scoring function Φr^*. At the same time, the approximate LP yields the scoring function $\Phi \tilde{r}$. The next sections are devoted to the derivation of guarantees that $\Phi \tilde{r}$ is not too much farther from J^* than Φr^* is.

4.2 A Simple Bound

Recall that $\| \cdot \|_\infty$ denotes the maximum norm, defined by $\|J\|_\infty = \max_{x \in \mathcal{S}} |J(x)|$, and that e denotes the vector with every component equal to 1. Our first bound is given by the following theorem.

Theorem 4.1. *Let e be in the span of the columns of Φ and c be a probability distribution. Then, if $\tilde{r}$ is an optimal solution to the approximate LP,*

$$\|J^* - \Phi \tilde{r}\|_{1,c} \leq \frac{2}{1-\alpha} \min_r \|J^* - \Phi r\|_\infty.$$

This bound establishes that when the optimal value function lies close to the span of the basis functions, the approximate LP generates a good approximation. In particular, if the error $\min_r \|J^* - \Phi r\|_\infty$ goes to zero (e.g., as we make use of more and more basis functions) the error resulting from the approximate LP also goes to zero.

Although the bound above offers some support for the linear programming approach, there are some significant weaknesses:

1. The bound calls for an element of the span of the basis functions to exhibit uniformly low error over all states. In practice, however, $\min_r \|J^* - \Phi r\|_\infty$ is typically huge, especially for large-scale problems.

2. The bound does not take into account the choice of state-relevance weights. As demonstrated in the previous chapter, these weights can significantly impact

the quality of the scoring function. A meaningful bound should take them into account.

In Section 4.3, we will state and prove the main result of this chapter, which provides an improved bound that aims to alleviate the shortcomings listed above. First, we prove Theorem 4.1.

Proof of Theorem 4.1

Let r^* be one of the vectors minimizing $\|J^* - \Phi r\|_\infty$ and define $\epsilon = \|J^* - \Phi r^*\|_\infty$. The first step is to find a feasible point $\bar{r}$ such that $\Phi\bar{r}$ is within distance $O(\epsilon)$ of J^*. Since

$$\|T\Phi r^* - J^*\|_\infty \leq \alpha\|\Phi r^* - J^*\|_\infty,$$

we have

$$T\Phi r^* \geq J^* - \alpha\epsilon e. \tag{4.1}$$

We also recall that for any vector J and any scalar k,

$$\begin{aligned} T(J - ke) &= \min_u \{g_u + \alpha P_u(J - ke)\} \\ &= \min_u \{g_u + \alpha P_u J - \alpha ke\} \\ &= \min_u \{g_u + \alpha P_u J\} - \alpha ke \\ &= TJ - \alpha ke. \end{aligned} \tag{4.2}$$

Combining (4.1) and (4.2), we have

$$\begin{aligned} T(\Phi r^* - ke) &= T\Phi r^* - \alpha ke \\ &\geq J^* - \alpha\epsilon e - \alpha ke \\ &\geq \Phi r^* - (1+\alpha)\epsilon e - \alpha ke \\ &= \Phi r^* - ke + [(1-\alpha)k - (1+\alpha)\epsilon]\, e. \end{aligned}$$

Since e is within the span of the columns of Φ, there exists a vector $\bar{r}$ such that

$$\Phi\bar{r} = \Phi r^* - \frac{(1+\alpha)\epsilon}{1-\alpha} e,$$

and $\bar{r}$ is a feasible solution to the approximate LP. By the triangle inequality,

$$\|\Phi\bar{r} - J^*\|_\infty \leq \|J^* - \Phi r^*\|_\infty + \|\Phi r^* - \Phi\bar{r}\|_\infty \leq \epsilon\left(1 + \frac{1+\alpha}{1-\alpha}\right) = \frac{2\epsilon}{1-\alpha}.$$

If $\tilde{r}$ is an optimal solution to the approximate LP, by Lemma 3.1, we have

$$\begin{aligned} \|J^* - \Phi\tilde{r}\|_{1,c} &\leq \|J^* - \Phi\bar{r}\|_{1,c} \\ &\leq \|J^* - \Phi\bar{r}\|_\infty \\ &\leq \frac{2\epsilon}{1-\alpha} \end{aligned}$$

where the second inequality holds because c is a probability distribution. The result follows. $\square$

4.3 An Improved Bound

To set the stage for the development of an improved bound, let us establish some notation. First, we introduce a weighted maximum norm, defined by

$$\|J\|_{\infty,\gamma} = \max_{x\in\mathcal{S}} \gamma(x)|J(x)|, \tag{4.3}$$

for any $\gamma : \mathcal{S} \mapsto \Re^+$. As opposed to the maximum norm employed in Theorem 4.1, this norm allows for uneven weighting of errors across the state space.

We also introduce an operator H, defined by

$$(HV)(x) = \max_{a\in\mathcal{A}_x} \sum_{y\in\mathcal{S}} P_a(x,y)V(y),$$

for all $V : \mathcal{S} \mapsto \Re$. For any V, $(HV)(x)$ represents the maximum expected value of $V(Y)$ if the current state is x and Y is a random variable representing the next state.

For each $V : \mathcal{S} \mapsto \Re$, we define a scalar β_V given by

$$\beta_V = \max_x \frac{\alpha(HV)(x)}{V(x)}. \tag{4.4}$$

We can now introduce the notion of a "Lyapunov function," as follows.

Definition 4.1 (Lyapunov function) We call $V : \mathcal{S} \mapsto \Re^+$ a Lyapunov function if $\beta_V < 1$.

Our definition of a Lyapunov function translates into the condition that there exist $V > 0$ and $\beta < 1$ such that

$$\alpha(HV)(x) \leq \beta V(x), \qquad \forall x \in \mathcal{S}. \tag{4.5}$$

If α were equal to 1, this would look like a Lyapunov stability condition: the maximum expected value $(HV)(x)$ at the next time step must be less than the current value $V(x)$. In general, α is less than 1, and this introduces some slack in the condition.

Our error bound for the approximate LP will grow proportionately with $1/(1-\beta_V)$, and we therefore want β_V to be small. Note that β_V becomes smaller as the $(HV)(x)$'s become small relative to the $V(x)$'s; β_V conveys a degree of "stability," with smaller values representing stronger stability. Therefore our bound suggests that, the more stable the system is, the easier it may be for the approximate LP to generate a good scoring function.

We are now ready to state our main result. For any given function V mapping $\mathcal{S}$ to positive reals, we use $1/V$ as shorthand for a function $x \mapsto 1/V(x)$.

Theorem 4.2. *Let $\tilde{r}$ be a solution of the approximate LP. Then, for any $v \in \Re^K$ such that Φv is a Lyapunov function,*

$$\|J^* - \Phi\tilde{r}\|_{1,c} \leq \frac{2c^T\Phi v}{1-\beta_{\Phi v}} \min_r \|J^* - \Phi r\|_{\infty,1/\Phi v}. \tag{4.6}$$

Let us now discuss how this new theorem addresses the shortcomings of Theorem 4.1 listed in the previous section. We treat in turn the two items from the aforementioned list.

1. The norm $\|\cdot\|_\infty$ appearing in Theorem 4.1 is undesirable largely because it does not scale well with problem size. In particular, for large problems, the optimal value function can take on huge values over some (possibly infrequently visited) regions of the state space, and so can approximation errors in such regions.

   Observe that the maximum norm of Theorem 4.1 has been replaced in Theorem 4.2 by $\|\cdot\|_{\infty,1/\Phi v}$. Hence, the error at each state is now weighted by the reciprocal of the Lyapunov function value. This should to some extent alleviate difficulties arising in large problems. In particular, the Lyapunov function should take on large values in undesirable regions of the state space — regions where J^* is large. Hence, division by the Lyapunov function acts as a normalizing procedure that scales down errors in such regions.

2. As opposed to the bound of Theorem 4.1, the state-relevance weights do appear in our new bound. In particular, there is a coefficient $c^T\Phi v$ scaling the right-hand-side. In general, if the state-relevance weights are chosen appropriately, we expect that $c^T\Phi v$ will be reasonably small and independent of problem size. We defer to Section 4.4 further qualification of this statement and a discussion of approaches to choosing c in contexts posed by concrete examples.

Proof of Theorem 4.2

The remainder of this section is dedicated to a proof of Theorem 4.2. We begin with a preliminary lemma bounding the effects of applying the dynamic programming operator to two different functions.

Lemma 4.1. *For any J and $\overline{J}$,*

$$|TJ - T\bar{J}| \leq \alpha \max_u P_u |J - \bar{J}|.$$

Proof: Note that, for any J and $\bar{J}$,

$$\begin{aligned} TJ - T\bar{J} &= \min_u \{g_u + \alpha P_u J\} - \min_u \left\{g_u + \alpha P_u \bar{J}\right\} \\ &= g_{u_J} + \alpha P_{u_J} J - g_{u_{\bar{J}}} - \alpha P_{u_{\bar{J}}} \bar{J} \end{aligned}$$

$$\begin{aligned}
&\leq g_{u_{\bar{J}}} + \alpha P_{u_{\bar{J}}} J - g_{u_{\bar{J}}} - \alpha P_{u_{\bar{J}}} \bar{J} \\
&\leq \alpha \max_u P_u (J - \bar{J}) \\
&\leq \alpha \max_u P_u |J - \bar{J}|,
\end{aligned}$$

where u_J and $u_{\bar{J}}$ denote greedy policies with respect to J and $\bar{J}$, respectively. An entirely analogous argument gives us

$$T\bar{J} - TJ \leq \alpha \max_u P_u |J - \bar{J}|,$$

and the result follows. □

Based on the above lemma, we can place the following bound on constraint violations in the approximate LP.

Lemma 4.2. *For any vector V with positive components and any vector J,*

$$TJ \geq J - (\alpha HV + V) \|J^* - J\|_{\infty,1/V}. \tag{4.7}$$

Proof: Note that

$$|J^*(x) - J(x)| \leq \|J^* - J\|_{\infty,1/V} V(x).$$

By Lemma 4.1,

$$\begin{aligned}
|(TJ^*)(x) - (TJ)(x)| &\leq \alpha \max_a \sum_{y \in \mathcal{S}} P_a(x,y) |J^*(y) - J(y)| \\
&\leq \alpha \|J^* - J\|_{\infty,1/V} \max_{a \in \mathcal{A}_x} \sum_{y \in \mathcal{S}} P_a(x,y) V(y) \\
&= \alpha \|J^* - J\|_{\infty,1/V} (HV)(x).
\end{aligned}$$

Letting $\epsilon = \|J^* - J\|_{\infty,1/V}$, it follows that

$$\begin{aligned}
(TJ)(x) &\geq J^*(x) - \alpha\epsilon (HV)(x) \\
&\geq J(x) - \epsilon V(x) - \alpha \epsilon (HV)(x).
\end{aligned}$$

The result follows. □

The next lemma establishes that subtracting an appropriately scaled version of a Lyapunov function from any Φr leads us to the feasible region of the approximate LP.

Lemma 4.3. *Let v be a vector such that Φv is a Lyapunov function, r be an arbitrary vector, and*

$$\overline{r} = r - \|J^* - \Phi r\|_{\infty,1/\Phi v} \left(\frac{2}{1-\beta_{\Phi v}} - 1\right) v.$$

Then,

$$T\Phi\bar{r} \geq \Phi\bar{r}.$$

Proof: Let $\epsilon = \|J^* - \Phi r\|_{\infty,1/\Phi v}$. By Lemma 4.1,

$$\begin{aligned} |(T\Phi r)(x) - (T\Phi\overline{r})(x)| &= \left|(T\Phi r)(x) - \left(T\left[\left(\Phi r - \epsilon\left(\frac{2}{1-\beta_{\Phi v}} - 1\right)\Phi v\right]\right)(x)\right| \\ &\leq \alpha \max_a \sum_{y\in\mathcal{S}} P_a(x,y)\epsilon\left(\frac{2}{1-\beta_{\Phi v}} - 1\right)(\Phi v)(y) \\ &= \alpha\epsilon\left(\frac{2}{1-\beta_{\Phi v}} - 1\right)(H\Phi v)(x), \end{aligned}$$

since Φv is a Lyapunov function and therefore $2/(1-\beta_{\Phi v}) - 1 > 0$. It follows that

$$T\Phi\overline{r} \geq T\Phi r - \alpha\epsilon\left(\frac{2}{1-\beta_{\Phi v}} - 1\right)H\Phi v.$$

By Lemma 4.2,

$$T\Phi r \geq \Phi r - \epsilon\left(\alpha H\Phi v + \Phi v\right),$$

and therefore

$$\begin{aligned} T\Phi\overline{r} &\geq \Phi r - \epsilon\left(\alpha H\Phi v + \Phi v\right) - \alpha\epsilon\left(\frac{2}{1-\beta_{\Phi v}} - 1\right)H\Phi v \\ &= \Phi\overline{r} - \epsilon\left(\alpha H\Phi v + \Phi v\right) + \epsilon\left(\frac{2}{1-\beta_{\Phi v}} - 1\right)(\Phi v - \alpha H\Phi v) \\ &\geq \Phi\overline{r} - \epsilon\left(\alpha H\Phi v + \Phi v\right) + \epsilon(\Phi v + \alpha H\Phi v) \\ &= \Phi\overline{r}, \end{aligned}$$

where the last inequality follows from the fact that $\Phi v - \alpha H \Phi v > 0$ and

$$\begin{aligned}
\frac{2}{1-\beta_{\Phi v}} - 1 &= \frac{2}{1 - \max_x \frac{\alpha (H\Phi v)(x)}{(\Phi v)(x)}} - 1 \\
&= \max_x \frac{2(\Phi v)(x)}{(\Phi v)(x) - \alpha (H \Phi v)(x)} - 1 \\
&= \max_x \frac{(\Phi v)(x) + \alpha (H\Phi v)(x)}{(\Phi v)(x) - \alpha (H \Phi v)(x)}.
\end{aligned}$$

□

Proof of Theorem 4.2: Given the preceding lemmas, we are poised to prove Theorem 4.2. From Lemma 4.3, we know that $\overline{r} = r^* - \|J^* - \Phi r^*\|_{\infty,1/\Phi v} \left(\frac{2}{1-\beta_{\Phi v}} - 1 \right) v$ is a feasible solution for the approximate LP. It follows that

$$\begin{aligned}
\|\Phi \overline{r} - \Phi r^*\|_{\infty,1/\Phi v} &\leq \|J^* - \Phi r^*\|_{\infty,1/\Phi v} \left(\frac{2}{1-\beta_{\Phi v}} - 1 \right) \|\Phi v\|_{\infty,1/\Phi v} \\
&= \|J^* - \Phi r^*\|_{\infty,1/\Phi v} \left(\frac{2}{1-\beta_{\Phi v}} - 1 \right).
\end{aligned}$$

From Lemma 3.1, we have

$$\begin{aligned}
\|J^* - \Phi \tilde{r}\|_{1,c} &\leq \|J^* - \Phi \bar{r}\|_{1,c} \\
&= \sum_x c(x)(\Phi v)(x) \frac{|J^*(x) - (\Phi \bar{r})(x)|}{(\Phi v)(x)} \\
&\leq \left(\sum_x c(x)(\Phi v)(x) \right) \max_x \frac{|J^*(x) - (\Phi \bar{r})(x)|}{(\Phi v)(x)} \\
&= c^T \Phi v \|J^* - \Phi \bar{r}\|_{\infty,1/\Phi v} \\
&\leq c^T \Phi v \left(\|J^* - \Phi r^*\|_{\infty,1/\Phi v} + \|\Phi \bar{r} - \Phi r^*\|_{\infty,1/\Phi v} \right) \\
&\leq c^T \Phi v \left(\|J^* - \Phi r^*\|_{\infty,1/\Phi v} + \|J^* - \Phi r^*\|_{\infty,1/\Phi v} \left(\frac{2}{1-\beta_{\Phi v}} - 1 \right) \right) \\
&\leq \frac{2}{1-\beta_{\Phi v}} c^T \Phi v \|J^* - \Phi r^*\|_{\infty,1/\Phi v}.
\end{aligned}$$

□

4.4 On the Choice of Lyapunov Function

The Lyapunov function Φv plays a central role in the bound of Theorem 4.2. Its choice influences three terms on the right-hand-side of the bound:

1. the error $\min_r \|J^* - \Phi r\|_{\infty,1/\Phi v}$;
2. the term $1/(1-\beta_{\Phi v})$;
3. the inner product $c^T \Phi v$ with the state-relevance weights.

An appropriately chosen Lyapunov function should make all three of these terms relatively small. Furthermore, for the bound to be useful in practical contexts, these terms should not grow much with problem size.

In the following sections, we present three examples involving choices of Lyapunov functions in queueing problems. The intention is to illustrate more concretely how Lyapunov functions might be chosen and that reasonable choices lead to practical error bounds that are independent of the number of states, as well as the number of state variables. The first example involves a single autonomous queue. A second generalizes this to a context with controls. A final example treats a network of queues. In each case, we study the three terms enumerated above and how they scale with the number of states and/or state variables.

4.4.1 An Autonomous Queue

Our first example involves a model of an autonomous (i.e., uncontrolled) queueing system. We consider a Markov process with states $0, 1, ..., N-1$, each representing a possible number of jobs in a queue. The system state x_t evolves according to

$$x_{t+1} = \begin{cases} \min(x_t + 1, N-1), & \text{with probability } p, \\ \max(x_t - 1, 0), & \text{otherwise,} \end{cases}$$

and it is easy to verify that the steady-state probabilities $\pi(0), \ldots, \pi(N-1)$ satisfy

$$\pi(x) = \pi(0) \left(\frac{p}{1-p} \right)^x .$$

If the state satisfies $0 < x < N-1$, a cost $g(x) = x^2$ is incurred. For the sake of simplicity, we assume that costs at the boundary states 0 and $N-1$ are chosen to ensure that the cost-to-go function takes the form

$$J^*(x) = \rho_2 x^2 + \rho_1 x + \rho_0,$$

for some scalars ρ_0, ρ_1, ρ_2 with $\rho_0 > 0$ and $\rho_2 > 0$[1]. We assume that $p < 1/2$ so that the system is "stable." Stability here is taken in a loose sense indicating that the steady-state probabilities are decreasing for all sufficiently large states.

Suppose that we wish to generate an approximation to the optimal value function using the linear programming approach. Further suppose that we have chosen the state-relevance weights c to be the vector π of steady-state probabilities and the basis functions to be $\phi_1(x) = 1$ and $\phi_2(x) = x^2$.

How good can we expect the scoring function $\Phi\tilde{r}$ generated by approximate linear programming to be as we increase the number of states N? First note that

$$\begin{aligned}
\min_r \|J^* - \Phi r\|_{1,c} &\leq \|J^* - (\rho_0\phi_1 + \rho_2\phi_2)\|_{1,c} \\
&= \sum_{x=0}^{N-1} \pi(x)|\rho_1|x \\
&= |\rho_1| \sum_{x=0}^{N-1} \pi(0)\left(\frac{p}{1-p}\right)^x x \\
&\leq |\rho_1| \frac{\frac{p}{1-p}}{1-\frac{p}{1-p}},
\end{aligned}$$

for all N. The last inequality follows from the fact that the summation in the third line corresponds to the expected value of a geometric random variable conditioned on

[1]It is easy to verify that such a choice of boundary conditions is possible. In particular, given the desired functional form for J^*, we can solve for ρ_0, ρ_1, and ρ_2, based on Bellman's equation for states $1, \ldots, N-2$:

$$J^*(x) = x^2 + \alpha(pJ^*(x+1) + (1-p)J^*(x-1), \qquad \forall x = 1, \ldots, N-2.$$

Note that the solution is unique as long as $N > 5$. We can then set $g(0) \equiv J^*(0) - \alpha(pJ^*(1) + (1-p)J^*(0))$ and $g(N-1) \equiv J^*(N-1) - \alpha(pJ^*(N-1) + (1-p)J^*(N-2))$ so that Bellman's equation is also satisfied for states 0 and $N-1$.

its being less than N. Hence, $\min_r \|J^* - \Phi r\|_{1,c}$ is uniformly bounded over N. One would hope that $\|J^* - \Phi \tilde{r}\|_{1,c}$, with $\tilde{r}$ being an outcome of the approximate LP, would be similarly uniformly bounded over N. It is clear that Theorem 4.1 does not offer a uniform bound of this sort. In particular, the term $\min_r \|J^* - \Phi r\|_\infty$ on the right-hand-side grows proportionately with N and is unbounded as N increases. Fortunately, this situation is remedied by Theorem 4.2, which does provide a uniform bound. In particular, as we will show in the remainder of this section, for an appropriate Lyapunov function $V = \Phi v$, the values of $\min_r \|J^* - \Phi r\|_{\infty,1/V}$, $1/(1-\beta_V)$ and $c^T V$ are all uniformly bounded over N, and together these values offer a bound on $\|J^* - \Phi \tilde{r}\|_{1,c}$ that is uniform over N.

We will make use of a Lyapunov function

$$V(x) = x^2 + \frac{2}{1-\alpha},$$

which is clearly within the span of our basis functions ϕ_1 and ϕ_2. Given this choice, we have

$$\begin{aligned}
\min_r \|J^* - \Phi r\|_{\infty,1/V} &\leq \max_{x\geq 0} \frac{|\rho_2 x^2 + \rho_1 x + \rho_0 - \rho_2 x^2 - \rho_0|}{x^2 + 2/(1-\alpha)} \\
&= \max_{x\geq 0} \frac{|\rho_1| x}{x^2 + 2/(1-\alpha)} \\
&\leq \frac{|\rho_1|}{2\sqrt{2/(1-\alpha)}}.
\end{aligned}$$

Hence, $\min_r \|J^* - \Phi r\|_{\infty,1/V}$ is uniformly bounded over N.

