Ph.D. Thesis

Simulation-Based Algorithms for Markov Decision Processes

by Ying He
Advisor: Steven I. Marcus and Michael Fu

PhD 2002-9

Simulation-Based Algorithms for Markov Decision Processes

by

Ying He

Dissertation submitted to the Faculty of the Graduate School of The
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2002

Abstract

Title of Dissertation: Simulation-Based Algorithms for Markov Decision Processes

Ying He, Doctor of Philosophy, 2002

Dissertation directed by: Professor Steven I. Marcus
Department of Electrical & Computer Engineering

Professor Michael C. Fu
Department of Decision & Information Technologies

Problems of sequential decision making under uncertainty are common in manufacturing, computer and communication systems, and many such problems can be formulated as Markov Decision Processes (MDPs). Motivated by a capacity expansion and allocation problem in semiconductor manufacturing, we formulate a fab-level decision making problem using a finite-horizon transient MDP model that can integrate life cycle dynamics of the fab and provide a trade-off between immediate and future benefits and costs.

However, for large and complicated systems formulated as MDPs, the classical methodology to compute optimal policies, dynamic programming, suffers from the so-called "curse of dimensionality" (computational requirement increases exponentially with number of states /controls) and "curse of modeling" (an explicit model for the cost structure and/or the transition probabilities is not available).

In problem settings to which our approaches apply, instead of the explicit transition probabilities, outputs are available from either a simulation model or from the actual system. Our methodology is first to find the structure of optimal policies for some special cases, and then to use the structure to construct parameterized heuristic policies for more general cases and implement simulation-based algorithms to determine parameters of the heuristic policies. For the fab-level decision-making problem, we analyze the structure of the optimal policy for a special "one-machine, two-product" case, and discuss the applicability of simulation-based algorithms.

We develop several simulation-based algorithms for MDPs to overcome the difficulties of "curse of dimensionality" and "curse of modeling", considering both theoretical and practical issues. First, we develop a simulation-based policy iteration algorithm for average cost problems under a unichain assumption, relaxing the common recurrent state assumption. Second, for weighted cost problems, we develop a new two-timescale simulation-based gradient algorithms based on perturbation analysis, provide a theoretical convergence proof, and compare it with two recently proposed simulation-based gradient algorithms. Third, we propose two new Simultaneous Perturbation Stochastic Approximation (SPSA) algorithms for weighted cost problems and verify their effectiveness via simulation; then, we consider a general SPSA algorithm for function minimization and show its convergence under a weaker assumption: the function does not have to be differentiable.

To Yingjiu and my parents ...

Acknowledgements

I wish to sincerely thank my thesis advisors, Dr. Marcus and Dr. Fu, for their guidance, encouragement and patience over the past five years. I especially appreciate their advice on how to conduct research honorably, responsibly, and ethically.

I would also like to thank Dr. Shayman, Dr. Newcomb and Dr. Hermann, for serving on my dissertation committee with invaluable comments.

Next, I want to acknowledge all of friends for their help and companionship. Specially, I am grateful to friends in my research group, Xiaodong Yao, Dr. Shalabh Bhatnagar and Abraham Thomas, for many illuminating conversations in helping me to formulate my research ideas. I am also indebted to visiting scholars, Dr. Emmanuel Fernadez-Gaucherand and Dr. Xiren Cao, with whom I collaborated on parts of my dissertation research.

I would like to thank the Department of Electrical and Computer Engineering and the Institute for Systems Research for providing me with Assistantships and Fellowships.

In addition, I own many thanks to my colleagues at GE CRD, where I worked as a summer intern in 2000. Special thanks go to Dr. Paul Houpt, Dr. Appa Madiwale, Dr. Bob Bitmead, and Dr. Allan Connolly.

Finally I want to thank my husband Yingjiu, my parents Binquan and Haitao, and my sister Rui for their constant encouragement and support.

This research was supported by a fellowship from General Electric Corporate Research and Development through the Institute for Systems Research, by National Science Foundation under Grant DMI-9713720, and by Semiconductor Research Corporation under Grants 97-FJ-491 and 2001-NJ-877.

Contents

List of Tables

List of Figures

Simulation-Based Algorithms for Markov Decision Processes

Ying He

October 8, 2002

Chapter 1

Introduction

1.1 Motivation

Problems of sequential decision making under uncertainty are common in manufacturing, computer and communication systems. Many such systems are very large and complicated. Consider a semiconductor fabrication facility capable of producing various wafers. The manufacturing process performed on each wafer contains a few hundred process steps and involves many types of equipment. Each piece of equipment can be used for various steps, and any given step can be executed on various pieces of equipment, perhaps at different rates. The complication is exacerbated by uncertainties such as frequent successive advances in technology and continual changes in demands for products. Here, decisions can be made concerning changes in product lines, recruitment of workers, equipment purchasing and discarding, the addition or deletion of products, process recipe of a product, lot dispatching, and so forth [1] [2].

Many such decision-making problems can be formulated as *Markov Decision Processes* (MDPs) [3] by defining appropriate states, actions (or controls), transition probabilities, time horizon, and cost criterion. For example, in our IPDPM (Integrating Product Dynamics and Process Models) project about planning and scheduling of semiconductor manufacturing fabs, we formulate a finite-horizon transient MDP for the fab-level decision making.

The methodology for solving MDPs is dynamic programming, based on Bellman's "Principle of Optimality": "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" [3]. This principle is often expressed by a system of equations called *optimality equations*. A finite horizon problem can be solved directly using dynamic programming, by analyzing a sequence of simpler inductively defined single-stage problems and solving these problems backward; or it can be solved by being converted into a *stochastic shortest path problem*, an infinite horizon problem with a cost-free termination state, by treating time as an extra component of the state. We will deal with the fab-level decision-making problem using the latter method. In addition, an infinite horizon problem represents a reasonable approximation of a finite horizon problem with a very large number of stages. Hence, we will focus on infinite horizon problems.

Two basic dynamic programming algorithms for solving infinite horizon MDPs are policy iteration and value iteration. Policy iteration includes a sequence of policy evaluation and policy improvement at each iteration. For problems with a total cost criterion, each policy evaluation corresponds to calculating the expected long-term cost (cost-to-go) from each state by solving linear equations with the same number of equations as the number of states; for problems with an average cost criterion, the evaluated costs are the average cost and differential costs, instead of cost-to-go. Each policy improvement step involves choosing an action for each state, where the action is "greedy" with respect to the evaluated costs. Value iteration calculates successively the optimal cost-to-go for total cost problems, or the optimal average cost and differential costs, by turning the optimality equations into update rules; the process continues until the difference between two sequential values of the evaluated costs is within some error bound.

Both policy iteration and value iteration mentioned above apply when an MDP model has a small or moderate size, measured by the number of states and number of actions. In many systems, however, MDP models are very large. For example, the MDP model for the fab-level decision making can have more than 10^{100} states and up to 10^{200} actions for some states just for a medium-sized fab. When the number of states is very large, there would be heavy storage and computational burdens due to the large number of cost-to-go functions and the large size of the transition probability matrix in the MDP model. For policy iteration, the number of linear equations would be accordingly very large. As the number of states increases linearly, the computational requirement increases exponentially, which leads to the so-called “curse of dimensionality”. When there are a large number of actions available in each state, the “greedy” search algorithm may lead to another form of “curse of dimensionality”. Value iteration has a similar problem.

Sometimes, an explicit model for the cost structure and the transition probabilities of the system is not available, in which case both policy iteration and value iteration cannot apply either. This situation causes the so-called “curse of modeling”. For example, in our fab-level decision-making problem, the exact analytic form of transition probabilities for the system is hard to obtain.

1.2 Approach

In order to break the “curse of dimensionality ”and the “curse of modeling”, we propose approaches based on the following ideas:

- simulation for the above mentioned systems is possible, in which case the task of evaluating the costs (cost-to-go, or average cost and differential costs) is transformed into that of estimating those from transitions on simulation sample paths;

- policies can be parameterized.

Namely, we would like to study simulation-based algorithms to solve MDPs which have three difficulties: large state space, large control space, and an unavailable explicit model for the cost structure and the transition probabilities.

In particular, our methodology is first to use dynamic programming to derive, or simulation-based dynamic programming to find an optimal policy structure or heuristic policy structure which can be parameterized, and then to apply simulation-based algorithms to find the optimal parameters.

Specially, we do the following:

- We show how simulation-based policy iteration algorithms give us knowledge about optimal policy structure, and we use dynamic programming to derive the structure of optimal policies for a special case of the fab-level decision-making problem.

- We develop and prove convergence for new simulation-based gradient algorithms, two-timescale and SPSA.

- We discuss implementation issues on applying simulation-based algorithms to the fab-level decision-making problem.

1.3 Related Work

There are several simulation-based policy iteration algorithms for the stochastic shortest path problem and the discounted cost problem. Some algorithms, such as Approximate Policy Iteration (API) with temporal-difference learning [4], require that each policy improvement step not be implemented until the corresponding approximate cost-to-go function converges. The merits of such algorithms are that the accuracy of the evaluated cost-to-go can be guaranteed and that the convergence of the evaluated cost-to-go has been proved. However, these algorithms would be slow since they require weighted cost-to-go function of each policy converges. To address this difficulty, Optimistic Policy Iteration (OPI), in which the policy improvement step is implemented before the convergence of the evaluated cost-to-go function, was proposed, and the algorithm has shown success in some experimental tests, but lacks theoretical support [4] [5]. Recently, another class of simulation-based policy iteration algorithms, originating from the machine learning community, the *actor-critic* algorithms [6] [7] [8], has drawn much attention. In actor-critic algorithms there is a separate memory structure that is independent of the value function that explicitly represents the policy. The policy structure is known as the *actor* or *action network*, since it is used to select actions, and the estimated value function is known as the *critic* or *critic network*, because it criticizes the actions made by the actor [9]. Konda et al. [6] connected actor-critic with simulation-based policy iteration, claiming that the need to have the policy evaluation recursion converge before implementing policy improvement is circumvented in actor-critic type simulation-based policy iteration algorithms by two-time scale stochastic approximation and providing a convergence proof of actor-critic type simulation-based policy iteration for the discounted cost problem using the ODE (Ordinary Differential Equation) method for stochastic approximation [6]. There are two time scales since the outer loop (actor) operates on a slower scale and thus sees the inner loop (critic) as essentially equilibrated, while the inner loop sees the outer one as quasi-static.

For the average cost problem, there are also several simulation-based policy iteration algorithms similar to those for discounted cost problems or stochastic shortest path problems mentioned above. However, there are two differences. The differences originate from the corresponding exact policy iteration steps.

First, with average cost problems, the evaluated costs in the policy evaluation step are the average cost and differential costs, instead of the total costs used in discounted cost problems or shortest path problems. Second, results and analyses of simulation-based policy iteration on average cost problems depend on the chain structure of the transition matrices of Markov chains generated by stationary policies, whereas those for discounted cost problems or shortest path problems do not [10] [11].