We next show that $1/(1-\beta_V)$ is uniformly bounded over N. In order to do that, we find bounds on HV in terms of V. For $0 < x < N-1$, we have

$$\begin{aligned}
\alpha(HV)(x) &= \alpha\left[p\left(x^2 + 2x + 1 + \frac{2}{1-\alpha}\right) + (1-p)\left(x^2 - 2x + 1 + \frac{2}{1-\alpha}\right)\right] \\
&= \alpha\left[x^2 + \frac{2}{1-\alpha} + 1 + 2x(2p-1)\right] \\
&\leq \alpha\left(x^2 + \frac{2}{1-\alpha} + 1\right)
\end{aligned}$$

$$\begin{aligned}
&= V(x)\left(\alpha + \frac{\alpha}{V(x)}\right) \\
&\leq V(x)\left(\alpha + \frac{1}{V(0)}\right) \\
&= V(x)\frac{1+\alpha}{2}.
\end{aligned}$$

For $x = 0$, we have

$$\begin{aligned}
\alpha(HV)(0) &= \alpha\left[p\left(1 + \frac{2}{1-\alpha}\right) + (1-p)\frac{2}{1-\alpha}\right] \\
&= \alpha p + \alpha\frac{2}{1-\alpha} \\
&\leq V(0)\left(\alpha + \frac{1-\alpha}{2}\right) \\
&= V(0)\frac{1+\alpha}{2}.
\end{aligned}$$

Finally, we clearly have

$$\alpha(HV)(N-1) \leq \alpha V(N-1) \leq V(N-1)\frac{1+\alpha}{2},$$

since the only possible transitions from state $N-1$ are to states $x \leq N-1$ and V is a nondecreasing function. Therefore, $\beta_V \leq (1+\alpha)/2$ and $1/(1-\beta_V)$ is uniformly bounded on N.

We now treat $c^T V$. Note that for $N \geq 1$,

$$\begin{aligned}
c^T V &= \sum_{x=0}^{N-1} \pi(0)\left(\frac{p}{1-p}\right)^x \left(x^2 + \frac{2}{1-\alpha}\right) \\
&= \frac{1 - p/(1-p)}{1 - [p/(1-p)]^N} \sum_{x=0}^{N-1} \left(\frac{p}{1-p}\right)^x \left(x^2 + \frac{2}{1-\alpha}\right) \\
&\leq \frac{1 - p/(1-p)}{1 - p/(1-p)} \sum_{x=0}^{\infty} \left(\frac{p}{1-p}\right)^x \left(x^2 + \frac{2}{1-\alpha}\right) \\
&= \frac{1-p}{1-2p}\left(\frac{2}{1-\alpha} + 2\frac{p^2}{(1-2p)^2} + \frac{p}{1-2p}\right),
\end{aligned}$$

so $c^T V$ is uniformly bounded for all N.

4.4.2 A Controlled Queue

In the previous example, we treated the case of an autonomous queue and showed how the terms involved in the error bound of Theorem 4.2 are uniformly bounded on the number of states N. We now address a more general case in which we can control the queue service rate. For any time t and state $0 < x_t < N-1$, the next state is given by

$$x_{t+1} = \begin{cases} x_t - 1, & \text{with probability } q(x_t), \\ x_t + 1, & \text{with probability } p, \\ x_t, & \text{otherwise.} \end{cases}$$

From state 0, a transition to state 1 or 0 occurs with probabilities p or $1-p$, respectively. From state $N-1$, a transition to state $N-2$ or $N-1$ occurs with probabilities $q(N-2)$ or $1-q(N-2)$, respectively. The arrival probability p is the same for all states and we assume that $p < 1/2$. The action to be chosen in each state x is the departure probability or service rate $q(x)$, which takes values in a finite set $\{q_i, i = 1, ..., A\}$. We assume that $q_A = 1 - p > p$, therefore the queue is "stabilizable". The cost incurred at state x if action q is taken is given by

$$g(x, q) = x^2 + m(q),$$

where m is a nonnegative and increasing function.

As discussed before, our objective is to show that the terms involved in the error bound of Theorem 4.2 are uniformly bounded over N. We start by finding a suitable Lyapunov function based on our knowledge of the problem structure. In the autonomous case, the choice of the Lyapunov function was motivated by the fact that the optimal value function was a quadratic. We now proceed to show that in the controlled case, J^* can be bounded above by a quadratic

$$J^*(x) \leq \rho_2 x^2 + \rho_1 x + \rho_0$$

for some $\rho_0 > 0$, ρ_1 and $\rho_2 > 0$ that are constant independent of the queue buffer size $N-1$. Note that J^* is bounded above by the value of a policy $\bar{\mu}$ that takes action

$q(x) = 1 - p$ for all x, hence it suffices to find a quadratic upper bound for the value of this policy. We will do so by making use of the fact that for any policy μ and any vector J, $T_\mu J \leq J$ implies $J \geq J_\mu$. Take

$$\begin{aligned}
\rho_2 &= \frac{1}{1-\alpha}, \\
\rho_1 &= \frac{\alpha\left[2\rho_2(2p-1)\right]}{1-\alpha}, \\
\rho_0 &= \max\left(\frac{\alpha p(\rho_2+\rho_1)}{1-\alpha}, \frac{m(1-p)+\alpha\left[\rho_2+\rho_1(2p-1)\right]}{1-\alpha}\right).
\end{aligned}$$

For any state x such that $0 < x < N-1$, we can verify that

$$\begin{aligned}
J(x) - (T_{\bar{\mu}}J)(x) &= \rho_0(1-\alpha) - m(1-p) - \alpha\left[\rho_2+\rho_1(2p-1)\right] \\
&\geq \frac{m(1-p)+\alpha\left[\rho_2+\rho_1(2p-1)\right]}{1-\alpha}(1-\alpha) - m(1-p) - \\
&\quad -\alpha\left[\rho_2+\rho_1(2p-1)\right] \\
&= 0.
\end{aligned}$$

For state $x = N-1$, note that if $N > 1 - \rho_1/2\rho_2$ we have $J(N) > J(N-1)$ and

$$\begin{aligned}
J(N-1) - (T_{\bar{\mu}}J)(N-1) &= J(N-1) - (N-1)^2 - m(1-p) - \qquad (4.8) \\
&\quad -\alpha\left[(1-p)J(N-2) + pJ(N-1)\right] \\
&\geq J(N-1) - (N-1)^2 - m(1-p) - \qquad (4.9) \\
&\quad -\alpha\left[(1-p)J(N-2) + pJ(N)\right] \\
&= \rho_0(1-\alpha) - m(1-p) - \alpha\left[\rho_2+\rho_1(2p-1)\right] \\
&\geq 0.
\end{aligned}$$

Finally, for state $x = 0$ we have

$$\begin{aligned}
J(0) - (T_{\bar{\mu}}J)(0) &= (1-\alpha)\rho_0 - \alpha p(\rho_2+\rho_1) \\
&\geq (1-\alpha)\frac{\alpha p(\rho_2+\rho_1)}{1-\alpha} - \alpha p(\rho_2+\rho_1) \\
&= 0.
\end{aligned}$$

It follows that $J \geq T_\mu J$, and for all $N > 1 - \rho_1/2\rho_2$,

$$0 \leq J^* \leq J_{\bar{\mu}} \leq J = \rho_2 x^2 + \rho_1 x + \rho_0.$$

A natural choice of Lyapunov function is, as in the previous example, $V(x) = x^2 + C$ for some $C > 0$. It follows that

$$\begin{aligned} \min_r \|J^* - \Phi r\|_{\infty,1/V} &\leq \|J^*\|_{\infty,1/V} \\ &\leq \max_{x\geq 0} \frac{\rho_2 x^2 + \rho_1 x + \rho_0}{x^2 + C} \\ &< \rho_2 + \frac{\rho_1}{2\sqrt{C}} + \frac{\rho_0}{C}. \end{aligned}$$

Now note that

$$\begin{aligned} \alpha(HV)(x) &\leq \alpha\left[p(x^2 + 2x + 1 + C) + (1-p)(x^2 + C)\right] \\ &= V(x)\left(\alpha + \frac{\alpha p(2x+1)}{x^2 + C}\right) \end{aligned}$$

and for C sufficiently large and independent of N, there is $\beta < 1$ also independent of N such that $\alpha HV \leq \beta V$ and $1/(1-\beta)$ is uniformly bounded on N.

It remains to be shown that $c^T V$ is uniformly bounded on N. For that, we need to specify the state-relevance vector c. As in the case of the autonomous queue, we might want it to be close to the steady-state distribution of the states under the optimal policy. Clearly, it is not easy to choose state-relevant weights in that way since we do not know the optimal policy. Alternatively, we will use the general shape of the steady-state distribution to generate sensible state-relevance weights.

Let us analyze the infinite buffer case and show that, under some stability assumptions, there should be a geometric upper bound for the tail of steady-state distribution; we expect that results for finite (large) buffers should be similar if the system is stable, since in this case most of the steady-state distribution will be concentrated on relatively small states. Let us assume that the system under the optimal policy is indeed stable – that should generally be the case if the discount factor is large. For a queue with infinite buffer the optimal service rate $q(x)$ is nondecreasing in x [4], and stability therefore implies that

$$q(x) \geq q(x_0) > p$$

for all $x \geq x_0$ and some sufficiently large x_0. It is easy then to verify that the tail of the steady-state distribution has an upper bound with geometric decay since it should satisfy

$$\pi(x)p = \pi(x+1)q(x+1),$$

and therefore

$$\frac{\pi(x+1)}{\pi(x)} \leq \frac{p}{q(x_0)} < 1,$$

for all $x \geq x_0$. Thus a reasonable choice of state-relevance weights is $c(x) = \pi(0)\xi^x$, where $\pi(0) = \frac{1-\xi}{1-\xi^N}$ is a normalizing constant making c a probability distribution. In this case,

$$\begin{aligned} c^T V &= \mathrm{E}\left[X^2 + C \mid X < N\right] \\ &\leq 2\frac{\xi^2}{(1-\xi)^2} + \frac{\xi}{1-\xi} + C, \end{aligned}$$

where X represents a geometric random variable with parameter $1-\xi$. We conclude that $c^T V$ is uniformly bounded on N.

4.4.3 A Queueing Network

Both previous examples involved one-dimensional state spaces and had terms of interest in the approximation error bound uniformly bounded over the number of states. We now consider a queueing network with d queues and finite buffers of size B to determine the impact of dimensionality on the terms involved in the error bound of Theorem 4.2.

We assume that the number of exogenous arrivals occuring in any time step has expected value less than or equal to Ad, for a finite A. The state $x \in \Re^d$ indicates the number of jobs in each queue. The cost per stage incurred at state x is given by

$$g(x) = \frac{|x|}{d} = \frac{1}{d}\sum_{i=1}^{d} x_i,$$

the average number of jobs per queue.

Let us first consider the optimal value function J^* and its dependency on the number of state variables d. Our goal is to establish bounds on J^* that will offer some guidance on the choice of a Lyapunov function V that keeps the error $\min_r \|J^* - \Phi r\|_{\infty,1/V}$ small. Since $J^* \geq 0$, we will only derive upper bounds.

Instead of carrying the buffer size B throughout calculations, we will consider the infinite buffer case. The optimal value function for the finite buffer case should be bounded above by that of the infinite buffer case, as having finite buffers corresponds to having jobs arriving at a full queue discarded at no additional cost.

We have

$$\mathrm{E}_x\left[|x_t|\right] \leq |x| + Adt,$$

since the expected total number of jobs at time t cannot exceed the total number of jobs at time 0 plus the expected number of arrivals between 0 and t, which is less than or equal to Adt. Therefore we have

$$\begin{aligned}
\mathrm{E}_x\left[\sum_{t=0}^{\infty} \alpha^t |x_t|\right] &= \sum_{t=0}^{\infty} \alpha^t \mathrm{E}_x\left[|x_t|\right] \\
&\leq \sum_{t=0}^{\infty} \alpha^t (|x| + Adt) \\
&= \frac{|x|}{1-\alpha} + \frac{Ad}{(1-\alpha)^2}. \qquad (4.10)
\end{aligned}$$

The first equality holds because $|x_t| \geq 0$ for all t; by the monotone convergence theorem, we can interchange the expectation and the summation. We conclude from (4.10) that the optimal value function in the infinite buffer case should be bounded above by a linear function of the state; in particular,

$$0 \leq J^*(x) \leq \frac{\rho_1}{d}|x| + \rho_0,$$

for some positive scalars ρ_0 and ρ_1 independent of the number of queues d.

As discussed before, the optimal value function in the infinite buffer case provides an upper bound for the optimal value function in the case of finite buffers of size B. Therefore, the optimal value function in the finite buffer case should be bounded above by the same linear function regardless of the buffer size B.

As in the previous examples, we will establish bounds on the terms involved in the error bound of Theorem 4.2. We consider a Lyapunov function $V(x) = \frac{1}{d}|x| + C$ for some constant $C > 0$, which implies

$$\begin{aligned}\min_r \|J^* - \Phi r\|_{\infty,1/V} &\leq \|J^*\|_{\infty,1/V} \\ &\leq \max_{x\geq 0} \frac{\rho_1|x| + d\rho_0}{|x| + dC} \\ &\leq \rho_1 + \frac{\rho_0}{C},\end{aligned}$$

and the bound above is independent of the number of queues in the system.

Now let us study β_V. We have

$$\begin{aligned}\alpha(HV)(x) &\leq \alpha\left(\frac{|x| + Ad}{d} + C\right) \\ &\leq V(x)\left(\alpha + \frac{\alpha A}{\frac{|x|}{d} + C}\right) \\ &\leq V(x)\left(\alpha + \frac{\alpha A}{C}\right),\end{aligned}$$

and it is clear that, for C sufficiently large and independent of d, there is a $\beta < 1$ independent of d such that $\alpha HV \leq \beta V$, and therefore $\frac{1}{1-\beta_V}$ is uniformly bounded on d.

Finally, let us consider $c^T V$. We expect that under some stability assumptions, the tail of the steady-state distribution will have an upper bound with geometric decay [6] and we take $c(x) = \left(\frac{1-\xi}{1-\xi^{B+1}}\right)^d \xi^{|x|}$. The state-relevance weights c are equivalent to the conditional joint distribution of d independent and identically distributed geometric random variables conditioned on the event that they are all less than $B+1$. Therefore,

$$\begin{aligned} c^T V &= E\left[\frac{1}{d}\sum_{i=1}^{d} X_i + C \;\middle|\; X_i < B+1, i = 1, ..., d\right] \\ &< E[X_1] + C \\ &= \frac{\xi}{1-\xi} + C, \end{aligned}$$

where $X_i, i = 1, ..., d$ are identically distributed geometric random variables with parameter $1-\xi$. It follows that $c^T V$ is uniformly bounded over the number of queues.

4.5 At the Heart of Lyapunov Functions

In this section, we focus on two central questions on Lyapunov functions: What exactly is their role in the approximate linear programming algorithm? And how do we identify Lyapunov functions in practical problems?

Some interesting answers lie in the connection between stochastic-shortest-path and infinite-horizon, discounted-cost problems.

In the stochastic-shortest-path problem, we consider an MDP with state space $\mathcal{S} \cup \{x^a\}$ ('a' stands for Absorbing state). We are interested in finding a policy u mapping states $x \in \mathcal{S}$ to actions $a \in \mathcal{A}_x$ that minimizes the sum of costs until state x^a is reached:

$$\mathrm{E}\left[\sum_{t=0}^{\tau^a - 1} g_u(x_t)\right],$$

where $\tau^a = \inf\{t : x_t = x^a\}$.

Denote by $J^p(x)$ the minimum expected total cost until reaching state x^a conditioned on the initial state being x ('p' stands for stochastic shortest Path) . We refer to J^p as the optimal value function for the stochastic-shortest-path problem. Knowledge of J^p allows one to derive an optimal policy.

By comparison with the infinite-horizon, discounted-cost problem, we expect that

J^p should satisfy the equation

$$J(x) = (T^p J)(x) = \min_{a \in \mathcal{A}_x} \left\{ g_{a(x)} + \sum_{y \in \mathcal{S}} P_a(x,y) J(y) \right\}, \quad \forall x \in \mathcal{S}. \tag{4.11}$$

This is Bellman's equation for the stochastic-shortest-path problem, and T^p the associated dynamic programming operator. Under certain conditions, T^p has a unique fixed point, corresponding to J^p.

A sufficient condition for J^p to be the unique fixed point of T^p is that all policies are *proper*; i.e., for any initial state x and under all policies u, state x^a is reachable. Recall that P_u is the matrix corresponding to transition probabilities under policy u; powers of this matrix P_u^k correspond to transition probabilities over multiple time steps ($P_u^k(x,y) = Prob(x_k = y|x_0 = x)$).

Definition 4.2 (Proper Policy) A policy u is *proper* if for every $x \in \mathcal{S}$ there is k such that $P_u^k(x, x^a) > 0$.

The connection between infinite-horizon discounted costs and stochastic shortest paths is well known. We can recast infinite-horizon, discounted-cost problems as stochastic-shortest-path problems by introducing an additional, absorbing state x^a to the state space and defining transition probabilities $\bar{P}$ so that, under all actions, the system transitions to x^a with probability $1-\alpha$, and to other states with the original probabilities rescaled by α:

$$\bar{P}_a(x,y) = \alpha P_a(x,y), \quad \forall x, y \in \mathcal{S} \tag{4.12}$$

$$\bar{P}_a(x,x^a) = 1-\alpha, \quad \forall x \in \mathcal{S}. \tag{4.13}$$

The graph in Figure 4.2(a) illustrates the transformation. Nodes represent states and directed edges represent state transitions that are possible under some policy.

Note that the operator T^p for the stochastic-shortest-path problem representing a discounted-cost problem coincides with the original operator T:

$$T^p J = \min_u \left\{ g_u + \bar{P}_u J \right\} = \min_u \left\{ g_u + \alpha P_u J \right\} = TJ.$$

All policies in stochastic-shortest-problems originated by infinite-horizon, discounted-cost problems are proper, as evidenced by equation (4.13).

We now illustrate the role of Lyapunov functions in approximate linear programming. We first note that our definition of a Lyapunov function for discounted-cost problems leads to a different definition of a Lyapunov function for the associated stochastic-shortest-path problem. In the original discounted-cost problem, there must exist $V : \mathcal{S} \mapsto \Re^+$ and $\beta < 1$ such that

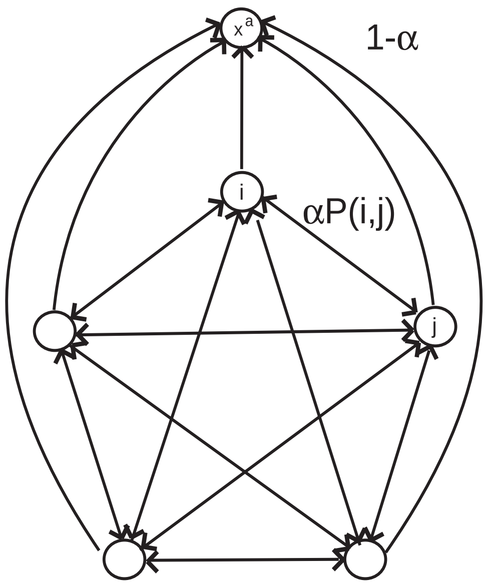
$$\alpha \max_a \sum_{y \in \mathcal{S}} P_a(x,y)V(y) \leq \beta V(x), \forall x \in \mathcal{S}. \tag{4.14}$$

In the stochastic-shortest-path problem, we extend V to $\mathcal{S} \cup x^a$ and let $V(x^a) = 0$. In the literature, Lyapunov inequalities typically involve a nonnegative function decreasing in some sense over time, with no regard to discount factors. A Lyapunov condition of this type in the stochastic-shortest-path problem is

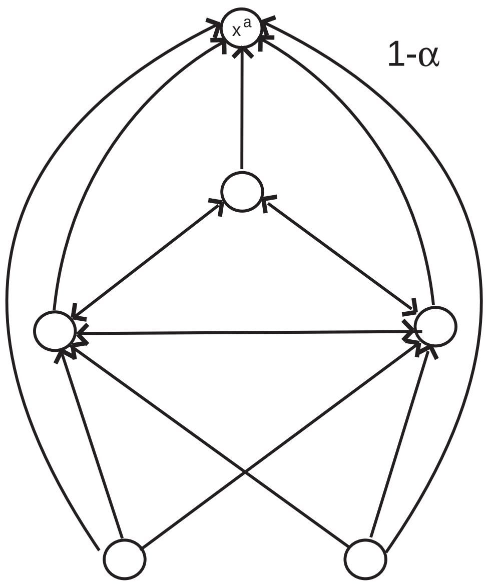
$$\max_a \sum_{y \in \mathcal{S}} \bar{P}_a(x,y)V(y) \leq \beta V(x), \forall x \in \mathcal{S}, \tag{4.15}$$

exactly the same as (4.14).

For stochastic-shortest-path problems representing discounted-cost problems, an obvious choice of Lyapunov function is $V(x) = 1$ for all $x \neq x^a$, and $V(x^a) = 0$. A consequence of this choice is that transition trends are disconsidered; for instance, one cannot distinguish between the situations in Figures 4.2(a) and 4.2(b). Compare these situations. They represent state transition structures for two different MDP's. In particular, the system represented in Figure 4.2(b) exhibits what might be considered a special structure: the two lower states are visited at most once. They are irrelevant in the decision-making process. Being able to distinguish the situation in Figure 4.2(b) from the situation in Figure 4.2(a) is highly desirable in approximate dynamic programming algorithms. With limited approximation power, it is important to take problem structure into account. This is illustrated in Figure 4.2(b), where exploiting the structure in the problem would allow us to ignore the two lower states, directing all power of the approximation architecture towards the other states in the system.



(a) stochastic-shortest-path version of infinite-horizon, discounted-cost problem.



(b) MDP exhibiting special structure.

Using the Lyapunov function as suggested by Theorem 4.2 does exactly that. Consider once again the problem in Figure 4.2(b). Since there are no transitions to the two lower states, they can be assigned arbitrarily large values in the Lyapunov function, and approximation errors in those state are highly discounted. Approximation errors in those states are irrelevant, and this is captured by Theorem 4.2.