On the basis of the chain structure, MDPs for average cost problems can be classified as recurrent, unichain, or multichain [3]. An MDP is *recurrent* if the transition matrix corresponding to every deterministic stationary policy contains a single recurrent class. An MDP is *unichain* if the transition matrix corresponding to every deterministic stationary policy contains one single recurrent class plus a possibly empty set of transient states, whereas it is *multichain* if the transition matrix corresponding to at least one stationary policy consists of two or more recurrent classes.

For the average cost problem, under the assumption that the MDP is recurrent, Cao [12] proposed two single-path algorithms and provided convergence conditions; Tsitsiklis and Van Roy [13] extended the temporal-difference learning algorithm from the discounted cost case to the average cost case; Konda [6] also gave several actor-critic algorithms, and provided convergence proofs. However, the recurrence assumption seldom holds in problems of practical interest.

Under the assumption that the MDP is unichain and that there is at least one common state that is recurrent under all policies, Bertsekas [14] converted the average cost problem into a stochastic shortest path problem, and provided a multi-run scheme and corresponding error bound for simulation-based policy iteration. In this algorithm, the reference state is the same from iteration to iteration, and the differential costs are obtained by subtracting from the expected total cost the product of the average cost and expected number of transitions.

A simulation-based version of value iteration, often referred to as Q-learning, updates directly estimates of the Q-factors associated with an optimal policy [14]. Q-learning for the stochastic shortest path problem and the discounted cost problem has been analyzed by Watkins et. al. [15] and Tsitsiklis [16]. For the average cost problem, under the unichain plus common recurrent state assumption, Abounadi et al. proved the convergence of Q-learning using the ODE method [17].

Simple simulation-based algorithms only involve a lookup table representation of the cost-to-go (differential costs for the average cost simulation-based policy iteration algorithm or Q factor for the Q-learning algorithm), in the sense that a separate variable $J(i)$ is kept in memory for each state i. $J(i)$ can be calculated for the stochastic shortest path problem for instance, as the sample mean of the cumulative cost from state i to the termination state. This cost-to-go can also be evaluated by incremental methods such as the method of temporal differences.

This lookup table representation is only applicable for moderate size problems. If a given problem has very large state space, a compact representation of the cost-to-go function is needed. The cost-to-go approximator can be thought as a scheme for depicting a high-dimensional cost-to-go vector, $\tilde{J}_\mu(i, r)$, using a lower-dimensional parameter vector. Developing a cost-to-go approximate representation involves choosing an approximate architecture, a certain functional form involving a number of free parameters, and features, which are meant to represent the most important characteristics of a given state.

Broadly, approximation architectures can be classified into two main categories: linear and nonlinear ones. A linear architecture is of the general form

$$\tilde{J}(i, r) = \sum_{m=0}^{M} r(m)\phi_m(i),$$

where $r(m)$, $m = 1, \ldots, M$, are the components of the parameter vector r, and

ϕ_m are fixed, easily computable functions. A common nonlinear architecture is a neural network model such as the multilayer perceptron with a single hidden layer. There are many algorithms to train the parameters, see Chapter 3 of [4] for details.

It is often the case that the approximation architecture is too complicated for state representation and some structural pieces are used to represent states. These structural pieces are called *features*, which are fed into the approximation architecture instead of the state itself. Usually, these features are handcrafted, based on the particular problem. Some example of features include state variables, heuristic cost-to-go and/or past cost-to-go.

Simulation-based dynamic programming methodologies have been successfully applied in several problems. Some of those problems are games, such as American football, Tetris and backgammon [4]; others are a maintenance repair problem [4], communication problems such as dynamic channel allocation [4], call admission control [5], a retailer management problem [18], and missile defense and interceptor allocation problems [19]. The features are problem dependent and in some of the cases, the state vector, various combinations of its components, a heuristic policy, and the cost-to-go for some sub-optimal solution have been used as features. Linear architectures as well as more complicated nonlinear architectures have been applied.

In some cases, the policy is also parameterized, as in [20]. Then, an MDP problem can be transformed to a problem of finding the optimal parameters minimizing only the expected cost functions, and thus can be solved by simulation-based optimization algorithms, such as stochastic approximation algorithms based on gradient estimation.

Marbach [20] derived one form of the gradient of the cost function with respect to the parameters for weighted cost problems using perturbation analysis, and developed two simulation-based gradient algorithms, one that updates parameters at regenerative points and the other that updates parameters at every time step. The technique, used in the proof of convergence of the algorithm that updates parameters at regenerative points in [20], updates at every time step, and is similar to the one presented in [21]. However, [21] assumes that the transition probabilities are independent of the parameters, which is not the case in [20]. Other related work on the convergence of stochastic approximation algorithms driven by perturbation analysis can be found in [22] [23] [24]. For more about gradient estimation via perturbation analysis, see [25] [26] [27]. Here we develop a two-timescale simulation-based gradient algorithm for weighted cost problems and prove its convergence. The concept of "two-timescale algorithms" was introduced by Borkar [28], though this idea of using two timescale stochastic approximation with the faster timescale doing appropriate averaging is not new [29]. Borkar [28] mentioned that one important instance of two-timescale is the infinitesimal perturbation analysis based stochastic approximation, which requires averaging over regeneration periods. Bhatnagar et al. [30] [31] [32] [33] proposed several Kiefer-Wolfowitz-type two-timescale stochastic approximation algorithms for average cost problems. For MDPs, Borkar and Konda [34] [6] [7] also proposed actor-critic algorithms and cast them as two-timescale algorithms for convergence proofs.