We now turn to the question of how to generate Lyapunov functions for practical problems. Although approximate linear programming does not require that we explicitly provide Lyapunov functions — the bound in Theorem 4.2 holds as long as *there is* a Lyapunov function in the span of the basis functions — knowing how to generate them may aid in the process of selection of basis functions.

The following result, standard in dynamic programming, is of particular relevance to our study of Lyapunov functions. It states that for stochastic-shortest-path problems in which that all policies are proper, the associated operator T^p is a weighted-maximum-norm contraction.

Theorem 4.3. *Let T^p be the dynamic programming operator for a stochastic-shortest-path problem such that all policies are proper. Then there is $\xi : \mathcal{S} \mapsto \Re^+$ and $\beta < 1$ such that for all $J : \mathcal{S} \mapsto \Re$ and $\bar{J} : \mathcal{S} \mapsto \Re$,*

$$\|T^p J - T^p \bar{J}\|_{\infty,\xi} \leq \beta \|J - \bar{J}\|_{\infty,\xi}.$$

The next theorem represents a modified version of an argument commonly used in proving Theorem 4.3 [4]. It provides insight on the connection between Lyapunov functions and the maximum-norm-contraction property. It also provides ideas on how to generate Lyapunov functions.

Theorem 4.4. *Let T^p be the dynamic programming operator for a stochastic-shortest-path problem such that all policies are proper. Consider the auxiliary problem of maximizing the sum of costs $\bar{g}(x)$ until state x^* is reached, for an arbitrary positive cost function $\bar{g} : \mathcal{S} \mapsto \Re^+$. Denote the maximum expected sum of costs conditioned on the initial state being x by $J^m(x)$ ('m' stands for Maximum costs). Then:*

1. *J^m is a Lyapunov function, in the sense of inequality (4.15).*

2. *Theorem 4.3 holds with*
$$\xi(\cdot) = 1/J^m(\cdot) \tag{4.16}$$
and
$$\beta = \max_x \frac{J^m(x) - \bar{g}(x)}{J^m(x)}. \tag{4.17}$$

Proof: Since all policies are proper, J^m is well-defined over $\mathcal{S}$ and given by equation

$$J(x) = \bar{g}(x) + \max_{a \in \mathcal{A}_x} \sum_{y \in \mathcal{S}} P_a(x,y) J(y).$$

It follows that

$$\begin{aligned} \max_{a \in \mathcal{A}_x} \sum_{y \in \mathcal{S}} P_a(x,y) J^m(y) &= J^m(x) - \bar{g}(x) \\ &\leq \beta J^m(x). \end{aligned} \tag{4.18}$$

and the first claim follows. Recalling the definition of ξ (4.16), we have

$$\begin{aligned} (T^p J)(x) - (T^p \bar{J})(x) &\leq \max_{a \in \mathcal{A}_x} \sum_{y \in \mathcal{S}} P_a(x,y) |J(y) - \bar{J}(y)| \\ &= \max_{a \in \mathcal{A}_x} \sum_{y \in \mathcal{S}} P_a(x,y) J^m(y) \xi(y) |J(y) - \bar{J}(y)| \\ &\leq \|J - \bar{J}\|_{\infty,\xi} \max_{a \in \mathcal{A}_x} \sum_{y \in \mathcal{S}} P_a(x,y) J^m(y) \\ &\leq \|J - \bar{J}\|_{\infty,\xi} \beta J^m(x). \end{aligned}$$

Since J and $\bar{J}$ are arbitrary, we conclude that

$$\begin{aligned} |(T^p J)(x) - (T^p \bar{J})(x)| &\leq \|J - \bar{J}\|_{\infty,\xi} \beta J^m(x), \quad \forall x \\ \xi(x) |(T^p J)(x) - (T^p \bar{J})(x)| &\leq \|J - \bar{J}\|_{\infty,\xi} \beta, \quad \forall x \\ \|T^p J - T^p \bar{J}\|_{\infty,\xi} &\leq \beta \|J - \bar{J}\|_{\infty,\xi}, \end{aligned}$$

and the second claim follows. □

Theorem 4.4 suggests the following algorithm for automatically generating a Lyapunov function for infinite-horizon, discounted-cost problems:

1. choose a positive cost function $\bar{g} : \mathcal{S} \mapsto \Re^+$;
2. find the optimal value function J^m for the problem of *maximizing* the discounted-cost over infinite horizon. J^m is a Lyapunov function.

At first sight, the algorithm may appear ludicrous: trying to generate approximations for the optimal value function for a minimum discounted-cost problem because computing the exact function is infeasible, we require exact solution of a similar problem, with the same intractable dimensions! However, two observations lead us to believe that identifying Lyapunov functions may be an easier problem. First, the cost function $\bar{g}$ is to some extent arbitrary, and we may be able to choose it so that finding the associated optimal value function becomes easy. Second, and more importantly, it is easy to verify that scoring functions generated by approximate linear programming as approximations to the optimal value function J^m are also Lyapunov functions. In fact, a scoring function $\tilde{J}^m$ generated by approximate linear programming satisfies

$$\tilde{J}^m \geq T^m \tilde{J}^m = \max_u \{\bar{g} + \alpha P_u tilde J^m\} > \alpha H \tilde{J}^m.$$

Note that the inequality $\tilde{J}^m \geq T^m \tilde{J}^m$ is reversed relative to the customary inequality $T \leq TJ$ in the LP's involved in cost minimization; the difference is that here we are considering a problem of cost maximization instead.

It is important to note that using this algorithm does require some thought about the choice of $\bar{g}$, as it influences the scaling properties of the contraction factor β (4.17) and the quality of the Lyapunov function being generated. For instance, choosing $\bar{g}$ to be a constant function immediately yields a Lyapunov function that is a constant; however, that takes us back to the undesirable bound in Theorem 4.1.

It is not difficult to verify that Theorem 4.1 is a special case of Theorem 4.2; it corresponds to having a Lyapunov function that is constant over the state space. From Theorem 4.3, it is not difficult to show how to go in the opposite direction and generalize Theorem 4.1 to obtain Theorem 4.2. Recall that the operator T associated with discounted-cost problems coincides with the operator T^p associated with the corresponding stochastic-shortest-path problem. We conclude that T is a weighted-maximum-norm contraction. The proof of Theorem 4.1 revolves around the fact that T is a unweighted-maximum-norm contraction; making the appropriate changes to use the weighted-maximum-norm-contraction property instead, we obtain Theorem 4.2.

The unweighted-maximum-norm-contraction property of operator T is a key result in dynamic programming, leading to a number of other important results, such as existence and uniqueness of the optimal value function, and convergence of value iteration. It is therefore not surprising that, in the design and analysis of approximate dynamic programming algorithms, this property has often been a central focus. The discussion in this section demonstrates that using Lyapunov functions in the approximate linear programming algorithm corresponds to exploiting the weighted-maximum-norm-contraction property of operator T, a generalization of the unweighted-maximum-norm-contraction property, and exposes some of the limitations of the latter. We expect that this insight will transcend its usefulness in approximate linear programming and potentially guide the design and analysis of other approximate dynamic programming algorithms.

Chapter 5

Constraint Sampling

The number of variables involved in the approximate LP is relatively small. However, the number of constraints — one per state-action pair — can be intractable, and this presents an obstacle. In this chapter, we study a constraint sampling scheme that selects a tractable subset of constraints. We define an auxiliary LP with the same variables and objective as the approximate LP, but with only the sampled subset of constraints. More specifically, we aim at generating a good approximation to the solution of the approximate LP by solving the problem

$$\begin{aligned} \max_r \quad & c^T \Phi r \\ \text{s.t.} \quad & (T_a \Phi r)(x) \geq (\Phi r)(x) \ \forall (x,a) \in \mathcal{X}, \end{aligned} \tag{5.1}$$

for some set $\mathcal{X}$ of state-action pairs. We refer to problem (5.1) as the *reduced LP*.

The reduced LP is motivated by the fact that, because there is a relatively small number of variables in the approximate LP, many of the constraints should have a minor impact on the feasible region and do not need to be considered. In particular, we show that, by sampling a tractable number of constraints, we can guarantee that the solution of the auxiliary LP will be *near-feasible*, with high probability.

The fact that constraint sampling yields near-feasible solutions is a general property of LP's with a large number of constraints and relatively few variables. Exploiting properties specific to approximate linear programming, we will also establish that, under certain conditions, the error in approximating the optimal value function yielded by reduced LP should be close to that yielded by the approximate LP, with high probability.

The remainder of the chapter is organized as follows. In Section 5.1, we motivate the use of sampling in approximate linear programming, studying the possible consequences of discarding some of the constraints. In Section 5.2, we establish bounds on the number of constraints to be sampled so that the resulting reduced LP yields near-feasible solutions. We provide bounds that relate the approximation error yielded by the reduced LP to that yielded by the approximate LP in Section 5.3. Section 5.4 contains closing remarks.

In this chapter we consider certain families of MDP's and dynamic programming operators associated with them. We reserve the notation T for the dynamic programming operator associated with the standard MDP $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot,\cdot), g(\cdot), \alpha)$ considered in previous chapter.

5.1 Relaxing the Approximate LP Constraints

In this section, we offer an interpretation for the optimal solution of the reduced LP. The key observation is that its solution is also the solution to an approximate LP related to another MDP. We investigate the differences between this new MDP and the original one to understand the behavior of the reduced LP and gain insight into which constraints in the approximate LP are most likely to be important.

We begin by introducing the notion of *effective cost*.

Definition 5.1 For any $r \in \Re^k$, we define the effective costs associated with r by

$$g_a^r(x) = \max\left(g_a(x), (\Phi r)(x) - \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y) \right).$$

Based on this definition, it is easy to show that a vector r is always feasible for the approximate LP associated with the MDP $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot,\cdot), g_\cdot^r(\cdot), \alpha)$.

We use $(\cdot)^+$ and $(\cdot)^-$ to represent the positive and negative part functions:

$$(x)^+ = \max(x, 0) \text{ and } (x)^- = (-x)^+.$$

Lemma 5.1. *Let $r \in \Re^k$ be an arbitrary vector and $g^r_\cdot(\cdot)$ the associated effective cost. Then:*

1. *For any policy u, $g^r_u - g_u \leq (\Phi r - T\Phi r)^+$.*
2. *If $\tilde{T}$ is the dynamic programming operator for the MDP $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot,\cdot), g^r_\cdot(\cdot), \alpha)$, we have $\tilde{T}\Phi r \geq \Phi r$.*

Proof:

1. For any state-action pair (x, a), we have

$$\begin{aligned} g^r_a(x) - g_a(x) &= \max(g_a(x), (\Phi r)(x) - \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y)) - g_a(x) \\ &= \max((\Phi r)(x) - g_a(x) - \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y), 0) \\ &\leq \max((\Phi r)(x) - (T\Phi r)(x), 0). \end{aligned}$$

2. If a and x are such that

$$g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y) \geq (\Phi r)(x),$$

then we have $g^r_a(x) = g_a(x)$ and

$$g^r_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y) \geq (\Phi r)(x).$$

Otherwise, we have

$$\begin{aligned} g^r_a(x) &= (\Phi r)(x) - \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y), \\ g^r_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y) &= (\Phi r)(x). \end{aligned}$$

□

We can use Lemma 5.1 to interpret the solution of the reduced LP.

Lemma 5.2. *Let $\hat{r}$ be the optimal solution of the reduced LP. Define $\hat{T}$ to be the dynamic programming operator associated with the MDP given by $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot,\cdot), g_\cdot^{\hat{r}}(\cdot), \alpha)$. Then $\hat{r}$ is also the optimal solution of*

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{s.t.} & (\hat{T}\Phi r)(x) \geq (\Phi r)(x) \quad \forall x \in \mathcal{S}. \end{array} \tag{5.2}$$

Proof: From Lemma 5.1 (ii), the optimal solution $\hat{r}$ of the reduced LP is feasible for problem (5.2). Therefore, it suffices to show that the feasible region of (5.2) is contained in that of the reduced LP. Take any r that is feasible for (5.2). Then, for all $(x,a) \in \mathcal{X}$,

$$\begin{aligned} (\Phi r)(x) &\leq g_a^{\hat{r}}(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y) \\ &= g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x,y)(\Phi r)(y), \end{aligned}$$

and r is feasible for the reduced LP. The equality holds because $g_a(x) = g_a^{\hat{r}}(x)$ for all $(x,a) \in \mathcal{X}$. We conclude that the feasible region of the reduced LP contains that of (5.2), completing the proof. □

Lemma 5.2 establishes that we can view the reduced LP as effectively trying to approximate the optimal value function of an MDP with potentially larger costs than those in the original MDP. We can expect it to be a reasonable approximation to the approximate LP as long as the optimal value function associated with the MDP $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot, g_\cdot^{\hat{r}}(\cdot), \cdot, \alpha)$ is close to J^*. The next lemma establishes a bound on the difference between optimal value functions for problems that are equal except for their costs.

Lemma 5.3. *Let $\tilde{T}$ be the dynamic programming operator associated with an MDP $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot,\cdot), \tilde{g}_\cdot(\cdot), \alpha)$, where $\tilde{g}_u \geq g_u$ for all u. Let J^* and $\tilde{J}$ be the fixed points of T and $\tilde{T}$, respectively. Then if u^* is any optimal policy associated with the MDP related* *to T, we have*

$$\tilde{J}(x) - J^*(x) \leq \sum_y d(x,y)(\tilde{g}_{u^*}(y) - g_{u^*}(y)), \tag{5.3}$$

where

$$d(x,y) = \sum_{t=0}^{\infty} \alpha^t P_{u^*}^t(x,y). \tag{5.4}$$

Furthermore, if π is a stationary state distribution associated with u^* ($\pi = \pi P_{u^*}$), we have

$$\|\tilde{J} - J^*\|_{1,\pi} \leq \frac{1}{1-\alpha}\|\tilde{g}_{u^*} - g_{u^*}\|_{1,\pi}. \tag{5.5}$$

Proof: Let $\tilde{u}$ be an optimal policy with respect to $\tilde{T}$. Then

$$\begin{aligned}
\tilde{J}(x) - J^*(x) &= \sum_{t=0}^{\infty}\alpha^t \sum_{y\in\mathcal{S}} P_{\tilde{u}}^t(x,y)\tilde{g}_{\tilde{u}}(y) - \sum_{t=0}^{\infty}\alpha^t \sum_{y\in\mathcal{S}} P_{u^*}^t(x,y) g_{u^*}(y) \\
&\leq \sum_{t=0}^{\infty}\alpha^t \sum_{y\in\mathcal{S}} P_{u^*}^t(x,y)\tilde{g}_{u^*}(y) - \sum_{t=0}^{\infty}\alpha^t \sum_{y\in\mathcal{S}} P_{u^*}^t(x,y) g_{u^*}(y) \\
&= \sum_{y\in\mathcal{S}} (\tilde{g}_{u^*}(y) - g_{u^*}(y)) \sum_{t=0}^{\infty} \alpha^t P_{u^*}^n(x,y),
\end{aligned}$$

and (5.3) follows. Note that the inequality holds because

$$\tilde{J}_{u^*} = \sum_{t=0}^{\infty}\alpha^t P_{u^*}^t \tilde{g}_{u^*} \geq \tilde{J}_{\tilde{u}} = \sum_{t=0}^{\infty}\alpha^t P_{\tilde{u}}^t \tilde{g}_{\tilde{u}},$$

since $\tilde{u}$ is optimal for the dynamic program with costs $\tilde{g}$.

From (5.3) we get

$$\begin{aligned}
\|\tilde{J} - J^*\|_{1,\pi} &= \mathrm{E}_\pi|\tilde{J}(X_0) - J^*(X_0)| \\
&\leq \mathrm{E}_\pi\left[\sum_{t=0}^{\infty}\alpha^t P_{u^*}^t(\tilde{g}_{u^*} - g_{u^*})\right] \\
&= \sum_{t=0}^{\infty}\alpha^t \mathrm{E}_\pi\left[\tilde{g}_{u^*}(X_t) - g_{u^*}(X_t)\right] \\
&= \sum_{t=0}^{\infty}\alpha^t \mathrm{E}_\pi\left[\tilde{g}_{u^*}(X_0) - g_{u^*}(X_0)\right] \\
&= \frac{1}{1-\alpha}\|\tilde{g}_{u^*} - g_{u^*}\|_{1,\pi},
\end{aligned}$$

where the second equality holds by the nonnegativity of $\tilde{g}_{u^*} - g_{u^*}$ and the monotone convergence theorem and the third equality holds because π is a stationary distribution associated with P_{u^*}. □

As discussed in Chapters 3 and 4, it seems sensible to emphasize approximation errors in different states according to how often they are visited. The rationale is that non-optimal actions should typically increase costs more significantly if they are taken in frequently visited states than in rarely visited ones. Note that the bound (5.5) involves a norm that weights states according to their relative frequencies under an optimal policy, therefore it seems to be a reasonable measure for the error incurred by using the reduced LP instead of the approximate LP.

Together, Lemmas 5.2 and 5.3 provide some intuition for choices of constraints to be included in the reduced LP. From Lemma 5.2, we see that the reduced LP can be viewed as the approximate LP for a different problem, with potentially larger costs in some of the non-sampled states. We expect that, the closer the optimal value functions associated with the new and the original problems are, the more the reduced LP's behavior will resemble that of the approximate LP. Therefore, a sensible approach might be to design the constraint sampling scheme so as to minimize the error bounds presented in Lemma 5.3. These bounds suggest that some constraints are more important than others:

- The error $g_a^r(x) - g_a(x)$ is equal to zero for state-action pairs (x, a) such that $g_a(x) + \alpha \sum_{y \in \mathcal{S}} P_a(x, y)(\Phi r)(y) \geq (\Phi r)(x)$; in particular, costs remain the same for state-action pairs whose constraints are included in the reduced LP. Therefore the expected difference between the optimal value functions (5.5), which is shown to be proportional to the expected difference between the one-stage costs, may be reduced by choosing $\mathcal{X}$ to include states x with higher stationary probability.

- The componentwise bound (5.3) establishes that the difference between effective costs and original costs in a state y does not affect all states equally; its impact on each state x depends on the metric $d(x, y)$. If we want to minimize errors in the region of the state space that is visited more frequently, it seems reasonable to sample states that are "far" from this region with smaller probability. This translates into a sampling scheme that emphasizes frequently visited states. Indeed, the metric $d(x, y)$ can be interpreted as the average number of visits to state y given that the initial state is x and the system operates under an optimal policy, and therefore if a state is visited frequently, we expect that states that are "close" to it should also be visited frequently, and states that are "far" from it should probably be visited rarely.

We have identified an important property of the approximate LP, namely, that not all constraints are equally relevant, and their relative relevance depends on how often their corresponding states are visited. In the next section, we will show how to draw samples from a set of linear constraints that have an associated measure of relative importance so as to ensure with high confidence that any vector that satisfies the sampled constraints is *near-feasible* — it violates at most a set of constraints that has small measure. The results of Section 5.2 will then be applied in Section 5.3 to establish a bound on the error associated with the reduced LP, relative to the approximate LP, when constraints are randomly sampled.

5.2 High-Confidence Sampling for Linear Constraints

We consider a set of linear constraints

$$A_z r + b_z \geq 0, \quad \forall z \in \mathcal{Z} \tag{5.6}$$

where $r \in \Re^k$ and $\mathcal{Z}$ is a generic set of constraint indices. To keep the exposition simple, we make the following assumption on $\mathcal{Z}$:

Assumption 5.1. *The set of constraint indices $\mathcal{Z}$ is finite or countable.*

This assumption is sufficient for our purposes, since we consider MDP's with finite state and action spaces.

We use $Ar + b$ as shorthand notation for the function $f_r(z) = A_z r + b_z$.

Motivated by the approximate LP setting, we assume that $k \ll |\mathcal{Z}|$. Hence we have relatively few variables and a possibly huge number of constraints. In such a situation, we expect that many of the constraints will be irrelevant, either because they are always inactive or because they have a minor impact on the feasible region. One might then speculate that the feasible region specified by all constraints can be well-approximated by a sampled subset of these constraints. In the sequel, we will show that this is indeed the case, at least with respect to a certain criterion for a "good" approximation. We will also show that the number of constraints necessary to guarantee a good approximation does not depend on the total number of constraints, but rather on the number of variables.

We start by specifying how constraints are sampled. We assume that there is a probability measure ψ on $\mathcal{Z}$. The distribution ψ will have a dual role in our approximation scheme: on one hand, constraints will be sampled according to ψ; on the other hand, the same distribution will be involved in the criterion for assessing the quality of a particular set of sampled constraints. We have seen that in the approximate linear programming method certain constraints are more likely to have a significant impact on the final approximation we are trying to generate than others. In that context, ψ would be chosen to capture this fact, so that more important constraints are selected with higher probability.

We now discuss what it means for a certain set of sampled constraints to be good. Note that in general we cannot guarantee that all constraints will be satisfied over the feasible region of any subset of constraints. In Figure 5.1, for instance, we exemplify a worst-case scenario in which it is necessary to include all constraints in order to ensure that all of them are satisfied. Note however that the impact of any one of them on the feasible region is minor and might be considered negligible. In this spirit, we consider a subset of constraints good if we can guarantee that, by satisfying this subset, the set of constraints that are not satisfied has small measure. In other words, given a tolerance parameter $0 < \epsilon < 1$, we want to have $\mathcal{W} \subset \mathcal{Z}$ satisfying

$$\sup_{r: A_z r + b_z \geq 0,\ \forall z \in \mathcal{W}} \psi\left(\{y : A_y r + b_y < 0\}\right) \leq \epsilon. \tag{5.7}$$

Defining a set of constraint indices $\mathcal{W}$ to be good if (5.7) holds, we might ask how many (possibly repeated) constraints have to be sampled in order to ensure that the

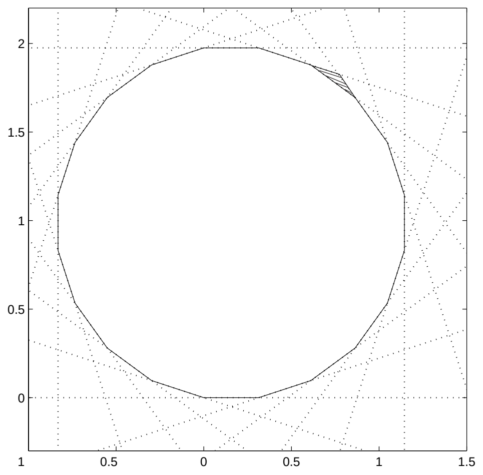


Figure 5.1: Large number of constraints in a low-dimensional feasible space. No constraint can be removed without affecting the feasible region. Shaded area demonstrates the impact of not satisfying one constraint on the feasible region.

resulting subset $\mathcal{W}$ is good with high probability. The next theorem establishes a bound on the number m of sampled constraints necessary to ensure that the set $\mathcal{W}$ formed by them is good with a prespecified probability $1 - \delta$.