Another type of stochastic algorithm based on gradient estimation is Simultaneous Perturbation Stochastic Approximation (SPSA), proposed by Spall [35], and has been successfully applied to many optimization problems [36]. Like stochastic approximation algorithms based on finite difference methods, such as the Kiefer-Wolfowitz-type stochastic approximation algorithm [37], SPSA also requires no detailed knowledge of the system dynamics. The chief merit of SPSA is that it requires only two sample estimates to calculate a gradient estimate of the objective function, regardless of the dimension of the parameter vector. Convergence of SPSA has been analyzed under various conditions. Much of the literature assumes the function to be three times differentiable [35] [38] [39] [40] [41] [42] [43], though weaker assumptions are found as well, e.g., [44] [45] [46] [47] [48]. However, all of them require that the function be at least differentiable. Among the weakest assumptions on the minimized function, Fu and Hill [45] assume that the function is differentiable and convex, and Chen et al. [44] assume that the function is differentiable and the gradient satisfies a Lipschitz condition.

The research literature on production planning and scheduling in semiconductor manufacturing is substantial and has yielded some impressive results [49] [50]. However, the majority of research on fab-level capacity allocation involves static optimization (mostly mathematical programming) [1] [51] [52] [53] [54] [55] [56] or steady-state queueing network models [57], where the objectives and parameters are stationary over time. On the other hand, in the case of life cycle dynamics and the effect of transient behavior at the fab-level, there has been significantly less research.

1.4 Contributions

The main contributions of this thesis are as follows:

First, we propose a Markov Decision Process model for fab-level decision making.

Second, we develop a simulation-based policy iteration algorithm for average cost problems under a unichain assumption, relaxing the common recurrent state assumption.

Third, we correct errors in recently proposed simulation-based gradient algorithms for weighted cost problems, compare them with a new two-timescale version that we develop, and provide a theoretical convergence proof.

Fourth, we propose two new Simultaneous Perturbation Stochastic Approximation (SPSA) algorithms for weighted cost problems and verify their effectiveness via simulation; then, we consider a general SPSA algorithm for function minimization and show its convergence under a weaker assumption: the function does not have to be differentiable.

Finally, we analyze the structure of the optimal policy for a special "one-machine, two-product" case of our fab-level decision-making problem, and discuss implementation issues on solving the model.

The thesis is organized in the following way:

In Chapter 2 we present the definition of Markov Decision Processes (MDPs) and describe the fab-level decision-making MDP model. The model employs an aggregate factory model for describing the state of the fab. Aggregation avoids excessive computational complexity, since a detailed factory model would have too many states. A policy will specify, for each possible factory state, the best actions to implement according to the objective function of the phase. Such actions include purchasing or discarding equipment, upgrading equipment and processes, and the allocation of equipment to product lines. Actions have costs that include the investment and operating cost and possible production shortages, and benefits that include increased capacity.

In Chapter 3 we develop a simulation-based policy iteration algorithm for average cost unichain problems and carry out experiments on inventory control problems. For the average cost problem, results and analyses of policy iteration, as well as simulation-based policy iteration algorithms, depend on the chain structure of the transition matrices of Markov chains generated by stationary policies, whereas those for total cost problems do not. On the basis of the chain structure, MDPs for average cost problems can be classified as recurrent, unichain, or multichain [3], with recurrent the strongest and multichain the weakest. The simulation-based algorithms for the average cost problem studied so far are under either a recurrence assumption or a unichain plus one common recurrent state assumption [3]. Here we propose a simulation-based policy iteration algorithm under a unichain assumption, relaxing the common recurrent state assumption.

In Chapter 4 we develop a new two-timescale simulation-based gradient algorithm for the weighted cost problem. Here, the weighted cost problem is an infinite horizon MDP problem with weighted expected total cost criterion, where the expected total cost is defined in the same way as that in the stochastic shortest path problem. This problem is first discussed by Marbach [20] (where it is referred to as the weighted reward-to-go problem), and it is meant to capture the situation where the decision maker wants to minimize the cost-to-go, given that the initial state is equal to a specific state with some known probability. We, as in [20], assume that policies of a given weighted cost problem can be parameterized. Then the weighted cost problem can be transformed to a problem of finding the optimal parameters minimizing only the weighted expected cost, and thus can be solved by stochastic approximation algorithms based on gradient estimation. Marbach, in [20], derived an expression of the gradient of the cost function with respect the parameters and developed two simulation-based gradient algorithms. However, there are errors in those algorithms. We propose a new two-timescale simulation-based algorithm, correct Marbach's algorithm, and compare these three algorithms via numerical experiments on a parking example.

In Chapter 5 we first describe two Simultaneous Perturbation Stochastic Approximation (SPSA) algorithms for weighted cost problems, carry out numerical experiments on the parking problem, and compare them with simulation-based gradient algorithms discussed in Chapter 4. In our SPSA algorithms, it is not necessary to have detailed knowledge of the system such as the transition probabilities, whereas such knowledge is required by the algorithms in Chapter 4.

Second, we consider a general SPSA algorithm for function minimization and show that the SPSA algorithm converges under a weaker assumption – the function does not have to be differentiable.

In Chapter 6 we deal with the fab-level decision-making MDP problem. Our methodology to solve this problem is first to characterize the structure of optimal policies for some special cases, and then to use the structure to motivate heuristic policies for more general cases and implement simulation-based algorithms to determine parameters of the heuristic parameterized policies. Sometimes we also use simulation-based policy iteration to obtain prior knowledge of optimal policies. First, we apply simulation-based policy iteration on a simple two-machine, two-product example of the fab-level decision-making problem. Numerical experiments indicate that a near-optimal policy that matches intuition can be obtained within 10 iterations. The resulting policies also give us some prior knowledge of optimal policies. However, this algorithm can not be extended to more complicated cases of the fab-level decision-making problem, since the control space expands very fast. Note that the crucial part of our methodology is obtaining parameterized near-optimal policies, for which we can use simulation-based algorithms introduced in Chapter 4 and Chapter 5 to obtain the optimal parameters of the heuristic policy. Hence, in the second part of this chapter, we focus on deriving the structure of an optimal policy for a special one-machine, two-product case. Finally, we discuss how to implement simulation-based algorithms on a testbed example, including reformulation of the finite horizon problem and parameterizing heuristic policies.

Chapter 2

Markov Decision Processes

2.1 Markov Decision Process Formulation

A Markov Decision Process is a framework containing states, actions, costs, probabilities and the decision horizon for the problem of optimizing a stochastic discrete-time dynamic system. The dynamic system equation is

$$x_{t+1} = f_t(x_t, u_t, w_t), \quad t = 0, 1, \ldots, T-1, \tag{2.1}$$

where t indexes a time epoch; x_t is the state of the system; u_t is the action to be chosen at time t; w_t is a random disturbance which is characterized by a conditional probability distribution $P(\cdot \mid x_t, u_t)$; and T is the decision horizon. We denote the set of possible system states by S and the set of allowable actions in state $i \in S$ by $U(i)$. We assume S, $U(i)$, and $P(\cdot \mid x_t, u_t)$ do not vary with t. We further assume that the sets S and $U(i)$ are finite sets, where S consists of n states denoted by $0, 1, \ldots, n-1$.

If, at some time t, the system is in state $x_t = i$ and action $u_t = u$ is applied, we incur a stage cost $g(x_t, u_t) = g(i, u)$, and the system moves to state $x_{t+1} = j$ with probability $p_{ij}(u) = P(x_{t+1} = j \mid x_t = i, u_t = u)$. $p_{ij}(u)$ may be given a priori or may be calculated from the system equation and the known probability distribution of the random disturbance. $g(i,u)$ is assumed bounded.

Consider the **infinite horizon total cost problem**, either the discounted cost problem, where there is a discount factor less than one, or the stochastic shortest path problem, in which it is assumed that there is a special cost-free termination state n in the system and the system remains there at no further cost once it reaches that state. The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_t : S \to U, \mu_t(i) \in U(i)$ for i and t, the total expected cost,

$$J_\pi(i) = \lim_{T\to\infty} E\left\{\sum_{t=0}^{T-1} \alpha^t g(x_t, \mu_t(x_t)) \mid x_0 = i\right\}. \tag{2.2}$$

where α is the discount factor with $0 < \alpha \leq 1$, with $\alpha = 1$ for the stochastic shortest path problem and $0 < \alpha < 1$ for the discounted cost problem.

A stationary policy is an admissible policy of the form $\pi = \{\mu, \mu, \ldots\}$; we denote it by μ_∞.

Under certain assumptions [14], the following hold for the infinite horizon total cost MDP problem:

- The optimal costs $J^*(0), \ldots, J^*(n-1)$ satisfy optimality equations,

  $$J^*(i) = \min_{u\in U(i)}[g(i,u) + \alpha \textstyle\sum_{j=0}^{n-1} p_{ij}(u)J^*(j)], \quad i = 0, \ldots, n-1, \tag{2.3}$$

  and in fact they are the unique solution of this equation.

- For any stationary policy μ_∞, the costs $J_\mu(0), \ldots, J_\mu(n-1)$ are the unique solution of the equation

  $$J_\mu(i) = [g(i,\mu(i)) + \alpha \textstyle\sum_{j=0}^{n-1} p_{ij}(u)J_\mu(j)], \quad i = 0, \ldots, n-1. \tag{2.4}$$

One method to solve the optimality equations is policy iteration. Policy iteration consists of a sequence of policy evaluation and policy improvement at each iteration. At each iteration step k, a stationary policy $\mu_\infty^k = \{\mu^k, \mu^k, \ldots\}$ is given.

1. **Policy evaluation**: obtain the corresponding cost-to-go $J_{\mu^k}(i)$ satisfying (2.4).
2. **Policy improvement**: find a stationary policy μ^{k+1}, where for all i, $\mu^{k+1}(i)$ is such that

$$g(i, \mu^{k+1}(i)) + \alpha \sum_{j=0}^{n-1} p_{ij}(\mu^{k+1}(i)) J_{\mu^k}(j) = \min_{u \in U(i)} [g(i,u) + \alpha \sum_{j=0}^{n-1} p_{ij}(u) J_{\mu^k}(j)]. \tag{2.5}$$

If $J_{\mu^{k+1}} = J_{\mu^k}$ for all i, the algorithm terminates; otherwise, the process is repeated with μ^{k+1} replacing μ^k.

Under certain assumptions, the policy iteration algorithm terminates in a finite number of iterations with an optimal stationary policy.

Another method of solving the optimality equations is value iteration. It is done by using the recursion

$$J_{k+1}(i) = \min_{u \in U(i)} [g(i,u) + \textstyle\sum_{j=0}^{n-1} p_{ij}(u) J_k(j)], \quad i = 0, \ldots, n-1. \tag{2.6}$$

given any initial conditions $J_0(0), \ldots, J_0(n-1)$, to compute the sequence $J_k(i)$.

For the **infinite horizon average cost problem**, the objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_t : S \to U, \mu_t(i) \in U(i)$ for i and t, the average cost per stage

$$J_\pi(i) = \lim_{T \to \infty} \frac{1}{T} E \left\{ \sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i \right\}. \tag{2.7}$$

A stationary policy is an admissible policy of the form $\pi = \{\mu, \mu, \ldots\}$; we denote it by μ_∞.

In an average cost MDP, results and analyses depend on the chain structure corresponding to stationary policies.