Theorem 5.1. *Let $0 < \delta < 1$, $0 < \epsilon < 1$ be given constants and let $\mathcal{W}$ be the set formed by m i.i.d. random variables on $\mathcal{Z}$ with distribution ψ. Then if*

$$m \geq \frac{32}{\epsilon^2}\left[\ln 8 + \ln\frac{1}{\delta} + k\left(\ln\frac{16e}{\epsilon} + \ln\ln\frac{16e}{\epsilon}\right)\right],$$

we have

$$\sup_{r: A_z r + b_z \geq 0,\ \forall z \in \mathcal{W}} \psi\left(\{y : A_y r + b_y < 0\}\right) \leq \epsilon, \tag{5.8}$$

with probability greater than or equal to $1 - \delta$.

The proof of Theorem 5.1 relies on the theory of uniform convergence of empirical probabilities (UCEP). We defer it to the next section, where we first describe some ideas pertaining to UCEP and reformulate our problem so that the theory can be applied.

5.2.1 Uniform Convergence of Empirical Probabilities

Consider a probability space $(\mathcal{Z}, \mathbf{Z}, \mu)$ and a class $\mathcal{C} = \{\mathcal{C}_i, i \in I\} \subset \mathbf{Z}$, where I is a given set of indices. One would like to estimate $\mu(\mathcal{C}_i)$ by drawing samples from $\mathcal{Z}$ and observing whether they belong to $\mathcal{C}_i$ or not. Let $\mathbf{z} \in \mathcal{Z}^m$ be a vector containing m samples. The fraction of samples belonging to a set $\mathcal{C}_i$ is called the *empirical probability* of $\mathcal{C}_i$ and denoted by $\hat{\mu}(\mathcal{C}_i, \mathbf{z})$. The class $\mathcal{C}$ will have the UCEP property if the empirical probabilities converge in probability to their true value, uniformly on I as the number of samples m increases. Results determining when a given class has this property and the associated convergence rate can be found in [20, 57].

Let us first recast our problem as one of UCEP. Consider the class

$$\mathcal{C} = \{\mathcal{C}_r : r \in \Re^k\}, \tag{5.9}$$

where

$$\mathcal{C}_r = \left\{z \in \mathcal{Z} : A_z r + b_z \geq 0\right\}.$$

Note that $\mathcal{C}_r$ is the set of indices of constraints that are satisfied by r. For any vector r, we can determine the empirical probability $\hat{\psi}$ of $\mathcal{C}_r$ by drawing a sequence of m samples of constraint indices collected in a vector $\mathbf{z} \in \mathcal{Z}^m$ according to ψ, and setting

$$\hat{\psi}(\mathcal{C}_r, \mathbf{z}) = \frac{1}{m} \sum_{i=1}^{m} 1_{\mathcal{C}_r}(\mathbf{z}_i),$$

where 1_A is the indicator function of set A and $\mathbf{z}_i$ is the i^{th} component of $\mathbf{z}$. If $\mathcal{C}$ has the UCEP property, there is a minimum number of samples m_0 that guarantees, with high probability, that the true measure of any set in $\mathcal{C}$ is close to the empirical measure.

To see how UCEP can be applied to the constraint sampling problem, note that, for any vector r that is feasible for the sampled constraints, we have $\hat{\psi}(\mathcal{C}_r) = 1$. Assuming that we have chosen the number of samples to ensure that $\hat{\psi}$ is close to ψ, we conclude that for all such vectors, $\psi(\mathcal{C}_r)$ is close to 1 with high probability, hence r is near-feasible.

The weak law of large numbers guarantees that, under certain conditions, the empirical probabilities will converge in probability to their true values for each r. The questions to be asked are: Does $\hat{\psi}_r$ converge uniformly on r? What is the convergence rate? The answers are closely related to a certain measure of the complexity of classes of sets — the VC dimension [20, 57].

Definition 5.2 (VC dimension) Let $(\mathcal{Z}, \mathbf{Z})$ be a measurable space. Let $\mathcal{A} \subset \mathbf{Z}$ be a class of sets. The VC dimension of $\mathcal{A}$, denoted $\dim(\mathcal{A})$, is the maximum integer n such that there is a set $\mathcal{P} \subset \mathcal{A}$ of cardinality n that is *shattered* by $\mathcal{A}$, that is, such that $|\{\mathcal{B} \cap \mathcal{P} : \mathcal{B} \in \mathcal{A}\}| = 2^n$.

The next two lemmas establish that for the class $\mathcal{C}$ defined in (5.9), we have $\dim(\mathcal{C}) \leq k$.

Lemma 5.4. *[20] Let F be an m-dimensional vector space of real functions on a set $\mathcal{Z}$ and g be any function on $\mathcal{Z}$. Let $\mathcal{C} = \{\mathcal{C}_f : f \in F\}$ be the class of sets of the form $\mathcal{C}_f = \{z \in \mathcal{Z} : f(z) + g(z) \geq 0\}$. Then* $\dim(\mathcal{C}) = m$.

Lemma 5.5. *Let $\mathcal{C}$ be as defined in (5.9). Then* $\dim(\mathcal{C}) \leq k$.

Proof: It is clear that the set $\{f : f(z) = A_z r, r \in \Re^k\}$ forms a real vector space of dimension at most k. Hence, by Lemma 5.5, $\dim(\mathcal{C}) \leq k$. $\square$

The following lemma, adapted from [20, 57], establishes the rate of convergence of empirical probabilities associated with a class of sets as a function of its VC dimension.

Lemma 5.6. *Let $(\mathcal{Z}, \boldsymbol{Z}, \mu)$ be a probability space. Let $\mathcal{A} \subset \boldsymbol{Z}$ be a class with finite VC-dimension. For any vector $\mathbf{z} \in \mathcal{Z}^m$ and any set $\mathcal{B} \in \mathcal{A}$, define*

$$\hat{\mu}(\mathcal{B}, \mathbf{z}) = \frac{1}{m} \sum_{i=1}^{M} 1_{\mathcal{B}}(\mathbf{z}_i).$$

Then

$$\mu^m \left(\left\{ \mathbf{z} \in \mathcal{Z}^m : \sup_{\mathcal{B} \in \mathcal{A}} |\mu(\mathcal{B}) - \hat{\mu}(\mathcal{B}, \mathbf{z})| > \epsilon \right\} \right) \leq 8 \left(\frac{16e}{\epsilon} \ln \frac{16e}{\epsilon} \right)^{\dim(\mathcal{A})} e^{-\frac{m\epsilon^2}{32}},$$

where μ^m denotes the joint distribution for m i.i.d. random variables distributed according to μ.

We are now poised to prove Theorem 5.1.

Proof of Theorem 5.1

Let $\mathcal{C}$ be as in (5.9). For any $\mathbf{z} \in \mathcal{Z}^m$ and any $\mathcal{B} \in \mathcal{C}$, define

$$\hat{\psi}(\mathcal{B}, \mathbf{z}) = \frac{1}{m} \sum_{i=1}^{m} 1_{\mathcal{B}}(\mathbf{z}_i).$$

Then from Lemmas 5.5 and 5.6, we have

$$\begin{aligned} \psi^m\left(\left\{\mathbf{z} \in \mathcal{Z}^m : \sup_{r \in \Re^d} \left|\psi(\mathcal{C}_r) - \hat{\psi}(\mathcal{C}_r, \mathbf{z})\right| > \epsilon\right\}\right) &\leq 8\left(\frac{16e}{\epsilon} \ln \frac{16e}{\epsilon}\right)^{\dim(\mathcal{C})} e^{-\frac{m\epsilon^2}{32}} \\ &\leq 8\left(\frac{16e}{\epsilon} \ln \frac{16e}{\epsilon}\right)^{k} e^{-\frac{m\epsilon^2}{32}}, \end{aligned}$$

where ψ^m denotes the joint distribution for m i.i.d. random variables distributed according to ψ. Let $\mathcal{W} = \{\mathbf{z}_1, ..., \mathbf{z}_m\}$ and $\mathcal{F}_{\mathcal{W}} = \{r : A_z + b_z \geq 0, \forall z \in \mathcal{W}\}$. We have

$$\begin{aligned} 8\left(\frac{16e}{\epsilon} \ln \frac{16e}{\epsilon}\right)^{k} e^{-\frac{m\epsilon^2}{32}} &\geq \psi^m\left(\left\{\mathbf{z} \in \mathcal{Z}^m : \sup_{r \in \mathcal{F}_{\mathcal{W}}} \left|\psi(\mathcal{C}_r) - \hat{\psi}(\mathcal{C}_r, \mathbf{z})\right| > \epsilon\right\}\right) \\ &= \psi^m\left(\left\{\mathbf{z} \in \mathcal{Z}^m : \sup_{r \in \mathcal{F}_{\mathcal{W}}} \left|\psi(\mathcal{C}_r) - 1\right| > \epsilon\right\}\right) \\ &= \psi^m\left(\left\{\mathbf{z} \in \mathcal{Z}^m : \sup_{r \in \mathcal{F}_{\mathcal{W}}} 1 - \psi(\mathcal{C}_r) > \epsilon\right\}\right) \\ &= \psi^m\left(\left\{\mathbf{z} \in \mathcal{Z}^m : \sup_{r \in \mathcal{F}_{\mathcal{W}}} \psi(\overline{\mathcal{C}_r}) > \epsilon\right\}\right) \\ &= \psi^m\left(\left\{\mathbf{z} \in \mathcal{Z}^m : \sup_{r \in \mathcal{F}_{\mathcal{W}}} \psi(\{y \in \mathcal{Z} : A_y r + b_y < 0\}) > \epsilon\right\}\right), \end{aligned}$$

Hence by setting

$$m \geq \frac{32}{\epsilon^2}\left[\ln 8 + \ln \frac{1}{\delta} + k\left(\ln \frac{16e}{\epsilon} + \ln \ln \frac{16e}{\epsilon}\right)\right],$$

we conclude after some simple algebra that the bound (5.8) on the measure of the set of the violated constraints holds with probability greater than or equal to $1 - \delta$. $\square$

Theorem 5.1 establishes what may be perceived as a puzzling result: as the number of constraints grows, the number of sampled constraints necessary for a good

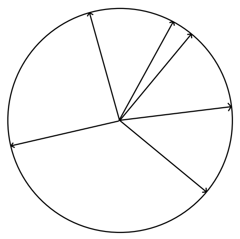


Figure 5.2: Constraints in the approximate LP can be viewed as vectors in a low-dimensional unit sphere.

approximation of a set of constraints indexed by $z \in \mathcal{Z}$ eventually depends only on the number of variables involved in these constraints and not on the set $\mathcal{Z}$. A geometric interpretation of the result helps to dissolve the apparent contradiction. Note that the constraints are fully characterized by vectors $[A_z \; b_z]$ of dimension equal to the number of variables plus one. Since near-feasibility involves only consideration of whether constraints are violated, and not by how much they are violated, we may assume without loss of generality that $\|[A_z \; b_z]\| = 1$, for an arbitrary norm. Hence constraints can be thought of as vectors in a low-dimensional unit sphere. After a large number of constraints is sampled, we are likely to have a *cover* for the original set of constraints — i.e., any other constraint is close to one of the already sampled ones, so that the sampled constraints *cover* the set of constraints. The number of sampled constraints necessary in order to have a cover for the original set of constraints is bounded above by the number of vectors necessary to cover the unit sphere, which naturally depends only on the dimension of the sphere, or alternatively, on the number of variables involved in the constraints.

5.3 High-Confidence Sampling and ALP

In this section, we place a bound on the error introduced by constraint sampling in the approximation of the optimal value function. It is easy to see that near-feasibility is not enough to ensure small error if the solution of the reduced LP involves arbitrarily large violations of the non–sampled constraints. We will present conditions that enable a graceful error bound.

Our main result is based on the following variant of the reduced LP, which we refer to as the *bounded reduced LP*:

$$\begin{aligned} \max \quad & c^T \Phi r \\ \text{s.t.} \quad & (T_a \Phi r)(x) \geq (\Phi r)(x), \ \forall (x,a) \in \mathcal{X} \\ & r \in \mathcal{N}. \end{aligned} \tag{5.10}$$

The set $\mathcal{N}$ is viewed as another algorithm parameter to be chosen, like c and Φ. It must contain an optimal solution of the approximate LP and is used for controlling the magnitude of violations of nonsampled constraints. A more comprehensive explanation of the constraint $r \in \mathcal{N}$ and appropriate selections for $\mathcal{N}$ will be discussed in Section 5.3.1.

Let us introduce certain constants and functions involved in our error bound. Recall the definition of $\mu_{u,\nu}$ (3.1). We define a family of probability distributions μ_u on the state space $\mathcal{S}$, given by

$$\mu_u^T \equiv \mu_{u,c}^T = (1-\alpha) c^T (I - \alpha P_u)^{-1}, \tag{5.11}$$

for each policy u. Note that

$$(I - \alpha P_u)^{-1} = \sum_{t=0}^{\infty} \alpha^t P_u^t,$$

and therefore, if c is a probability distribution, $\mu_u(x)/(1-\alpha)$ is the expected discounted number of visits to state x under policy u, if the initial state is distributed according to c. Furthermore, $\lim_{\alpha \uparrow 1} \mu_u(x)$ is the stationary distribution associated with policy u. We interpret μ_u as a measure of the relative importance of states under policy u.

We can prove the following lemma involving μ_u.

Lemma 5.7. *For all J and $p \geq 1$, we have*

$$\|\alpha P_u J\|_{p,\mu_u} \leq \|J\|_{p,\mu_u}.$$

Proof: We have

$$\begin{aligned}
\|\alpha P_u J\|_{p,\mu_u}^p &= \alpha^p \sum_x \mu_u(x) \left| \sum_{y \in \mathcal{S}} P_u(x,y) J(y) \right|^p \\
&\leq \alpha^p \sum_x \mu_u(x) \left(\sum_{y \in \mathcal{S}} P_u(x,y) |J(y)| \right)^p \\
&\leq \alpha^p \sum_x \mu_u(x) \sum_{y \in \mathcal{S}} P_u(x,y) |J(y)|^p \\
&= \alpha^p \mu_u^T P_u |J|^p \\
&\leq \alpha(1-\alpha) c^T \sum_{t=0}^{\infty} \alpha^t P_u^t P_u |J|^p \\
&= (1-\alpha) c^T \sum_{t=1}^{\infty} \alpha^t P_u^t |J|^p \\
&\leq (1-\alpha) c^T \sum_{t=0}^{\infty} \alpha^t P_u^t |J|^p \\
&= \mu_u^T |J|^p \\
&= \|J\|_{p,\mu_u}^p.
\end{aligned}$$

The notation $|J|^p$ indicates componentwise absolute value and power of the vector J. We used Jensen's inequality on the first and second lines with the convex functions $|.|$ and x^p, which is convex on $\Re^+$ for all $p \geq 1$. The third inequality follows from $\alpha^p \leq \alpha$. □

Define for each policy u and for each p the constant

$$M_{u,p} = \max(\sup_{r \in \mathcal{N}} \|\Phi r\|_{p,\mu_u}, \|g_u\|_{p,\mu_u}) \tag{5.12}$$

and the distribution

$$\psi_u(x,a) = \frac{\mu_{u^*}(x)}{|\mathcal{A}_x|}, \ \forall a \in \mathcal{A}_x,$$

and denote by A the maximum number of actions available for any given state:

$$A = \max_x |\mathcal{A}_x|.$$

We now present the main result of the chapter — a bound on the error introduced by constraint sampling in the approximation of the optimal value function.

Theorem 5.2. *Let $\mathcal{N} \subset \Re^k$ be a set containing an optimal solution of the approximate LP. Let u^* be an optimal policy and $\mathcal{X}$ be a (random) set of m state-action pairs sampled independently according to the distribution $\psi_{u^*}(x,a)$, where*

$$m \geq \frac{32A^2}{\left(\frac{(1-\alpha)\epsilon}{6}\right)^{\frac{2p}{p-1}}} \left[\ln 8 + \ln\frac{1}{\delta} + k\left(\ln\frac{16Ae}{\left(\frac{(1-\alpha)\epsilon}{6}\right)^{\frac{p}{p-1}}} + \ln\ln\frac{16Ae}{\left(\frac{(1-\alpha)\epsilon}{6}\right)^{\frac{p}{p-1}}}\right)\right], \tag{5.13}$$

for a given scalar $p > 1$. Let $\tilde{r}$ be a solution to the approximate LP and $\hat{r}$ be a solution to the bounded reduced LP. Then, with probability at least $1-\delta$, we have

$$\|J^* - \Phi\hat{r}\|_{1,c} \leq \|J^* - \Phi\tilde{r}\|_{1,c} + M_{u^*,p}\epsilon. \tag{5.14}$$

Proof: From Theorem 5.1, we have, with probability no less than $1-\delta$,

$$\begin{aligned}
\frac{\left(\frac{(1-\alpha)\epsilon}{6}\right)^{\frac{p}{p-1}}}{A} &\geq \psi_{u^*}\left(\{(x,a) : (T_a\Phi\hat{r})(x) < (\Phi\hat{r})(x)\}\right) \\
&= \sum_x \mu_{u^*}(x)\frac{1}{|\mathcal{A}_x|}\sum_a 1_{(T_a\Phi\hat{r})(x)<(\Phi\hat{r})(x)} \\
&\geq \frac{1}{A}\sum_x \mu_{u^*}(x) 1_{(T_{u*}\Phi\hat{r})(x)<(\Phi\hat{r})(x)},
\end{aligned} \tag{5.15}$$

therefore for all $q \geq 1$ we have

$$\left\|1_{(T_{u*}\Phi\hat{r})(\cdot)-(\Phi\hat{r})(\cdot)<0}\right\|_{q,\mu_{u^*}}^q \leq \left(\frac{(1-\alpha)\epsilon}{6}\right)^{\frac{p}{p-1}}. \tag{5.16}$$

We also have

$$\begin{aligned}
\|J^* - \Phi\hat{r}\|_{1,c} &= c^T \left|(I-\alpha P_{u^*})^{-1}(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})\right| \\
&\leq c^T (I-\alpha P_{u^*})^{-1} \left|g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r}\right| \\
&= c^T (I-\alpha P_{u^*})^{-1} \left[(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})^+ + (g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})^-\right] \\
&= c^T (I-\alpha P_{u^*})^{-1} \Big[(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})^+ - (g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})^- + \\
&\qquad +2\left(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r}\right)^-\Big] \\
&= c^T (I-\alpha P_{u^*})^{-1} \left[g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r} + 2\left(T_{u^*}\Phi\hat{r} - \Phi\hat{r}\right)^-\right] \\
&= c^T (J^* - \Phi\hat{r}) + 2c^T (I-\alpha P_{u^*})^{-1}\left(T_{u^*}\Phi\hat{r} - \Phi\hat{r}\right)^- . \qquad (5.17)
\end{aligned}$$

The inequality comes from the fact that $c > 0$ and

$$(I-\alpha P_{u^*})^{-1} = \sum_{n=0}^{\infty} \alpha^n P_{u^*}^n \geq 0,$$

where the inequality is componentwise, so that

$$\begin{aligned}
\left|(I-\alpha P_{u^*})^{-1}(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})\right| &\leq \left|(I-\alpha P_{u^*})^{-1}\right| \left|(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})\right| \\
&= (I-\alpha P_{u^*})^{-1} \left|(g_{u^*} - (I-\alpha P_{u^*})\Phi\hat{r})\right| .
\end{aligned}$$

We assume without loss of generality that $\tilde{r} \in \mathcal{N}$, since by assumption there is at least one optimal solution of the ALP that satisfies this constraint[1]. Therefore $\tilde{r}$ is feasible for the bounded reduced LP. Since $\hat{r}$ is the optimal solution of the same problem, we have $c^T\Phi\hat{r} \geq c^T\Phi\tilde{r}$ and

$$\begin{aligned}
c^T(J^* - \Phi\hat{r}) &\leq c^T(J^* - \Phi\tilde{r}) \\
&= \|J^* - \Phi\tilde{r}\|_{1,c}, \qquad (5.18)
\end{aligned}$$

hence we just need to show that the second term in (5.17) is small to guarantee that problem (5.10) does not perform much worse than the approximate LP.

[1]Note that all optimal solutions of the approximate LP yield the same approximation error $\|J^* - \Phi r\|_{1,c}$, hence the error bound (5.14) is independent of the choice of $\tilde{r}$.

Now

$$
\begin{aligned}
2c^T(I-\alpha P_{u^*})^{-1}\left(T_{u^*}\Phi\hat{r}-\Phi\hat{r}\right)^- &= \qquad (5.19)\\
&= \frac{2}{1-\alpha}\mu_{u^*}^T\left(T_{u^*}\Phi\hat{r}-\Phi\hat{r}\right)^-\\
&= \frac{2}{1-\alpha}\left\langle T_{u^*}\Phi\hat{r}-\Phi\hat{r}, 1_{(T_{u^*}\Phi\hat{r})(\cdot)-(\Phi\hat{r})(\cdot)<0}\right\rangle_{\mu_{u^*}}\\
&\leq \frac{2}{1-\alpha}\|T_{u^*}\Phi\hat{r}-\Phi\hat{r}\|_{p,\mu_{u^*}}\|1_{(T_{u^*}\Phi\hat{r})(\cdot)-(\Phi\hat{r})(\cdot)<0}\|_{\frac{p}{p-1},\mu_{u^*}}, \qquad (5.20)
\end{aligned}
$$

where the last step is an application of Hölder's inequality.