Under the unichain assumption, the following hold for average cost MDPs [3] [14]:

- The average cost per stage associated with an initial state i and a stationary policy μ_∞, $J_\mu(i)$, and the optimal average cost from state i, $J^*(i)$, are independent of the initial state i. We denote these by η_μ and η^*, respectively.

- The optimal average cost η^* together with some vector $h^* = \{h^*(0), \ldots, h^*(n-1)\}$ satisfies the optimality equations

$$\eta^* + h^*(i) = \min_{u \in U(i)}[g(i,u) + \textstyle\sum_{j=0}^{n-1} p_{ij}(u)h^*(j)], \quad i = 0, \ldots, n-1. \tag{2.8}$$

  Furthermore, if $\mu(i)$ attains the minimum in the above equation for all states, the stationary policy μ_∞ is optimal. In addition, we may set $h^*(0) = 0$ to uniquely determine the vector h^*, which is also called the optimal differential cost.

- Given a stationary policy μ_∞ with corresponding average cost per stage η_μ, there is a unique vector $h_\mu = \{h_\mu(0), \ldots, h_\mu(n-1)\}$ such that $h_\mu(0) = 0$ and

$$\eta_\mu + h_\mu(i) = g(i, \mu(i)) + \textstyle\sum_{j=0}^{n-1} p_{ij}(\mu(i))h_\mu(j), \quad i = 0, \ldots, n-1. \tag{2.9}$$

  h_μ is also called the differential cost associated with a stationary policy μ_∞.

The policy iteration algorithm for the average cost problem is as follows: at each iteration step k, a stationary policy $\mu_\infty^k = \{\mu^k, \mu^k, \ldots\}$ is given.

1. **Policy evaluation**: obtain the corresponding average and differential cost η^k and $h^k(i)$ satisfying (2.9).

2. **Policy improvement**: find a stationary policy μ^{k+1}, where for all i, $\mu^{k+1}(i)$ is such that

$$g(i,\mu^{k+1}(i)) + \sum_{j=0}^{n-1} p_{ij}(\mu^{k+1}(i))h^k(j) = \min_{u\in U(i)}[g(i,u) + \sum_{j=0}^{n-1} p_{ij}(u)h^k(j)]. \tag{2.10}$$

If $\eta^{k+1} = \eta^k$ and $h^{k+1}(i) = h^k(i)$ for all i, the algorithm terminates; otherwise, the process is repeated with μ^{k+1} replacing μ^k.

Under the unichain assumption, the policy iteration algorithm terminates in a finite number of iterations with an optimal stationary policy. See [3] for multichain problems.

Consider **the finite horizon problem**, where T is finite. The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \ldots \mu_{T-1}\}$ with decision rules $\mu_t : S \to U, \mu_t(i) \in U(i)$ for i and t, the total expected cost

$$J_\pi(i) = E\left\{G(x_T) + \sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i\right\}. \tag{2.11}$$

The optimality equations are given by:

$$\begin{aligned} &J_T^*(i) = G(i); && i = 0,\ldots,n-1 \\ &J_t^*(i) = \min_{u\in U(i)} \textstyle\sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + J_{t+1}^*(j)). \end{aligned} \tag{2.12}$$

Note that the finite horizon problem can be converted into the stochastic shortest path problem by viewing time as an extra component of the state. In the reformulation, transitions occur from state-time pairs $[i,t]$ to state-time pairs $[j,t+1]$ according to the transition probabilities $p_{ij}(u)$ of the finite horizon problem; the termination state corresponds to the end of the horizon; it is reached in a single transition from any state-time pair of the form $[j,T]$ at a terminal cost $G(j)$ [4]. The reformulation is as follows:

$$\begin{aligned} &J^*([i,T]) = G(i); && i = 0,\ldots,n-1 \\ &J^*([i,t]) = \min_{u\in U([i,t])} \textstyle\sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + J^*([j,t+1])). \end{aligned} \tag{2.13}$$

So potentially, policy iteration or value iteration algorithms for the stochastic shortest path problem can apply to the finite horizon problem.

2.2 Fab-Level Decision-Making Problem

2.2.1 Fab-Level Decision Making MDP Model

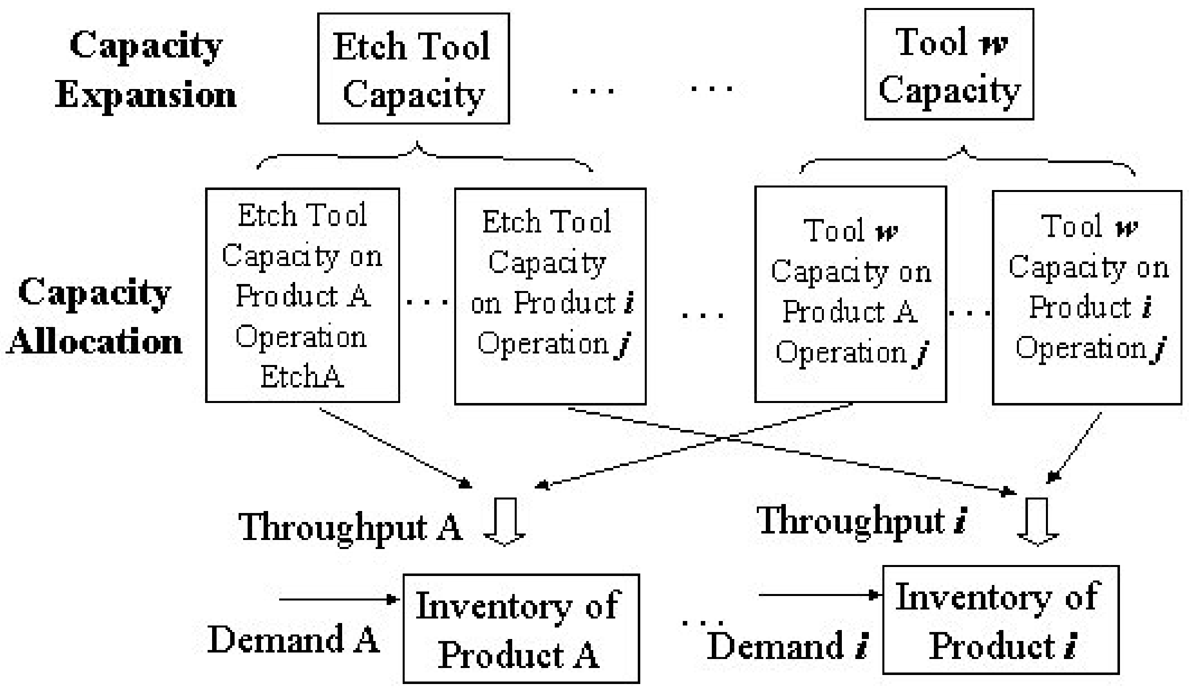
In our IPDPM project, we proposed a Markov Decision Process (MDP) model for the highest level of the hierarchy that will yield decision support for operating the fab in each of the phases of its life cycle and include life cycle dynamics [58].