Applying (5.16) with $q=\frac{p}{p-1}$ yields

$$\|1_{(T_{u^*}\Phi\hat{r})(\cdot)-(\Phi\hat{r})(\cdot)<0}\|_{\frac{p}{p-1},\mu_{u^*}} \leq \frac{(1-\alpha)\epsilon}{6}, \qquad (5.21)$$

with probability at least $1-\delta$.

Finally, we have

$$
\begin{aligned}
\|T_{u^*}\Phi\hat{r}-\Phi\hat{r}\|_{p,\mu_{u^*}} &\leq \|g_{u^*}\|_{p,\mu_{u^*}}+\|\Phi\hat{r}\|_{p,\mu_{u^*}}+\|\alpha P_{u^*}\Phi\hat{r}\|_{p,\mu_{u^*}}\\
&\leq \|g_{u^*}\|_{p,\mu_{u^*}}+2\|\Phi\hat{r}\|_{p,\mu_{u^*}} \qquad (5.22)\\
&\leq 3M_{u^*,p}. \qquad (5.23)
\end{aligned}
$$

The second inequality follows from Lemma 5.7, and the third one is follows from the definition of $M_{u^*,p}$.

The error bound (5.14) follows from (5.17), (5.18), (5.20), (5.21) and (5.23). □

If we think of the quantity $M_{u^*,p}$ as an indicator of the magnitude of the functions involved in the approximation scheme — the optimal value function and its approximation, the error bound (5.14) implies that we can make the approximation error yielded by the bounded reduced LP relatively close to that obtained by the approximate LP, with a tractable number of sampled constraints.

Some aspects of Theorem 5.2 deserve further discussion. The first of them is the constraint $r\in\mathcal{N}$, which differentiates the bounded reduced LP from the reduced LP. This constraint is discussed in Section 5.3.1. Another aspect relates to the sampling distribution ψ_{u^*}. It is clear that we will not be able to compute and use it in practice. We discuss in Section 5.3.2 how sampling with a different distribution affects our error bounds. Finally, some difficulties may arise in problems involving a large number of actions per state — our bounds require a number of samples on the order of $O(A^2 \ln A)$ on the maximum number of available actions per state A. We address how deal with large state spaces in Section 5.3.3.

5.3.1 Bounding Constraint Violations

The motivation for the bounded reduced LP is that constraint sampling alone only guarantees near-feasibility, without regard to the magnitude of the violations of constraints that are not sampled. It is reasonable to expect that, the larger these violations are, the worse the approximation yielded by the reduced LP relative to that of the approximate LP. Recall that the set $\mathcal{N}$, involved in the bounded reduced LP, affects the constant $M_{u^*,p}$ present in the error bound (5.14). The constant $M_{u^*,p}$, in turn, reflects how large vectors Φr involved in the approximation can be. The underlying idea is to choose $\mathcal{N}$ so as to keep approximations $\Phi \hat{r}$ produced by the bounded reduced LP relatively small, thereby preventing constraint violations from growing out of control.

Two important issues that affect the significance of the error bounds presented in Theorem 5.2 are the magnitude of $M_{u^*,p}$ and the feasibility of choosing an appropriate set $\mathcal{N}$ efficiently. We will begin by investigating $M_{u^*,p}$.

Recall that $M_{u^*,p}$ is defined as the maximum of $\sup_{r \in \mathcal{N}} \|\Phi r\|_{p,\mu_{u^*}}$ and $\|g_{u^*}\|_{p,\mu_{u^*}}$. In our analysis, we focus on the first term; the implicit assumption is that typically the magnitude of the optimal value function and its approximation is of the order of $1/(1-\alpha)$ of the magnitude of one-stage costs, so that $\sup_{r \in \mathcal{N}} \|\Phi r\|_{p,\mu_{u^*}}$ drives $M_{u^*,p}$.

The set $\mathcal{N}$ must contain an optimal solution of the approximate LP. Hence the magnitude of the functions involved in the bounded reduced LP is at least in part dictated by the behavior of the approximate LP. The next lemma provides a bound on a certain norm of optimal solutions to the approximate LP.

Lemma 5.8. *Let $\tilde{r}$ be an optimal solution of the approximate LP. Then*

$$\|\Phi\tilde{r}\|_{1,c} \leq 2\|J^*\|_{1,c}. \tag{5.24}$$

Proof: First, note that $J^* \geq \Phi\tilde{r}$ and $J^* \geq 0$, therefore

$$c^T(\Phi\tilde{r})^+ \leq c^T J^* = \|J^*\|_{1,c}. \tag{5.25}$$

Since we assume that all costs are nonnegative, $r = 0$ is always a feasible solution for the approximate LP, therefore

$$\begin{aligned} 0 &\leq c^T\Phi\tilde{r} \\ &\leq c^T(\Phi\tilde{r})^+ - c^T(\Phi\tilde{r})^-, \end{aligned}$$

and

$$c^T(\Phi\tilde{r})^- \leq c^T(\Phi\tilde{r})^+. \tag{5.26}$$

From (5.25) and (5.26):

$$\begin{aligned} \|\Phi\tilde{r}\|_{1,c} &= c^T(\Phi\tilde{r})^+ + c^T(\Phi\tilde{r})^- \\ &\leq 2c^T(\Phi\tilde{r})^+ \\ &\leq 2\|J^*\|_{1,c} \end{aligned}$$

and the claim follows. □

Lemma 5.8 establishes that solutions to the approximate LP are comparable in magnitude to the optimal value function J^* — approximate LP solutions have norm $\|\cdot\|_{1,c}$ at most twice as large as J^*. Ideally, one would choose $\mathcal{N}$ so as to reproduce that behavior in the bounded reduced LP; in other words, $\mathcal{N}$ would impose that the approximations involved in the bounded reduced LP are on the same order of magnitude as the optimal value function. Note that, if $c = \pi_u$, the stationary distribution associated with policy u, $\|J_u\|_{1,c}$ is equal to the average cost associated with policy u multiplied by $1/(1-\alpha)$. Therefore, in rough terms, $M_{u^*,p}$ is representative of the average cost of the system scaled by the factor $1/(1-\alpha)$, at least for relatively small values of p.

We now turn to the question of how to choose $\mathcal{N}$. Appropriate choices will depend on the problem at hand. Here we discuss three different approaches that are not completely developed, but may represent a starting point in setting up the bounded reduced LP for a practical application.

The simplest approach is to use the reduced LP in its original form, with constraints sampled randomly according to the distribution suggested in Theorem 5.2. The hope is that the constraints relative to a large number of (random) state-action pairs would eventually lead to a finite upper bound for any Φr satisfying them, as any feasible solution of the approximate LP is bounded above by J^*. That being the case, we may consider a bounded reduced LP containing the same constraints and $c^T \Phi r \geq 0$, which is always satisfied by any optimal solution of the approximate LP or of the reduced LP (recall that $r = 0$ is always feasible since costs are nonnegative). Vectors Φr that are bounded above and satisfy this constraint must be bounded. Therefore we may think of the sampled constraints and $c^T \Phi r \geq 0$ as a constraint of the form $r \in \mathcal{N}$, and Theorem 5.2 applies with a finite $M_{u^*,p}$. However, since the constraint $c^T \Phi r \geq 0$ is always satisfied by optimal solutions of the reduced LP, we conclude that the solutions of the reduced LP and of the bounded reduced LP under consideration coincide, and the bounds provided in Theorem 5.2 for the bounded reduced LP should hold for the reduced LP.

We expect the aforementioned approach to be effective in practice, but it certainly lacks theoretical guarantees; in other words, it does not allow for a priori bounds on the constant $M_{u^*,p}$. Having such bounds would make it possible to choose the tolerance level ϵ — equivalently, the number of sampled constraints — so as to achieve a prespecified error $M_{u^*,p}\epsilon$. In the absence of these bounds, one can at best solve the reduced LP with a given ϵ and then *estimate* the constraint sampling error based on the solution $\Phi \hat{r}$, rather than *control* it. An alternative approach is as follows. Along the same lines used to conclude that the constraints of the reduced LP should eventually constitute a constraint of the form $r \in \mathcal{N}$ with the desired properties, one could conjecture that, in some applications, it is possible to find a small set of state-action pairs such that the constraints related to that set are sufficient to make the solution of the reduced LP bounded. In this case, one would just have to include these constraints in the reduced LP along with the sampled ones — the set $\mathcal{X}$ of state-action pairs would be the union of a fixed, deterministic, relatively small set and a random set as prescribed by Theorem 5.2. This approach is illustrated in the following example.

Example 5.1

We consider a single queue with controlled service rate. The state x denotes the number of jobs currently in the queue, and actions a indicate the service rate (or departure probability per time stage) for jobs in the queue. Jobs arrive in any stage with probability $p < 1/2$, and up to one event can happen in any stage. The system administrator can set the service rate to p, $1/2$ or $1-p$, except when there are no jobs in the queue, in which case $a = 0$. Costs per stage are of the form

$$g_a(x) = x + m(a),$$

where m is a nonnegative cost associated with service effort with $m(0) = 0$.

We assume for the sake of simplicity that the queue has infinite buffer. This contradicts the hypothesis of finite state space, but the same conclusions should hold for the case of relatively large, finite buffer size.

We assume that the discount factor α is large enough to make the system "stable" under the optimal policy, where stability is taken in a loose sense indicating that $a^*(x) > p$ for all large enough x. Note that this implies that the optimal stationary state distribution has an upper bound with exponential decay, motivating the use of state-relevance weights c of the form

$$c(x) = (1-\rho)\rho^x$$

for some ρ strictly between 0 and 1.

We consider basis functions $\Phi_1(x) = x$, $\Phi_2(x) = 1$.

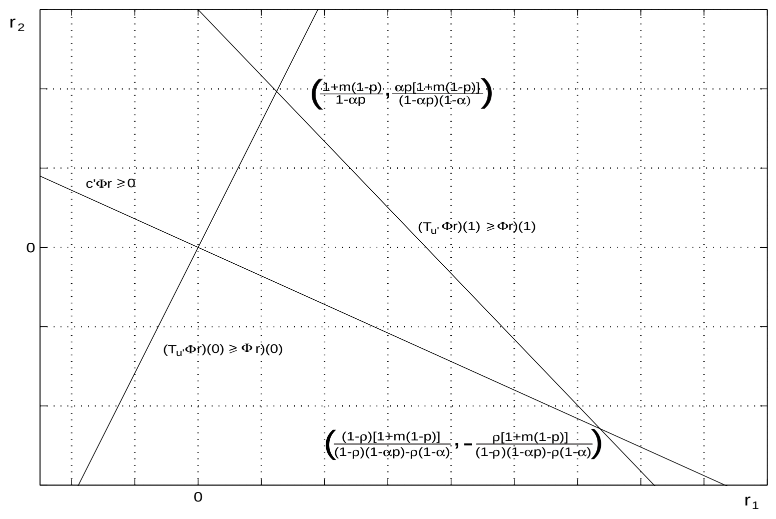


Figure 5.3: Typical feasible region for Example 5.1, constraints (5.27)-(5.29), with $0 < \rho < 1$, $\alpha \approx 1$, $p < 1/2$(graph constructed with $\rho = 0.9$, $\alpha = 0.99$, $p = 0.4$, $m(0) = 0$, $m(1-p) = 1$).

Now consider the constraints

$$-\alpha p r_1 + (1-\alpha) r_2 \leq 0 \tag{5.27}$$

$$(1-2\alpha p) r_1 + (1-\alpha) r_2 \leq 1 + m(1-p) \tag{5.28}$$

$$\rho r_1 + (1-\rho) r_2 \geq 0 \tag{5.29}$$

The first and second constraints correspond to state-action pairs $(0,0)$ and $(1, 1-p)$. The third one is equivalent to $c^T \Phi r \geq 0$. This constraint is automatically satisfied by any optimal solution of the reduced LP, since all costs are nonnegative, implying that $r = 0$ is always a feasible solution. The feasible region of (5.27)-(5.29) is shown in Figure 5.3. It is seen to be bounded with maximum values of $|r_i|$ of the order of $1/(1-\alpha)$. Therefore $\Phi\hat{r}$ must be bounded in norm for any reduced LP including (5.27) and (5.28), and one expects $M_{u^*,p}$ to be $O(1/(1-\alpha))$.

Finally, we mention a third possibility, which renders the bounded reduced LP a problem of convex optimization rather than an LP. This involves adding the constraint

$$\|\Phi r\|_{1,c} \leq 2\|J_u\|_{1,c}, \tag{5.30}$$

for an arbitrarily chosen policy u. This constraint implies constraint (5.24) on the norm of solutions of the approximate LP, therefore it is satisfied by optimal solutions of the approximate LP. It should make $M_{u^*,p}$ proportional to $\|J_u\|_{1,c}$, at least for small $p > 1$. A potential advantage of this approach is that in some cases it may be easier to estimate $\|J_u\|_{1,c}$ than to find a small set of constraints to ensure boundedness in the reduced LP. However, it requires devising a way of dealing with constraint (5.30) when solving the bounded reduced LP. This might not be an easy task, considering this constraint involves computation of L_1 norms (i.e., summations) in a high-dimensional space.

5.3.2 Choosing the Constraint Sampling Distribution

In Theorem 5.2, we assume that states are sampled according to distribution μ_{u^*} for some optimal policy u^*. We will generally not be able to compute μ_{u^*} in practice, as we do not know any optimal policy u^*; we have to sample constraints according to an alternative distribution $\bar{\mu}$. This affects the error bound (5.14) through the term

$$\|1_{(\Phi\hat{r})(\cdot)-(T_{u^*})(\Phi\hat{r})(\cdot)>0}\|_{\frac{p-1}{p},\mu_{u^*}}, \tag{5.31}$$

which can no longer be shown to be less than or equal to $\frac{(1-\alpha)\epsilon}{6}$. Instead, we have

$$\|1_{(\Phi\hat{r})(\cdot)-(T_{u^*})(\Phi\hat{r})(\cdot)>0}\|_{\frac{p-1}{p},\bar{\mu}} \leq \frac{(1-\alpha)\epsilon}{6}.$$

The problem of ensuring that (5.31) is small given that constraints are sampled according to $\bar{\mu}$ corresponds to a problem of UCEP in which the sampling distribution and the testing distributions differ. Intuitively, we expect that if $\bar{\mu}$ is reasonably close to μ_{u^*}, (5.31) should remain relatively small.

How to choose $\bar{\mu}$ is still an open question. As a simple heuristic, noting that

$\mu^*(x) \to c(x)$ as $\alpha \to 0$, one might choose $\bar{\mu} = c$. This is the approach taken in the case studies presented in Chapter 6. This choice is also justified by the realization that in many applications c would be the best estimate for π_{u*} that can be obtained, and if it were correct, we would have $\mu_{u^*} = c = \pi_{\mu^*}$.

5.3.3 Dealing with Large Action Spaces

Generating approximations to the optimal value function within an approximation architecture aims to alleviate the problems arising when one deals with large state spaces. If the action space is also large, additional difficulties may be encountered; in the specific case of approximate linear programming, the number of constraints involved in the reduced LP becomes intractable as the cardinality of the action sets $\mathcal{A}_x$ increases. In particular, our bound (5.13) on the number of sampled constraints grows polynomially in A, the cardinality of the largest action set.

Complexity in the action space can be exchanged for complexity in the state space by breaking each action under consideration into a sequence of actions taking values in smaller sets [5]. Specifically, given an alphabet with S symbols — assume for simplicity the symbols are 0,1,...,S-1 — and a finite set of actions $\mathcal{A}_x$ of cardinality less than or equal to A, actions in this set can be mapped to words of length at most $\lceil \log_S A \rceil$. Hence we can change the decision on an action $a \in \mathcal{A}_x$ into a decision on a sequence of $\lceil \log_S A \rceil$ symbols.

We define a new MDP as follows:

- States $\bar{x}$ are given by a tuple $(x, \hat{x}, i)$, interpreted as follows: $x \in \mathcal{S}$ represents the state in the original MDP; $\hat{x} \in \{0, 1, \ldots, S-1\}^{\lceil \log_S A \rceil}$ represents an encoding of an action in $\mathcal{A}_x$ being taken; $i \in \{1, 2, \ldots, \lceil \log_S A \rceil\}$ represents which entry in vector a we will decide upon next.

- There are S actions associated with each state $(x, \hat{x}, i)$, corresponding to setting $\hat{x}_i$ to $0, 1, \ldots S-1$.

- Taking action "set $\hat{x}_i$ to v" causes a deterministic transition from $\hat{x}$ to $\hat{x}^+$, where $\hat{x}_j^+ = \hat{x}_j^+$ for $j \neq i$ and $\hat{x}_i^+ = v$. The system transitions from state (x, x^a, i) to state $(x, \hat{x}^+, i+1)$ and no cost is incurred, if $i < \lceil \log_S A \rceil$. It transitions from state $(x, \hat{x}, \lceil \log_S A \rceil)$ to state $(y, \hat{x}^+, 1)$ with probability $P_a(x, y)$, where a is the action in $\mathcal{A}_x$ corresponding to the encoding $\hat{x}^+$. A cost $g_a(x)$ is incurred in this transition.

- The discount factor is given by $\alpha^{1/\lceil \log_S A \rceil}$.

It is not difficult to verify that this MDP solves a problem equivalent to that of the original MDP $(\mathcal{S}, \mathcal{A}_\cdot, P_\cdot(\cdot, \cdot), g_\cdot(\cdot), \alpha)$.

The new MDP involves a higher dimensional state space and smaller action spaces, and is hopefully amenable to treatment by approximate dynamic programming methods. In particular, dealing with the new MDP instead of the original one affects the constraint sampling complexity bounds provided for approximate linear programming. The following quantities involved in the bound are affected:

- the number of actions per state.

  In the new MDP, the number of actions per state is reduced from A to S. In principle, S is arbitrary, and can be made as low as 2, but as we will show next, it affects other factors in constraint sampling complexity bound, hence we have to keep these effects in mind for a suitable choice.

- the term $1/(1-\alpha)$.

  In the new MDP, the discount factor is increased from α to $\alpha^{1/\lceil \log_S A \rceil}$. Note that

$$\begin{aligned} 1-\alpha &= 1 - \left(\alpha^{1/\lceil \log_S A \rceil}\right)^{\lceil \log_S A \rceil} \\ &= \left(1 - \alpha^{1/\lceil \log_S A \rceil}\right) \sum_{i=0}^{\lceil \log_S A \rceil - 1} \alpha^i \\ &\leq \left(1 - \alpha^{1/\lceil \log_S A \rceil}\right) \lceil \log_S A \rceil, \end{aligned}$$

  so that $1/(1 - \alpha^{1/\lceil \log_S A \rceil}) \leq \lceil \log_S A \rceil / (1-\alpha)$.

- the number of basis functions K.

In the new MDP, we have a higher dimensional state space, hence we may need a larger number of basis functions in order to achieve an acceptable approximation to the optimal value function. The actual increase on the number of basis functions will depend on the structure of the problem at hand.

The bound on the number of constraints being sampled is polynomial in the three terms above. Hence implementation of the reduced LP for the modified version of an MDP will require a number of constraints polynomial in S and in $\lceil \log_S A \rceil$, contrasted with the number of constraints necessary for the original MDP, which is polynomial in A. However, the original complexity of the action space is transformed into extra complexity in the state space, which may incur extra difficulties in the selection of basis functions. Note the number of constraints is also polynomial in the number of basis functions. Nevertheless, there is a potential advantage of using the new MDP, as it provides an opportunity for structures associate with the action space to be exploited in the same way as structures associated with the state space are.

5.4 Closing Remarks

In this chapter, we have analyzed a constraint sampling algorithm as an approximation method for dealing with the large number of constraints involved in the approximate LP. We have shown how near-feasibility can be ensured by sampling a tractable number of constraints and established bounds on the error that near-feasibility introduces in the optimal value function approximation. The error bounds show that, under moderate assumptions, the approximation to the optimal value function yielded by the reduced LP has error comparable to the error resulting from the approximate LP. They also provide an ideal constraint sampling distribution, which can serve as a guide in the sampling process even though it cannot be calculated.

An important observation is that the bounds on the number of samples necessary to ensure near-feasibility are loose. Indeed, they only exploit the fact that constraints live in a low-dimensional space; efficiency of the constraint sampling scheme is then enabled by the fact that after a certain number of constraints is sampled, any other constraint will most likely be close to one of the already sampled constraints. Hence we do not exploit any possible regularity associated with the structure of the constraints, or a particular choice of basis functions, which might lead to much tighter bounds or even exact solution of the approximate LP. The latter approach can be found in the literature. Morrison and Kumar [39] formulate approximate linear programming algorithms for queueing problems with a certain choice of basis functions that renders all but a relatively small, tractable number of constraints provably redundant. Guestrin et al. [24] exploit the structure present in *factored MDP's* to efficiently implement approximate linear programming with an exponentially lower number of constraints. Schuurmans and Patrascu [42] devise a constraint generation scheme for factored MDP's which, although it lacks the guarantees of the algorithm presented in [24] (the worst case is still exponential), requires a smaller amount of time on average. Grötschel [23] presents an efficient cutting-plane method for the traveling salesman problem.

Exploitation of particular properties and regularities of the problem at hand is obviously useful and the constraint sampling scheme is not meant as a substitute for that; the main contribution of this chapter is to show that, even if such regularities are not known or do not exist in a given problem, variants of approximate linear programming can still be implemented efficiently.

There are several ways in which the present results can be extended, and we mention a few. It may be possible to improve the constant and exponential terms in the bounds on the number of constraints needed for near-feasibility; our main focus was on establishing that the number of constraints is tractable, rather than on derivation of the tightest possible bounds. The problem of sampling constraints according to one distribution but measuring errors according to another one, described in Section 5.3.2, pertains not only to approximate linear programming but to the UCEP theory in general, and is discussed in [57], but still remains an open question. With the impossibility of computing the stationary distribution associated with optimal policies, there is a question whether one may use an adaptive scheme to choose sampling distributions according to policies being generated in intermediate steps. Finally, the ideas on how to generate the bounded reduced LP presented here require further investigation.

Chapter 6

Case Studies

In this chapter, we present applications of approximate linear programming to queueing problems and web server farm management. The applications illustrate some practical aspects in the implementation of approximate linear programming and demonstrate its competitiveness relative to other algorithms for tackling the problems being considered.

6.1 Queueing Problems

We start with three queueing problems, with increasing state space dimensions. In all examples, we assume that at most one event (arrival/departure) occurs at each time step. The first example, involving a single queue, illustrates how state-relevance weights influence the solution of the approximate LP, testing the ideas presented in Chapter 3 against a larger and more realistic problem. The next two examples involve higher dimensional state spaces and demonstrate the performance of approximate linear programming as compared to other heuristics commonly used in queueing problems.

6.1.1 Single Queue with Controlled Service Rate

In Section 4.4.2, we studied the problem of a single queue with controlled service rate and determined that the bounds on the error of the approximate LP are uniform over the number of states. That analysis provides some guidance on the choice of basis functions; in particular, we now know that including a quadratic and a constant function guarantees that an appropriate Lyapunov function is in the span of the columns of Φ. Furthermore, our analysis of the (unknown) steady-state distribution reveals that state-relevance weights of the form $c(x) = (1-\xi)\xi^x$ are a sensible choice. In this section, we present results of experiments with different values of ξ for a particular instance of the model described in Section 4.4.2. The values of ξ chosen for experimentation are motivated by ideas developed in Chapter 3.

We assume that jobs arrive at a queue with probability $p = 0.2$ in any unit of time. Service rates/probabilities $q(x)$ are chosen from the set $\{0.2, 0.4, 0.6, 0.8\}$. The cost associate with state x and action q is given by

$$g_q(x) = x + 60q^3.$$

We set the buffer size to 49999 and the discount factor to $\alpha = 0.98$. We select basis functions $\phi_1(x) = 1$, $\phi_2(x) = x$, $\phi_3(x) = x^2$, $\phi_4(x) = x^3$ and state-relevance weights $c(x) = (1-\xi)\xi^x$.

The approximate LP is solved for $\xi = 0.9$ and $\xi = 0.999$ and we denote the solutions by r^ξ. The numerical results are presented in Figures 6.1, 6.2, 6.3 and 6.4.

Figure 6.1 shows the approximations Φr^ξ to the value function generated by the approximate LP. Note that the results agree with the analysis developed in Chapter 3; small states are approximated better when $\xi = 0.9$ whereas large states are approximated almost exactly when $\xi = 0.999$.

In Figure 6.2 we see the greedy policies induced by the scoring functions Φr^ξ. The optimal action is selected for almost all "small" states with $\xi = 0.9$. On the other hand, $\xi = 0.999$ yields optimal actions for all relatively large states in the relevant range.

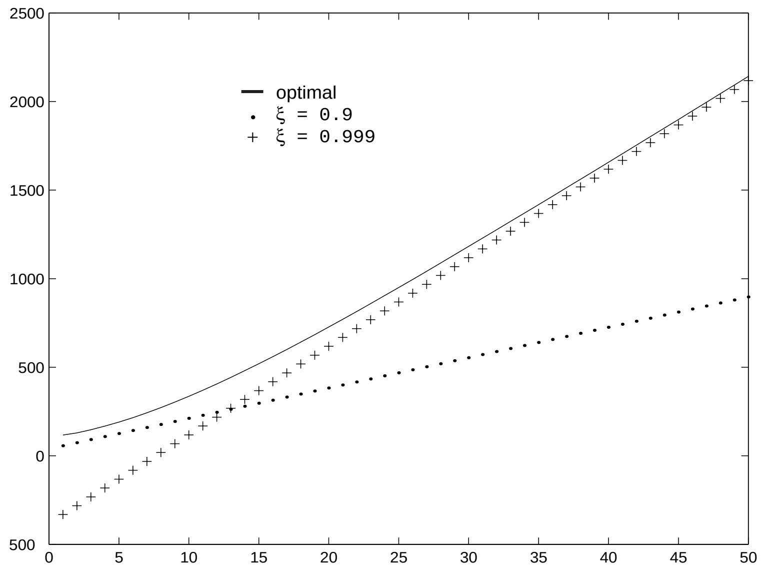


Figure 6.1: Approximate value function for the example in Section 6.1.1.

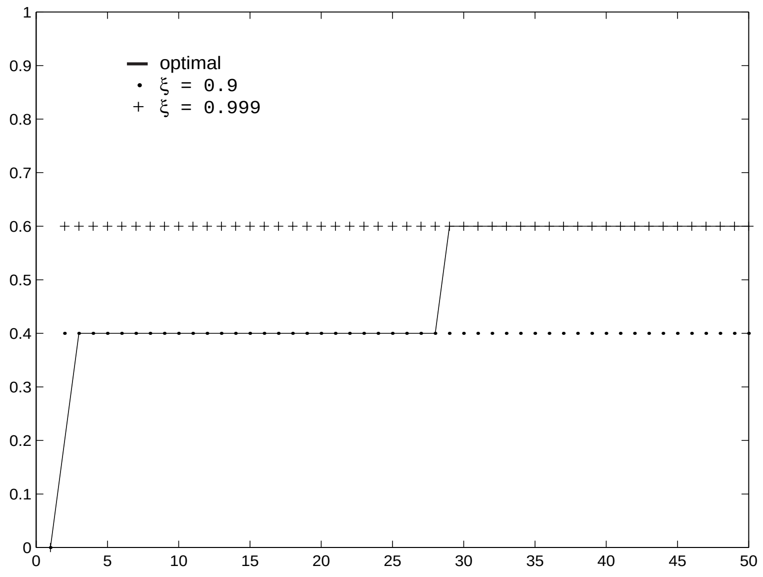


Figure 6.2: Greedy action for the example in Section6.1.1.

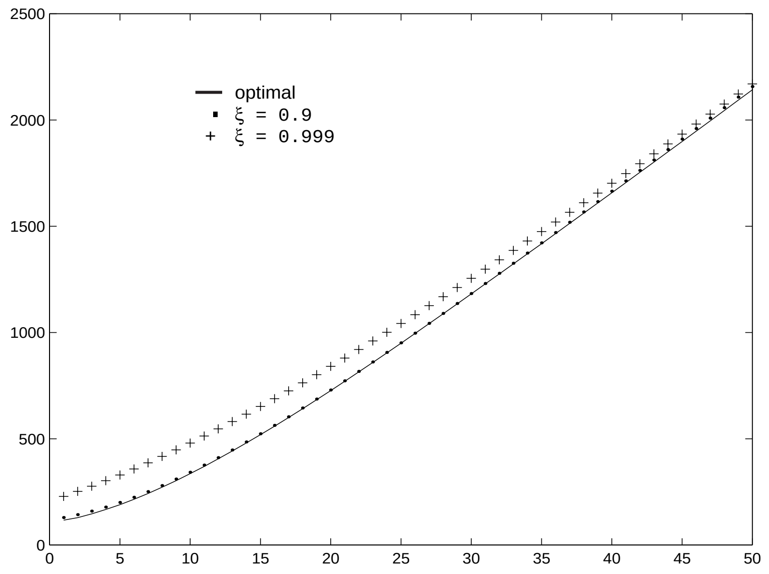


Figure 6.3: Value function for the example in Section 6.1.1.

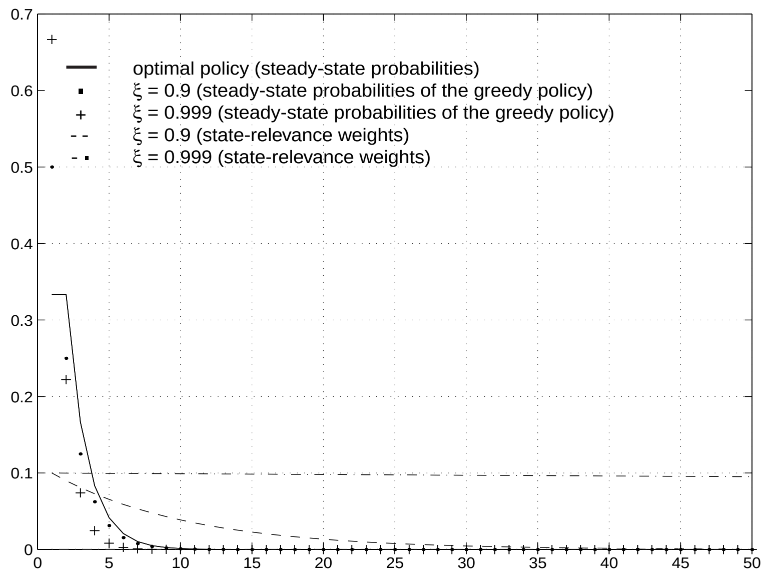


Figure 6.4: Steady-state probabilities for the example in Section 6.1.1.

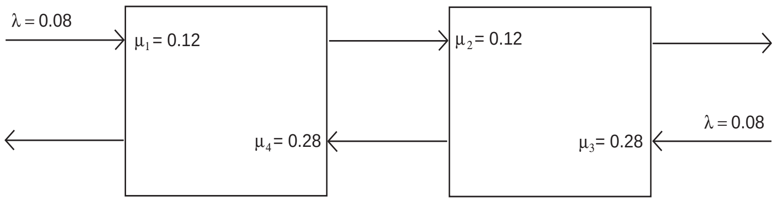


Figure 6.5: System for the example in Section 6.1.2.

The actual performance of the policies generated by approximate linear programming is illustrated in Figure 6.3, which depicts the value functions associated with these policies. Note that despite taking suboptimal actions for all relatively large states, the policy induced by $\xi = 0.9$ performs better than that generated with $\xi = 0.999$ in the range of relevant states, and it is close in value to the optimal policy even in those states for which it does not take the optimal action. Indeed, the average cost incurred by the greedy policy with respect to $\xi = 0.9$ is 2.92, relatively close to the average cost incurred by the optimal (discounted cost) policy, which is 2.72. The average cost incurred when $\xi = 0.999$ is 4.82, which is significantly higher.

Steady-state probabilities for each of the different greedy policies, as well as the corresponding (rescaled) state-relevance weights are shown in Figure 6.4. Note that setting ξ to 0.9 captures the relative frequencies of states, whereas setting ξ to 0.999 weights all states in the relevant range almost equally.

6.1.2 A Four-Dimensional Queueing Network

In this section we study the performance of the approximate LP algorithm when applied to a queueing network with two servers and four queues. The system is depicted in Figure 6.5 and it is the same as one studied in [10, 30, 41]. Arrival and departure probabilities are indicated by λ and $\mu_i, i = 1, ..., 4$, respectively.

We set the discount factor to $\alpha = 0.99$. The state $x \in \Re^4$ indicates the number of jobs in each queue, and the cost incurred in any period is $g(x) = |x|$, the total number of jobs in the system. Actions $a \in \{0, 1\}^4$ satisfy $a_1 + a_4 \leq 1$, $a_2 + a_3 \leq 1$ and the non-diddling assumption, i.e., a server must be working if any of its queues

Policy	Average cost
$\xi = 0.95$	33.37
LONGEST	45.04
FIFO	45.71
LBFS	144.1

Table 6.1: Performance of different policies for the example in Section 6.1.2. Average cost estimated by simulation after 50000000 iterations, starting with empty system.

is nonempty. We have $a_i = 1$ iff queue i is being served.

Constraints for the approximate LP are generated by sampling 40000 states according to the distribution given by the state-relevance weights c. We choose the basis functions to span all of the polynomials in x of degree 3; therefore, there are 35 basis functions.

We choose the state-relevance weights to be of the form $c(x) = (1-\xi)^4 \xi^{|x|}$. Experiments were performed for a range of values of ξ. The best results were generated when $0.95 \leq \xi \leq 0.99$. The average cost was estimated by simulation with 50,000,000 iterations, starting with an empty system.

We can compare the average cost obtained by the greedy policy with respect to the solution of the approximate LP with that of several different heuristics, namely, first-in-first-out (FIFO), last-buffer-first-served (LBFS), and a policy that always serves the longest queue (LONGEST). LBFS serves the job that is closest to leaving the system; hence priority is given to jobs in queues 2 and 4. Results are summarized in Table 6.1 and we can see that the approximate LP yields significantly better performance than all of the other heuristics.

6.1.3 An Eight-Dimensional Queueing Network

In our last example in queueing problems, we consider a queueing network with eight queues. The system is depicted in Figure 6.6, with arrival and departure probabilities indicated by $\lambda_i, i = 1, 2$, and $\mu_i, i = 1, ..., 8$, respectively.

The state $x \in \Re^8$ represents the number of jobs in each queue. The cost associated with state x is $g(x) = |x|$, and we set the discount factor α to 0.995. Actions

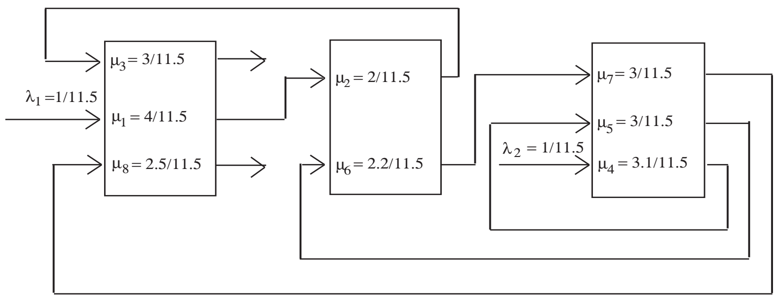


Figure 6.6: System for the example in Section 6.1.3.

$a \in \{0,1\}^8$ indicate which queues are being served; $a_i = 1$ iff a job from queue i is being processed. We consider only non-diddling policies and, at each time step, a server processes jobs from one of its queues exclusively.

We choose state-relevance weights of the form $c(x) = (1-\xi)^8 \xi^{|x|}$. The basis functions are chosen to span all polynomials in x of degree at most 2; therefore, the approximate LP has 47 variables. Constraints for the approximate LP are generated by sampling 5000 states according to the distribution associated with the state-relevance weights c. Experiments were performed for $\xi = 0.85, 0.9$ and 0.95, and $\xi = 0.9$ yielded the policy with smallest average cost.

To evaluate the performance of the policy generated by the approximate LP, we compared it with first-in-first-out (FIFO), last-buffer-first-serve (LBFS) and a policy that serves the longest queue in each server (LONGEST). LBFS serves the job that is closest to leaving the system; for example, if there are jobs in queue 2 and in queue 6, a job from queue 2 is processed since it will leave the system after going through only one more queue, whereas the job from queue 6 will still have to go through two more queues. We also choose to assign higher priority to queue 8 than to queue 3 since queue 8 has higher departure probability.

We estimated the average cost of each policy with 50,000,000 simulation steps, starting with an empty system. Results appear in Figure 6.7 and Table 6.2. The policy generated by the approximate LP performs significantly better than each of the heuristics, yielding more than 10% improvement over LBFS, the second best policy. We expect that even better results could be obtained by refining the choice

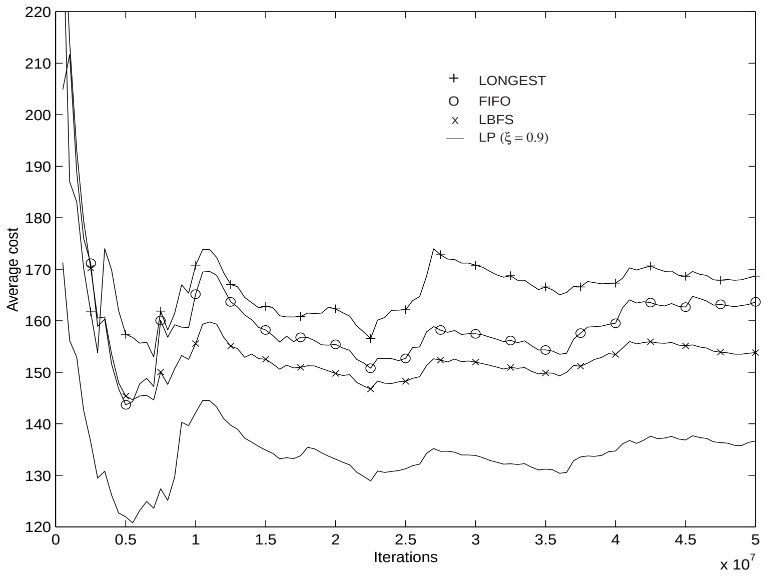


Figure 6.7: Running average number of jobs in the system for the example in Section 6.1.3.

ALP	136.67
LBFS	153.82
FIFO	163.63
LONGEST	168.66

Table 6.2: Average number of jobs in the system after 50,000,000 simulation steps for the example in Section 6.1.3.

of basis functions and state-relevance weights.

The constraint generation step took 74.9 seconds and the resulting LP was solved in approximately 3.5 minutes of CPU time with CPLEX 7.0 running on a Sun Ultra Enterprise 5500 machine with Solaris 7 operating system and a 400 MHz processor.

6.2 Web Server Farm Management

As Internet-based transactions become an increasingly important part of business, we observe a drift from relatively amateurish web site management to outsourcing of this kind of activity to companies providing information technology services. The benefits of outsourcing web site management are clear: first, businesses can rely on core competencies of information technology companies and are free to focus on their main activities; second, information technology companies having contracts with multiple web sites have the opportunity to leverage economies of scale.

The development of this new kind of service brings about a host of challenging issues. A large part of the incentive to outsourcing web site management is that specialized information technology companies may be able to provide better quality of service to web site users. Hence various questions arise: What kinds of guarantees can the web site hosting service provider guarantee? What is desirable and acceptable from the web site standpoint? How should different service-level guarantees be priced?

Besides questions on how contracts should be made, the web site hosting service provider must also face a number of questions on how to manage resources in their physical systems. Optimal resource utilization reduces costs and allows for better quality of service; furthermore, it leads to a more precise assessment of how meeting different service-level guarantees maps to use of resources. One might expect that much insight on the contract-level questions of how to price services and how to design service-level guarantees could be gained in the process of understanding how to best utilize physical resources.

Our focus in this section is on resource allocation problems. We assume that the web site hosting service provider has contracts with multiple web sites and maintains a collection of servers — a web server farm — to process requests for pages in these web sites. We consider the problems of allocation of servers to web sites, and scheduling and routing of requests for each distinct web site.

In the next section, we provide a general description of the resource allocation problems. We show how there is a natural division between the problem of server allocation and the problem of scheduling and routing, due to differences in time scales in which such decisions are made. Changes scheduling and routing are implemented relatively fast, and in our model we assume changes o be made instantaneously, whereas changing server allocations occurs over a certain length of time because it involves having files deleted from or downloaded to servers. We propose a hierarchical approach to addressing these problems. We then describe a solution to the problem of scheduling and routing. Server allocation policies are largely dependent on the statistics of web site traffic, and we introduce our model for request arrival rates in Section 6.2.3. We then formulate the server allocation problem in the MDP framework and discuss implementation of approximate linear programming for this problem, demonstrating how problems stemming from state space and action space complexity can be addressed in this setting. We present experimental results in Section 6.2.7 and offer closing remarks in Section 6.2.8.

6.2.1 Problem Formulation

We consider a collection of M web servers that are shared by N customer web sites, shown in Figure 6.8. Incoming requests can be from K distinct classes, distributed among the web sites. At time t, a request of class k arrives with rate $\lambda_k(t)$. We assume that arrivals are Poisson. After being served, a request can transition to any other class k' with probability $p_{kk'}$ or leave the system. Our first simplification will be to consider the aggregate arrival rates $L_k(t)$, which include both exogenous and endogenous arrivals, instead of addressing the transitions between classes explicitly. As discussed in [31], this leads to a simpler network model, allowing for tractable solution of the problems of scheduling and routing. Service times for class k requests are exponential with mean μ_k^{-1} .

Each class k has an associated service level agreement (SLA). An SLA is given by a maximum response time z_k and a probability β_k, and establishes that the fraction of requests with response time greater than z_k must be kept below β_k. Web sites are charged proportionally to their utilization of web servers and rates for distinct classes may differ.