The MDP can include life cycle dynamics such as technology shrink. MDP models result in policies that utilize the available information in a way that provides a trade-off between immediate and future benefits and costs, and that utilizes the fact that observations will be available in the future (cf., e.g., [3], or [14]). The proposed MDP employs an aggregate factory model for describing the state of the fab. Aggregation avoids excessive computational complexity, since a detailed factory model would have too many states. A policy specifies, for each possible factory state, the best actions to implement according to the objective function of the phase. Such actions include purchasing (or discarding) equipment, upgrading equipment/processes, and the allocation of equipment to product lines. Actions have costs that include the investment and operating cost and possible production shortages (from the production targets), and benefits that includes increased capacity (maximum throughput).

The costs, benefits, and the system state are subject to random uncontrollable events that are both exogenous and endogenous to the fab: equipment may be delivered late or may fail, the performance of newly-installed equipment is uncertain, and the market for certain chips may collapse.

The chief components of the MDP model are the following:

- the factory state – aggregate summary of factory operations sufficient to characterize the objectives and potential actions;
- the actions to be taken – higher-level decisions such as the addition of new process technologies (day-to-day operational decisions such as work order releases and detailed scheduling are left to lower-level decision models that follow the chosen policies);
- the objective function – a key feature of the model is that the weights will differ for each of the life cycle phases;
- sources of uncertainty – characterize the dynamic nature of the system over the life cycle and hence change over time.



Capacity Expansion
Etch Tool Capacity
Tool *w* Capacity
Capacity Allocation
Etch Tool Capacity on Product A Operation EtchA
Etch Tool Capacity on Product *i* Operation *j*
Tool *w* Capacity on Product A Operation *j*
Tool *w* Capacity on Product *i* Operation *j*
Throughput A
Throughput *i*
Demand A
Inventory of Product A
Demand *i*
Inventory of Product *i*

Figure 2.1: Fab-Level Decision Making MDP Model

Our proposed MDP model (see Figure 2.1) for higher level control is one in which the aggregate factory state is summarized by a vector of capacities $\overline{X}(t)$ at time epoch t, where the components $X_{(l,i),w}(t)$ represent the capacity (measured, for example, in wafer starts per day or number of machines) of type w allocated to product l and operation i (this could be a type of sub-factory manufacturing a particular product or a type of process). Actions to be taken could be the decisions to

(i) increase the capacity of type w, possibly by the introduction of new technology; or

(ii) switch over units of type w capacity from product l and operation i to product m and operation j (for example, by qualifying tools for a different process).

Randomness is explicitly modeled by the demand $d_l(t)$ for product l. The dynamics of the model includes the fact that, after the decision is made to increase capacity, there is a delay, possibly random, in the ability to fully utilize the increased capacity, and that the capacity may gradually ramp up to the expected level. The evaluation criteria include a number of factors, including costs for excess capacity (overage) and capacity shortages (underage), cost of production, cost of converting capacity from one type of operation to another, and the cost of increasing capacity. A more precise description of the model is given below.

We denote by T the duration of the planning horizon and by t the periods i.e., $t \in \{1, 2, .., T\}$. Each product is characterized by a sequence of operations, where an operation is defined as a job to be performed on a wafer. There are typically many operations that are performed on a wafer before the final product emerges. Also, an operation may be performed several times on a wafer. For instance, for a product which requires operations i, j and k to be performed, the actual sequence of operations before the final product (say P) comes out could (in the order from left to right) be $iijjjkijkk$. In what follows, we shall not take into account the particular sequences of operations required for manufacturing the products. However, we shall take into account the number of operations of a particular type required for any given type of product. Thus in the example above, product P requires three i, four j and three k operations to be performed on it. We shall measure capacity in terms of the number of machines or tools. Note that in general, machines or tools are capable of performing one or more types of operations. Thus a machine capable of performing operations i, j and k could be used for manufacturing all those products which require one or more of these operations (and in any numbers and combinations of them).