Three distinct types of decisions have to be made, which we will refer to as web

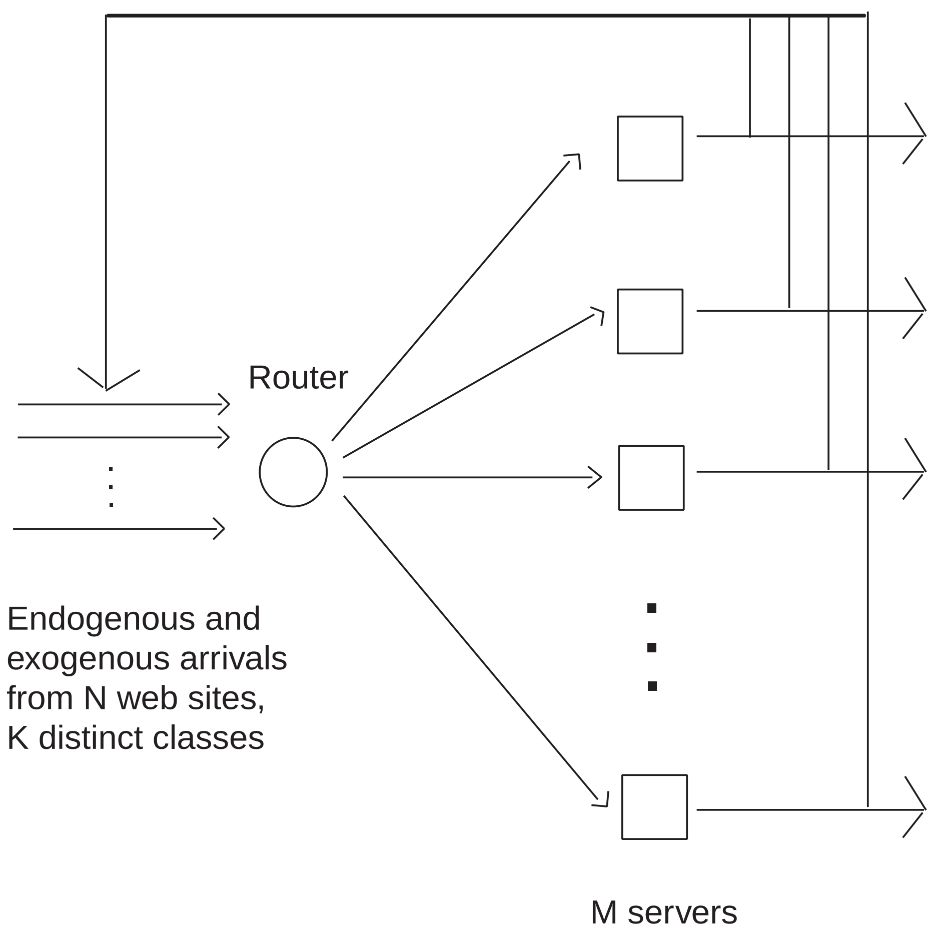

Router
Endogenous and exogenous arrivals from N web sites, K distinct classes
M servers

Figure 6.8: Web server farm

server allocation, scheduling and routing, explained as follows. We assume that at any point in time, a server is used by at most one web site. This is likely to occur in practice as it is usual for customers to demand exclusive use of servers at any point in time for security and privacy reasons. Based on the current incoming traffic and service level agreements, a decision has to be made as to how many servers to allocate to each web site — the web server allocation problem. Within the servers assigned to a web site, we need then to decide how requests from each class will be served — a scheduling problem — and finally, when a request arrives, it needs to be directed to one of the servers assigned to its web site – a routing problem. We assume that changes in scheduling and routing policies can be accomplished instantaneously, as they correspond to simple changes of rules on where to route new requests or how to serve requests in each server, whereas switching a web server from one web site to another takes 5 minutes — 2.5 minutes to remove it from a web site and 2.5 minutes to allocate it to another one, corresponding to time to delete files from the old web site and time to download files from the new web site.

Generating an optimal strategy would require us to address the web server allocation, scheduling and routing problems simultaneously to maximize the web server host's expected profits. We take a suboptimal, hierarchical approach: the scheduling and routing problem are solved via a fluid model analysis for each 2.5-minute interval during which the server allocation remains fixed, and the profits generated by such a scheme are then used as one-step rewards in a dynamic program for finding an optimal web server allocation.

In the next section, we will describe a solution to the scheduling and routing problems using a fluid model approach.

6.2.2 Solving the Scheduling and Routing Problems

We now introduce the optimization problem used in the determination of a scheduling and routing policy. Our model is an adaptation of that found in [31]; for a detailed discussion, we refer the reader to that work.

As previously discussed, we will solve the scheduling and routing problem over

2.5-minute intervals. The web server allocation and arrival rates are assumed to remain fixed in each of these intervals. We use the symbol I to denote a particular encoding of the web server allocation and let L denote a vector of aggregated endogenous/exogenous arrival rates L_k. The solution of the scheduling and routing problem is a function of the pair (I, L).

Using a fluid model approach, our routing and scheduling decision variables reduce to $\lambda_{i,k}$, the arrival rate from class k requests routed to server i, and $\phi_{i,k}$, the fraction of capacity of server i assigned to class k. The arrival rates across servers for any given class have to equal the total arrival rate for that class L_k. Also, the total assigned capacity for any given server cannot exceed 1.

The service-level agreement establishes that the response time τ_k for each class k must satisfy

$$P(\tau_k \leq z_k) \geq \beta_k,$$

for given parameters z_k and β_k. As discussed in [31], we approximate this constraint by

$$e^{(\lambda_{i,k} - \phi_{i,k}\mu_k)z_k} \leq \beta_k,$$

via a large-deviations argument.

Server usage is charged per time with rate P_k for class k. The expected time server i denotes to class k in each 2.5-minute interval is given by $\lambda_{i,k}/\mu_k$, provided that arrival rates and service times are expressed in the correct time scale, and therefore the expected profit generated by class k requests processed in server i is given by

$$P_k \frac{\lambda_{i,k}}{\mu_k}.$$

We have the following optimization problem for determining scheduling and routing policies:

$$\max_{\lambda_{i,k},\phi_{i,k}} \quad \sum_{i=1}^{M} \sum_{k=1}^{K} \mathcal{P}_k \frac{\lambda_{i,k}}{\mu_k} \tag{6.1}$$

$$
\begin{aligned}
\text{s.t.} \qquad & \lambda_{i,k} \leq \frac{\ln(\beta_k)}{z_k} + \phi_{i,k}\mu_k, \text{ if } I(i,k) = 1, i = 1, \ldots, M, k = 1, \ldots, K; \qquad (6.2)\\
& \sum_{i=1}^{M} \lambda_{i,k} \leq L_k, k = 1, \ldots, K;\\
& \lambda_{i,k} = 0, \text{ if } I(i,k) = 0, k = 1, \ldots, K, i = 1, \ldots, M;\\
& \lambda_{i,k} \geq 0, \text{ if } I(i,k) = 1, k = 1, \ldots, K, i = 1, \ldots, M;\\
& \sum_{k=1}^{K} \phi_{i,k} \leq 1, i = 1, \ldots, M;\\
& \phi_{i,k} = 0, \text{ if } I(i,k) = 0, k = 1, \ldots, K, i = 1, \ldots, M;\\
& \phi_{i,k} \geq 0, \text{ if } I(i,k) = 1, k = 1, \ldots, K, i = 1, \ldots, M.
\end{aligned}
$$

Here $I(i,k)$ is a slight abuse of notation and indicates whether server i is allocated to the web site associated with class k.

The optimal value of problem (6.1) is denoted by $R(I,L)$, corresponding to the expected profit over a 2.5-minute interval when arrival rates are given by L and the web server allocation is given by I.

The LP (6.1) can be solved analytically, which speeds up computation. Note that it decomposes into N smaller problems of scheduling and routing for each web site. Moreover, we can show that the following greedy strategy is optimal:

1. assign minimum capacity $-\ln(\beta_k)/z_k\mu_k$ to each class k;

2. assign remaining capacity as needed to classes based on a priority scheme, serving classes according to profit P_k.

Optimality of the procedure above is easily verified as follows. Suppose we have two classes k and k' with $P_k < P_{k'}$, with $\sum_{\lambda_{i,k'}} < L_k$ and $\phi_{i,k} > 0$ for some i. Then we can reallocate server capacity according to $\bar{\phi}_{i,k} = \phi_{i,k} > 0 - \epsilon$, $\bar{\phi}_{i,k'} = \phi_{i,k'} + \epsilon$ so that $\sum_{\lambda_{i,k'}} + \epsilon\mu_k \leq L_k$ and $\bar{\phi}_{i,k} \geq 0$, which is a new feasible solution to (6.1). This incurs a change in profit of

$$\frac{P_{k'}}{\mu_{k'}}\epsilon\mu_{k'} - \frac{P_k}{\mu_k}\epsilon\mu_k = P_{k'} - P_k > 0.$$

We conclude that an optimal policy must serve all requests of the most expensive classes first, hence the greedy policy is optimal.

Note that constraint

$$\lambda_{i,k} \leq \frac{\ln(\beta_k)}{z_k} + \phi_{i,k}\mu_k,$$

corresponding to the service-level agreement for class k, requires that a minimum server capacity be to requests of class k even if that server is not processing any requests of that type. This is due to the fact that the SLA constraint is based on a large-deviation limit, which is not accurate for small (or zero) arrival rates $\lambda_{i,k}$. Ideally, problem (6.1) would be reformulated to correct for that; however, that would lead to a nonconvex optimization problem. In a different relaxation of the original scheduling and routing problem, we may approximate the number of servers needed for serving all requests of class k by

$$\frac{L_k}{\mu_k + \frac{\ln(\beta_k)}{z_k}}, \tag{6.3}$$

and assign available servers to the classes associated with a web site according to that number. In this situation, server capacity is not assigned to classes with no requests being processed in a given server. Expression (6.3) is motivated by the fact that, if a server i is totally dedicated to class k requests ($\phi_{i,k} = 1$), it can process at most $\mu_k + \frac{\ln(\beta_k)}{z_k}$ requests of that type.

With expression (6.3) as an approximation for the number of servers needed for each class, we have a nearly optimal policy by assigning available servers greedily according to profit P_k/μ_k.

The web server allocation problem will be solved by dynamic programming. We will use the optimal value of problem (6.1) — R(I,L) — as one-step rewards. Dynamic programming is called for due to the time-variant nature of arrival rates: changes in the incoming traffic will typically require adjustments in the number of servers allocated to each web site. Before tackling the web server allocation problem, we will discuss the arrival rate process.

6.2.3 The Arrival Rate Process

There are two types of web page requests in a web server farm: exogenous, corresponding to users initiating a browsing session, and endogenous, corresponding to subsequent requests in an already initiated session. As mentioned before, we will consider the aggregated arrival rate, making no distinction between the two types of requests.

We consider the following model for the arrival rate process for requests of class k associated with web site i:

$$L_k(t+1) = \max(\bar{L}_k + a_k(L_k(t) - \bar{L}_k) + \sigma_k N_k(t) + M_k B_i(t), 0). \tag{6.4}$$

where $\{N_k(t), k = 1, \ldots, K, t = 0, 1, \ldots\}$ is a collection of independent standard normal random variables and $\{B_i(t), i = 1, \ldots, N, t = 0, 1, \ldots\}$ is a collection of independent Bernoulli variables.

We interpret the arrival rate process (6.4) as follows. $\bar{L}_k$ represents a prediction for the average arrival rate associated with class k. We assume that arrival rates fluctuate around their average value $\bar{L}_k$, and a_k is a scalar between 0 and 1 representing how persistent deviations from the average arrival rate are. The normal random variables $N_k(t)$ represent regular fluctuations around the average arrival rate, and the Bernoulli variables $B_i(t)$ capture arrival bursts. Note that the occurrence of a burst is associated with a web site as a whole, not with particular classes of users.

We now turn to the web server allocation problem, which is formulated in the dynamic programming framework.

6.2.4 An MDP Model for the Web Server Allocation Problem

We model the web server allocation problem in discrete time, with each time step corresponding to 2.5 minutes, corresponding to the time necessary to allocate or deallocate a server.

The state should contain all information that is important for the allocation decision. In the web server allocation problem, with the simplified model (6.4) for arrival rates presented in Section 6.2.3, a natural choice for the state variable is the pair (I, L), where I indicates the servers allocated to each web site and L is a K-dimensional vector of arrival rates.

Actions A take values on the same space as web server configurations I and indicate new web server configurations. Valid actions must satisfy the constraint that only currently deallocated servers can be assigned to a web site.

A state (I, L) under action A transitions to state $(A, \tilde{L})$ with probability $P(L, \tilde{L})$ determined by the arrival rate processes (6.4).

We have to specify rewards associated with each state-action pair. For simplicity, we will assume that the arrival rate L remains constant over each time step in the web server allocation problem, and consider the expected reward $R(I, L)$ for the static scheduling/routing decision given by the optimization problem (6.1).

We will seek to optimize the discounted infinite-horizon reward. We expect to use reasonably large discount factors (in the experiments, $\alpha = 0.99$) so that our criterion can be viewed as an approximation to average reward.

6.2.5 Dealing with State Space Complexity

In the previous section, we specified the parameters necessary to formulate the web server allocation problem as a dynamic programming problem. Ideally, an optimal policy would be determined based on the associated optimal value function. However, we observe that even with a reasonably simple model for the arrival rates process such as that given by (6.4) — and certainly with more sophisticated models that one might eventually want to consider — our problem suffers from the curse of dimensionality. The number of state variables capturing arrival rate processes grows linearly in the number of web sites being hosted, and the number of states for the mapping of servers to web sites is on the order of $O(M^N)$. Clearly, for all but very small web server farms, we will not be able to apply dynamic programming exactly. Alternatively, we use approximate linear programming.

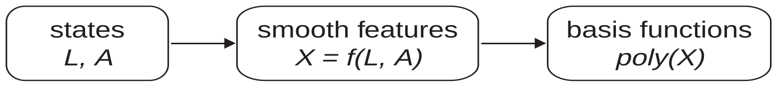


Figure 6.9: Choosing basis functions

We face certain design issues in the implementation of approximate linear programming. In this section, we discuss choices of basis functions and state-relevance weights. The web server allocation problem also requires dealing with complexity in the action space, which will be addressed in the next section.

A suitable choice of basis functions is dictated by conflicting objectives. On one hand, we would like to have basis functions that accurately reflect the advantages of being in each state. To satisfy this objective we might want to have reasonably sophisticated basis functions; for instance, values of each state under a reasonably good heuristic could be a good choice. On the other hand, choices are limited by the fact that implementation of the approximate LP in acceptable time involves the ability to compute a variety of expectations of the basis functions relatively fast. In particular, we need to compute or estimate the objective function coefficients $c^T\Phi$, which correspond to the vector of expected values of each of the basis functions conditioned on the states being distributed according to the state-relevance weights c. Expected values of the basis functions also appear in the approximate LP constraints; specifically, a constraint corresponding to state (I_k, L_k) and action A_k involves computing the expected value of the basis functions evaluated at (A_k, L_{k+1}), conditioned on the current arrival rates L_k. To keep running time acceptable, we would like to have basis functions that are reasonably simple to compute and estimate. We would also like to keep the number of basis functions reasonably small.

Our approach was to extract a number of features from the state that we thought were relevant to the decision-making process. The focus was on having smooth features, so that one could expect to find a reasonable scoring function by using basis functions that are polynomial on the features. The process is represented in Figure 6.9. After some amount of trial and error, we have identified the following features that have led to promising results:

- number of servers being used by each class, assuming that server capacity is split equally among all classes associated with each web site. Denote by $I(i)$ the number of servers allocated to web site i, and by $N(i)$ the number of classes associated with that class. The number of servers U_k being used by each class k associated with web site i is the minimum of $I(k)/N(k)$ and expression (6.3) for the approximate number of servers needed by that class.

- current arrival rates per class, given by L_k.

- average server utilization for each web site. This is computed as the ratio between the total number of servers being used by all classes associated with a given web site, assuming that server capacity is split equally among all classes, and the total number of servers assigned to that web site. More specifically, $\sum_k U_k/I(i)$.

We let server capacity be split equally among all classes associated with each web site in the choice of features for the sake of speed, as that leads to simpler expressions for the features and allows for analytical computation of certain expected values of the features and functions thereof.

We have a total of $N + 2K$ features, where N is the number of web sites and K is the total number of classes. We consider basis functions that are linear in the features, for a total of $N + 2K + 1$ basis functions.

We need to specify a distribution over the state-action pairs to serve as state-relevance weights. Recall that states are given by pairs (I, L) corresponding to the current server allocation and arrival rates. Following the ideas presented in Chapter 3, we sample pairs (I, L) with a distribution that approximates the stationary distribution of the states under an optimal policy. Since the arrival rates processes are independent of policies, it is not difficult to estimate their stationary distribution. We use the following approximation to the stationary distribution of arrival rates:

$$L_k(\infty) \approx \left(\bar{L}_k + \frac{\sigma_k}{\sqrt{1 - a_k^2}} N(0,1) + J_i \sum_{t=0}^{\infty} a_k^t B_t, 0 \right)^+ .$$

In choosing state-relevance weights, we have simplified the expression further and considered

$$L_k(\infty) \approx \max\left(\bar{L}_k + \frac{\sigma_k}{\sqrt{1-a_k^2}} N(0,1) + \frac{J_i}{1-a_k} B_0\right)^+ .$$

We sample arrival rate vectors L according to the distribution above, and then select a server allocation I based on L. While the evolution of arrival rates is independent of decisions being made in the system in our model, the same is not true for server allocations. Hence sampling web server allocations I based on the arrival rates is a more involved task. Our approach is to choose a server configuration based on the approximate number of servers needed per class, according to expression (6.3). Servers are greedily assigned to web sites corresponding to classes with the highest profits P_k/μ_k. As discussed in Section 6.1 in the context of the scheduling and routing problems, such a procedure is nearly optimal for fixed arrival rates. Hence our underlying assumption is that an optimal dynamic server allocation policy should somewhat track the behavior of optimal allocation short-run optimal allocation.

Naturally we cannot expect that states visited in the course of running the system under an optimal policy would always correspond to pairs of arrival rates L and server allocations I that are optimal with respect to L. To account for that, we randomize the allocation being sampled for each vector of arrival rate by moving some servers between web sites and between allocated and deallocated status with some probability, relatively to the optimal fixed allocation. More specifically, the randomization is performed in two steps:

1. for each originally unallocated server, with probability p_a allocated it to a web site chosen uniformly from all web sites;

2. for each originally allocated server, with probability p_d deallocate it.

The choice of probabilities for the randomization step involved some trial and error. We found that relatively high values of p_a lead to best performance overall, whereas p_d can be made reasonably small. Typical values in our experiments were $p_a = 0.9$ and $p_d = 0.1$.

The objective function coefficient $c^T\Phi$ is estimated by sampling according to the rules described above. Note that with our choice of basis functions, conditional expected values of $\phi_i(\cdot)$ involved in the constraints can be computed analytically.

In our constraint sampling scheme, we sample states according to the same procedure used for the state-relevance weights. Once a state (I, L) is sampled, we still need to sample an action A corresponding a new server allocation. We choose a feasible action as follows:

1. compute approximate expected arrival rates at the next time step

   $$L_k(+) = \bar{L}_k + a_k(L_k - \bar{L}_k) + s_k + 20 * Prob(B_i = 1)M_k.$$

   Note that we overestimate the arrival rates; the true expected arrival rate at the next time step is actually given by $\bar{L}_k + a_k(L_k - \bar{L}_k) + Prob(B_i = 1)M_k$. However, experimental results suggested that overestimating future arrival rates led to improvements in the overall quality of the policy generated by approximate linear programming; it is helpful to plan for some slack in the capacity allocated to each web site since true arrival rates are uncertain.

2. compute number of needed servers per class k for arrival rates $L_k(+)$, as given by expression (6.3). For each class, with probability p_r randomize number of needed servers by multiplying it by a uniform random variable between 0 and 1.

3. assign unused servers according to need, with priority given to classes k with higher values of P_k/μ_k.

4. determine number of servers with usage less than a threshold value (in our experiments, 40%). With probability p_r, randomize number of underused servers by multiplying it by a uniform random variable between 0 and 1.

5. deallocate underused servers.

In our experiments, p_r was set to 0.1. Note that the action sampling procedure corresponds to a randomized version of a greedy policy.

6.2.6 Dealing with Action Space Complexity

The web server allocation problem suffers from complexity in the action space, in addition to having a high-dimensional state space. In particular, a naive definition of the problem leads to a large number of actions per state: there are two choices for each allocated server (to keep it allocated or to deallocate it) and $N+1$ choices for each deallocated server (allocate it to each of the web sites or keep it deallocated).

An approach to dealing with the large number of actions is to split them into sequences of actions, as described in Section 5.3.3. A natural choice is to consider actions for each server in turn; we would have a sequence of M decisions, with each decision being a choice from at most $N+1$ values. This approach does not fully solve the problem of having a large action space; instead, we transfer the complexity to the state space. As evidenced in the discussion of the previous section, we choose an alternative approach: we "prune" the action space, using the structure present in the web server allocation problem to discard actions that are most likely suboptimal in practice.

Our pruning of the action space is based on different rules for allocating and deallocating servers. In deciding how many servers to allocate to each web site, we first generate a rough estimate of how many extra servers each class will need, given by expression (6.3) and order classes according to the associated profits P_k/μ_k. We then use the scoring function generated by approximate linear programming to decide on the *total number* of web servers to be allocated, ranging from 0 to the total number of free servers. Servers are allocated based upon need, following the profit-based class ordering. In deciding how many servers to deallocate, we first estimate their usage over the current and next time steps, and order servers based on usage. We then consider deallocating a number of servers ranging from 0 to the number of servers with usage under some threshold (in our experiments, 40%). We decide on the *total number* of servers to be deallocated based on the scoring function generated by approximate linear programming. Actual servers being deallocated are decided based on the usage ordering (least used servers are deallocated first).

	browsers	buyers
service rate (μ_k)	10	5
profit (P_k)	1	3
maximum response time (z_k)	1	2
maximum fraction of requests with service time $\geq z_k$) (β_k)	0.1	0.2

Table 6.3: Characteristics of browsers and buyers.

class \ web site	1	2	3	4	5	6	7	8	9	10
browsers	20	26	26	19	19	16	14	11	9.6	10
buyers	5	3	4	4	2.5	2	4	4	2	5

Table 6.4: Average arrival rates $\bar{L}_k$.

6.2.7 Experimental Results

To assess the quality of the policy being generated by approximate linear programming, we compared it to a fixed policy that is optimal for the average arrival rates $\bar{L}_k + M_k Prob(B_i = 1)/(1 - a_k)$. We considered problems with up to 65 web servers and 10 web sites, with 2 classes of requests per web site ("browsers" and "buyers"), for a total of 20 classes.

The actual advantage of using approximate linear programming as opposed to a fixed allocation depends on the characteristics of the arrival rate processes. In particular, in our experiments the best gains were obtained when arrivals were bursty.

Tables 6.3–6.7 present data for a representative example with 65 servers and 10 web sites. The two classes of requests per web site — "browsers and buyers" — where assumed to have the same characteristics across web sites. Service rates, price per unit of time and service-level agreement values for browsers and buyers are presented

class \ web site	1	2	3	4	5	6	7	8	9	10
browsers	0.95	0.95	0.97	0.95	0.95	0.95	0.97	0.95	0.95	0.95
buyers	0.95	0.95	0.97	0.95	0.95	0.95	0.97	0.95	0.95	0.95

Table 6.5: Persistence of perturbations in arrival rates over time a_k.

class \ web site	1	2	3	4	5	6	7	8	9	10
browsers	1.92	1.28	2.56	0.64	1.60	1.28	1.60	0.96	0.64	0.64
buyers	0.32	0.32	0.48	0.32	0.22	0.32	0.80	0.16	0.22	0.48

Table 6.6: Arrival rate fluctuation factors σ_k.

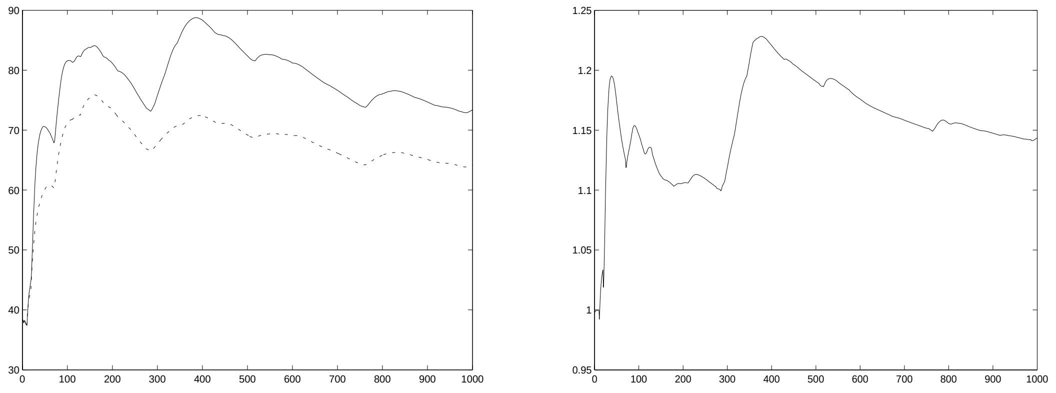
class \ web site	1	2	3	4	5	6	7	8	9	10
browsers	60	100	170	80	100	58	130	53	50	70
buyers	40	20	90	24	14	10	70	32	20	70

Table 6.7: Magnitude of bursts in arrival rates M_k.

in Table 6.3. Characteristics of web site traffic are presented in Tables 6.4–6.7. We let the probability $Prob(B_i = 1)$ of observing a burst of arrivals in any time step for each web site i be the same and equal to 0.005.

Simulation results comparing the policy generated by approximate linear programming with the fixed allocation policy optimizing for average arrival rates $\bar{L}_k + M_k/(1 - a_k)$ are given in Figures 6.10 and 6.11. The policy generated by approximate linear programming led to average profit 15% higher than the profits obtained by the fixed policy, as well as dramatic increase in the quality of service, with about half as many dropped requests. Furthermore, achieving approximately the same profit with a fixed allocation was empirically determined to require as many as 120 servers.

When the bursty component $M_k B_i(t)$ in the arrival rate processes is relatively small, gains resulting in using approximate linear programming relative to the naive fixed allocation are smaller. Note that in this situation the fluctuations are driven by the term $\sigma_k N(t)$. We have two different possibilities in this case: σ_k is small, in which case there is little fluctuation and fixed policies should do well; and σ_k is large, in which case there is much fluctuation and one might think that dynamic allocations would do better. The problem with the latter is that there is a considerable amount of noise in the system, which makes it difficult for any dynamic policy to track the variations in arrival rates fast enough. In fact, in our experiments the policies generated by approximate linear programming in this case tended to perform few changes in the server allocation, settling in a fixed allocation that had average reward comparable



(a) Running average profit. Solid line represents dynamic allocation. Dotted line represents fixed allocation.

(b) Ratio between profits for dynamic and fixed allocation.

Figure 6.10: Simulated average profit.

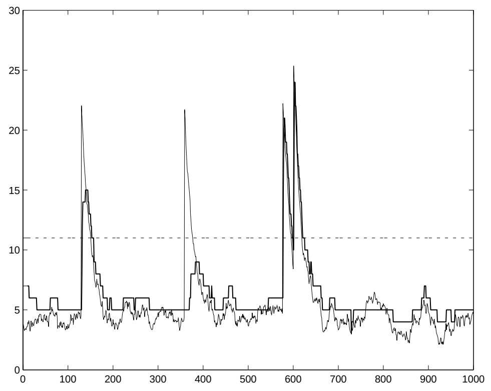
that of the fixed allocation that is optimized for average arrival rates.

The approximate linear programming algorithm was implemented in C++ and the approximate LP was solved by CPLEX 7.5 on a Sun Ultra Enterprise 6500 machine with Solaris 8 operating system and a 400 MHz processor. Reported results were based on policies obtained with approximate LP's involving 200,000 sampled constraints. The constraint sampling step took ∼46 seconds on average and solution of the approximate LP took 17 minutes.

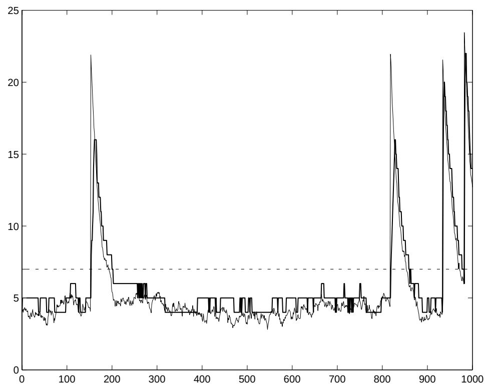
6.2.8 Closing Remarks

We proposed a model for the web server allocation problem and developed a solution via approximate linear programming. The experimental results suggest that our approach can lead to major improvement in performance compared to the relatively naive approach of optimizing allocation for average traffic, in particular when web site traffic is bursty.

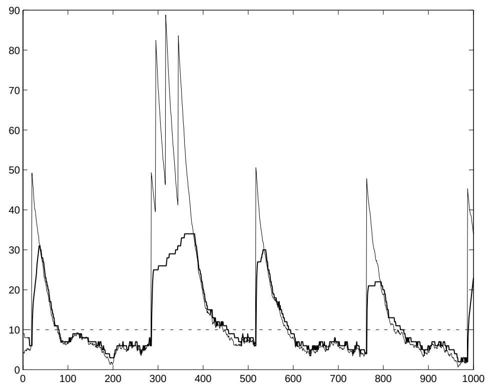
Several extensions on the current model and solution may be considered in the



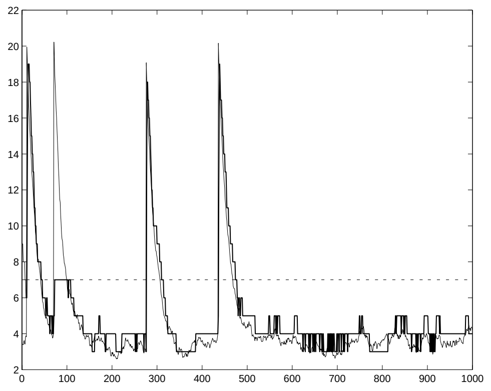
(a) Number of servers for web site 1.



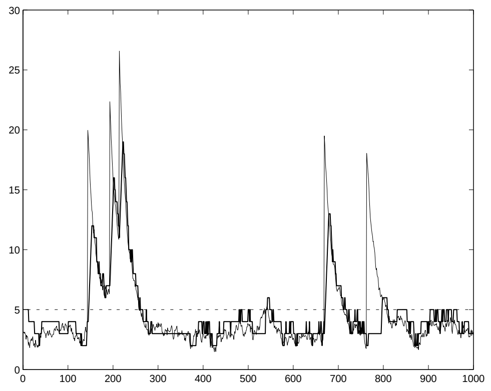
(b) Number of servers for web site 2.



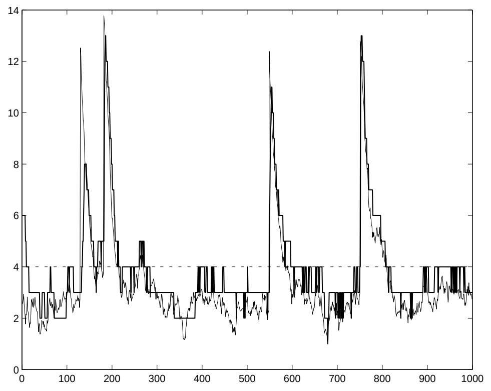
(c) Number of servers for web site 3.



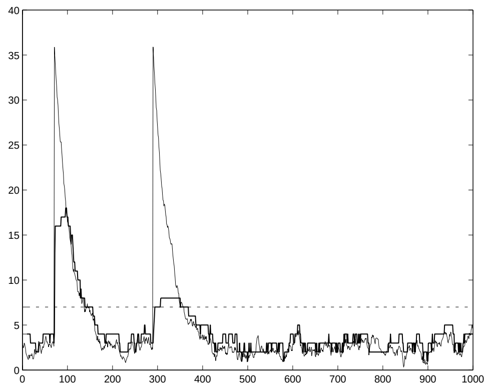
(d) Number of servers for web site 4.



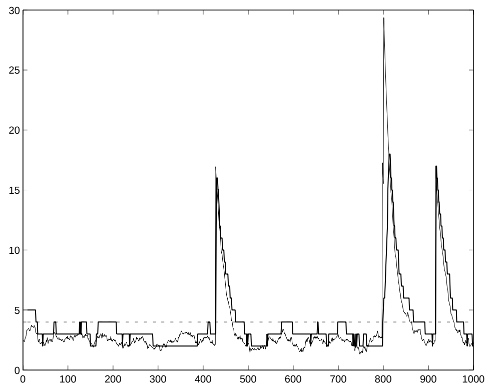
(e) Number of servers for web site 5.



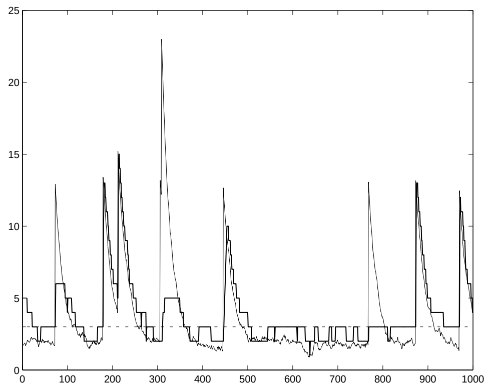
(f) Number of servers for web site 6.



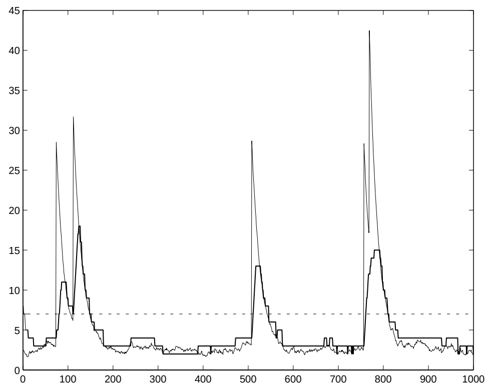
(g) Number of servers for web site 7.



(h) Number of servers for web site 8.



(i) Number of servers for web site 9.



(j) Number of servers for web site 10.

Figure 6.11: Server allocation over time. Dotted line represents fixed allocation. Dark line represents dynamic allocation. Light line represents number of servers needed.

future:

- More extensive comparison with other heuristics;
- Refinement of the arrival rate processes, with the possibility of including forecasts and accounting for cyclical patterns often observed in Internet traffic;
- Refinement of the choice of basis functions;
- Further development of the action space pruning strategies;
- Integration with the problems of scheduling and routing, by considering all of them in the dynamic programming framework, rather than solving the first two by a fluid model approximation;
- Consideration of capacity planning, where the number of servers does not remain fixed but may increase as new servers are acquired over time.

Chapter 7

Conclusions

Over the years, interest in approximate dynamic programming has been fueled by stories of empirical success in applications in areas that span from games [48] to dynamic resource allocation [12, 36, 45, 59] to finance [32, 40]. Nevertheless, practical use of the methodology remains limited by a lack of systematic guidelines for implementation and theoretical guarantees, which reflects on the amount of trial and error involved in each of the success stories found in the literature and on the difficulty in duplicating the same success in other applications. I believe that until a thorough understanding of approximate dynamic programming is achieved and algorithms become fairly streamlined, the methodology will not fulfill its potential for becoming a widely used tool assisting decision-making in complex-systems.

Part of the research presented in this dissertation represents an effort toward developing more streamlined approximate dynamic programming algorithms and bridging the gap between empirical success and theoretical guarantees. In particular, the analysis in Chapters 3 and 4 sheds light on the impact of the algorithm parameters involved in approximate linear programming on the overall performance of the algorithm. Performance of the approximate linear programming algorithm is well defined by bounds on the error in the approximation of the optimal value function and on the performance of the policy ultimately generated. The same bounds provide some guidance on the choice of basis functions. We also achieve relatively good understanding of the role of state-relevance weights, which may help reduce the amount of trial and error involved in successfully implementing approximate linear programming, as compared to other approximate dynamic programming algorithms.

The results in Chapter 5 represent a step toward making approximate linear programming applicable to a wider number of applications; implementations of the algorithm described in the literature typically require problems with special structure making exact solution of the approximate LP tractable despite the huge number of constraints [24, 39, 42], or involve a number of heuristic argument lacking theoretical guarantees [49, 50]. Our constraint sampling algorithm is the first method designed for general MDP's.

It should be noted that, while the results presented here represent advances in the understanding and development of approximate dynamic programming, questions still outnumber answers in this area. I believe that an ideal method should be able to

- distinguish between scenarios and options that are relevant and others that can be neglected;
- allow for the incorporation of prior knowledge about the problem obtained in the derivation and experimentation with heuristic solutions or through common sense;
- offer theoretical guarantees of performance so that successes and failures are easily explained;
- be reliable and simple enough to be used with confidence by practitioners in industry.

Approximate linear programming satisfies some of these conditions, but as illustrated in the case studies of Chapter 6, there is still some amount of trial and error involved in implementation of the algorithm. It is unlikely that we will ever have a fully automated algorithm for decision-making in complex systems. However, the conditions above serve as a guideline for desirable properties we should seek in the development of approximate dynamic programming algorithms.

There is much space for further developments in approximate linear programming. We mention a few possibilities:

- In principle, approximate linear programming requires explicit knowledge of the costs and transition probabilities in the system. Implementation is still possible if a system simulator is available, provided that simulation of transitions starting in any given state can be performed multiple times. In this case, one can build estimates of the costs and conditional expected values of the basis functions involved in the approximate LP constraints for each of the sampled states. It would be desirable to determine the impact of inaccuracies in these estimates on the overall performance of the approximate LP.

- Some approximate dynamic programming algorithms can run *online*, i.e., adaptively learn policies as the system runs in real time. This is particularly desirable if the system parameters are slowly changing over time because the method can then adapt to those changes. Using approximate linear programming in such instances requires re-running the algorithm from time to time when the system parameters have changed "enough." It would be useful to have a characterization of how much change is "enough," determining when it would be useful to re-run the algorithm.

- Some of the results presented here suggest iterative implementation of the method, with parameters changing over time; for instance, the bounds on the performance of the ALP policy suggest adaptive generation of state-relevance weights. Turning approximate linear programming into an iterative method raises new questions, such as whether the algorithm converges and, if it does, what kinds of performance guarantees can be offered.

- As indicated in Chapter 5, there are many open questions concerning the constraint sampling algorithm: What is the impact of not using the ideal distribution prescribed by the method for sampling the constraints? How can one efficiently identify a set of constraints ensuring boundedness of the solution of the reduced approximate LP? Can we improve the bounds on the number of constraints needed for near-feasibility?

Perhaps the most important question that has not been addressed here refers to the choice of basis functions or, more generally, of an approximation architecture. Some fairly general approaches to choosing basis functions are described in the case studies in Chapter 6, such as mapping states to smooth features and then using basis functions that are polynomial functions of features. However, such approaches lack in rigor. It is likely that there will always be some amount of problem-specific analysis and experimentation involved in the design of an approximation architecture. Nevertheless, it would be interesting to explore whether there exist canonical choices of architectures at least in some application domains, or what kinds of algorithms one could design for successively refining the choice of approximation architecture.

Bibliography

[1] L. Baird. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 11, 1998.

[2] J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning. Technical report, Research School of Information Sciences and Engineering, Australian National University, 1999.

[3] R.E. Bellman and S.E. Dreyfus. Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 13:247–251, 1959.

[4] D. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 1995.

[5] D. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[6] D. Bertsimas, D. Gamarnik, and J.N. Tsitsiklis. Performance of multiclass Markovian queueing networks via piecewise linear Lyapunov functions. Submitted to *Annals of Applied Probability*, 2000.

[7] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[8] D. Blackwell. Discrete dynamic programming. *Annals of Mathematical Statistics*, 33(2):719–726, 1962.

[9] V. Borkar. A convex analytic approach to Markov decision processes. *Probability Theory and Related Fields*, 78:583–602, 1988.

[10] R-R. Chen and S. Meyn. Value iteration and optimization of multiclass queueing networks. *Queueing Systems*, 32:65–97, 1999.

[11] V.C.P. Chen, D. Ruppert, and C.A. Shoemaker. Applying experimental design and regression splines to high-dimensional continuous-state stochast dynamic programming. *Operations Research*, 47(1):38–53, 1999.

[12] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 8, 1996.

[13] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992.

[14] D.P. de Farias and B. Van Roy. On the existence of fixed points for appproximate value iteration and temporal-difference learning. *Journal of Optimization Theory and Applications*, 105(3), 2000.

[15] D.P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. Conditionally accepted to *Operations Research*, 2001.

[16] D.P. de Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. Conditionally accepted to *Mathematics of Operations Research*, 2001.

[17] G. de Ghellinck. Les problèmes de décisions séquentielles. *Cahiers du Centre d'Etudes de Recherche Opérationnelle*, 2:161–179, 1960.

[18] E.V. Denardo. On linear programming in a Markov decision problem. *Management Science*, 16(5):282–288, 1970.

[19] F. D'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[20] R.M. Dudley. *Uniform Central Limit Theorems.* Cambridge University Press, Cambridge, 1998.

[21] R. Durrett. *Probability: Theory and Examples.* Duxbury Press, 1995.

[22] G. Gordon. *Approximate Solutions to Markov Decision Processess.* PhD thesis, Carneggie Mellon University, 1999.

[23] M. Grőtschel and O. Holland. Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, 51:141–202, 1991.

[24] C. Guestrin, D. Koller, and R. Parr. Efficient solution algorithms for factored MDPs. Submitted to Journal of Artificial Intelligence Research, 2001.

[25] S. Haykin. *Neural Networks: A Comprehensive Formulation.* McMillan, 1994.

[26] A. Hordijk and L.C.M. Kallenberg. Linear programming and Markov decision chains. *Management Science*, 25:352–362, 1979.

[27] V.R. Konda and V.S. Borkar. Actor-critic type learning algorithms for Markov decision processes. *SIAM Journal on Control and Optimization*, 38(1):94–133, 1999.

[28] V.R. Konda and J.N. Tsitsiklis. Actor-critic algorithms. Submitted to the SIAM Journal on Control and Optimization, 2001.

[29] P.R. Kumar and S. Meyn. Duality and linear programs for stability and performance analysis of queueing networks and scheduling policies. *IEEE Transactions on Automatic Control*, 41(1):4–17, 1996.

[30] P.R. Kumar and T.I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35(3):289–298, 1990.

[31] Z. Liu, M.S. Squillante, and J.L. Wolf. On maximizing service-level-agreement profits. Working paper, IBM T.J. Watson Research Center, 2000.

[32] F. Longstaff and E.S. Schwartz. Valuing American options by simulation: A simple least squares approach. *The Review of Financial Studies*, 14:113–147, 2001.

[33] D.G. Luenberger. *Optimization by Vector Space Methods.* Wiley-Interscience, 1969.

[34] A.S. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, 1960.

[35] P. Marbach. *Simulation-Based Methods for Markov Decision Processes.* PhD thesis, Massachussetts Institute of Technology, 1998.

[36] P. Marbach, O. Mihatsch, and J.N. Tsitsiklis. Call admission control and routing in integrated service networks using neuro-dynamic programming. *IEEE Journal on Selected Areas in Communications*, 18(2):197–208, 2000.

[37] P. Marbach and J.N. Tsitsiklis. Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[38] S.P. Meyn. Sequencing and routing in multiclass queueing networks, part i: Feedback regulation. To appear, *SIAM Journal of Control and Optimization*, 2001.

[39] J.R. Morrison and P.R. Kumar. New linear program performance bounds for queueing networks. *Journal of Optimization Theory and Applications*, 100(3):575–597, 1999.

[40] B. Van Roy and J.N. Tsitsiklis. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, 2001.

[41] A.N. Rybko and A.L. Stolyar. On the ergodicity of stochastic processes describing the operation of open queueing networks. *Problemy Peredachi Informatsii*, 28:3–26, 1992.

[42] D. Schuurmans and R. Patrascu. Direct value-approximation for factored MDPs. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

[43] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.

[44] C. Shannon. Programming a digital computer for playing chess. *Philosophical Magazine*, 41:356–375, 1950.

[45] S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems*, volume 9, 1997.

[46] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063, 2000.

[47] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[48] G.J. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38:58–68, 1995.

[49] M. Trick and S. Zin. A linear programming approach to solving dynamic programs. Unpublished manuscript, 1993.

[50] M. Trick and S. Zin. Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics*, 1, 1997.

[51] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[52] B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[53] B. Van Roy. *Learning and Value Function Approximation in Complex Decision Processes*. PhD thesis, Massachusetts Institute of Technology, 1998.

[54] B. Van Roy. Neuro-dynamic programming: Overview and recent trends. In E. Feinberg and A. Schwartz, editors, *Markov Decision Processes: Models, Methods, Directions, and Open Problems*. Kluwer, 2000.

[55] A.F. Veinott Jr. Discrete dynamic programming with sensitive discount optimality criteria. *Annals of Mathematical Statistics*, 40(5):1635–1660, 1969.

[56] A.F. Veinott Jr. Lectures in dynamic programming and stochastic control, Spring 1999. Lecture notes for EESOR 351, Department of Engineering-Economic Systems and Operations Research, Stanford University.

[57] M. Vidyasagar. *A Theory of Learning and Generalization*. Springer, London, 1997.

[58] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[59] W. Zhang and T.G. Dietterich. High-performance job-shop scheduling with a time-delay TD(λ) network. In *Advances in Neural Information Processing Systems*, volume 8, 1996.