We now introduce more necessary notation for the MDP model. Let N_t represent the total number of operations associated with all the machines in the firm, in period t. For simplicity, we shall denote the operations by $1, 2, .., N_t$. Note that N_t depends on t since in any period t, one could decide to purchase new equipment (machines) which allows a greater number of operations. Let $\mathcal{X}_t$ be the set of all operations in period t, or $\mathcal{X}_t = \{1, .., N_t\}$. Let 0 represent 'no operation'. Let $\mathcal{P}_t$ be the set of products that the firm produces in period t, plus the 'no product' element 0, i.e., $0 \in \mathcal{P}_t$. Note that $\mathcal{P}_t$ set also depends on t, so that product mixes may change over time. If a machine is idle, we can say that it is manufacturing product 0, by performing operation 0. We call a *word* as any lexicographic ordering of elements of $\mathcal{X}_t$ with the first letter 0 in it. Let $\mathcal{Z}_t$ represent the set of all possible words. We call the capacity associated with word w as type w capacity. Specifically, if there is a nonzero capacity associated with a particular word $\bar{w}$, it would mean that there exists at least one machine in the facility which is capable of performing all operations in the word $\bar{w}$. Later, we shall also define the set of all *feasible words*. In the following, the various quantities are indexed by terms of the type (l, i) and w. These will be used to indicate the corresponding quantities associated with product l and operation i on machines of type w. Note that in the tuple (l, i), if either l or i is zero, the other quantity (i or l respectively) will automatically be zero. Also, if a particular combination of l, i and w is infeasible, all the corresponding quantities containing (l, i) and w will be assumed to be taking value zero. The following notation will also be used:

- $T_w(t)$: total type w capacity in period t.
- $B_w(t)$: total amount of new type w capacity bought in period t.
- $D_w(t)$: total amount of old type w capacity discarded in period t.
- $U_w(t)$: total amount of available reserve capacity of type w, in period t.
- $X_{(l,i),w}(t)$: amount of available type w capacity allocated to product l, for operations of type $i \in w$ ($l \neq 0$, $i \neq 0$), in period t.
- K_w: availability factor for type w machines defined as the long-run average fraction of times that type w machines are available.
- $C_{(l,i),w}$: number of wafers per unit time of product l produced after type i operations are performed on machine w, constant for given l, i and w; $C_{(0,0),w} = 0 \ \forall \ w \in \mathcal{Z}_t$.
- $F_{(l,i),w}$: number of type i operations in product l, on machines (capacity) of type w.
- $d_l(t)$: demand for product l, in period t.
- $I_l(t)$: inventory for product l, in period t.
- $V_w^{(l,i),(m,j)}(t)$: amount of type w capacity switched over from product l and operations of type i to product m and operations of type j, in period t, for $i, j \in w$, $l \neq 0$ and $i \neq 0$.

- $V_w^{(0,0),(m,j)}(t)$: amount of available reserve and/or newly bought capacity of type w allocated to product m and operations of type $j \in w$, in period t.
- $V_w^{(l,i),(0,0)}(t)$: portion of allocated type w capacity for product l and operations of type i, taken away (sent to reserve or permanently discarded), in period t.
- $C_w^a(x)$: cost of increasing type w capacity by purchasing x units of new capacity.
- $C_w^b(x)$: cost of decreasing type w capacity by discarding x units of old capacity.
- $C_w^c(x)$: cost of switching over x units of type w capacity from one type of production and/or operation to another.
- $C_l^d(y)$: inventory holding/backlogging cost for y units of product l.
- $C_w^e(x)$: operating cost for x units of type w capacity.

We now define the set of all *feasible words* $\mathcal{A}_t \subset \mathcal{Z}_t$ as follows. Let $\mathcal{A}_1 \triangleq$ the set of all words with nonzero total capacity associated with them in the first period of the planning horizon and such that each word is characterized by a set of equipment that can perform all the operations in it. Also for subsequent periods ($t > 1$), let $\mathcal{A}_{t+1} \triangleq \mathcal{A}_t \cup \{w \notin \mathcal{A}_t | B_w(t+1) > 0\}$. In what follows, we shall restrict our attention to the set of all *feasible words* $\mathcal{A}_t$, in period t, since it contains all words which currently have or had in an earlier period nonzero total capacity associated with them.

Remark: Note that $\mathcal{A}_t$ for $t > 1$ may also contain words with zero total capacity associated with them in period t, as a result of discarding capacity in a previous period. Words of the type $0i \in \mathcal{A}_t$ represent dedicated capacity for product i. The availability factor K_w for type w machines is required to take into account the effect of breakdowns, periodic maintenance etc. Also for $w \in \mathcal{A}_t$, the available reserve capacity $U_w(t) \triangleq K_w T_w(t) - \sum_{\{(l,i)\in\mathcal{P}_t\times w | l\neq 0, i\neq 0\}} X_{(l,i),w}(t)$ is the capacity available in period t after the various capacity allocations to the products and operations have been made.

The state vector at time t is given by

$$\overline{X}(t) = \left[T_w(t), X_{(l,i),w}(t), I_l(t)\right],$$

$l \in \mathcal{P}_t \backslash \{0\}, i \in w\backslash\{0\}, w \in \mathcal{A}_t$, and the action vector at time t is

$$\overline{U}(t) = \left[B_w(t), D_w(t), V_w^{(l,i),(m,j)}(t)\right],$$

$l, m \in \mathcal{P}_t, i, j \in w, w \in \mathcal{A}_t$, where it is assumed that $V_w^{(l,i),(m,j)}(t) = 0$ if $(l,i) = (m,j)$. At the beginning of a period, the decision maker observes the system state and chooses an action.

The total cost over the entire planning horizon that we want to minimize is

$$J = E\left[\sum_{t=0}^{T-1} g(\overline{X}(t), \overline{U}(t))\right]$$

where

$$\begin{aligned} g(\overline{X}(t), \overline{U}(t)) &= \sum_{w\in\mathcal{A}_t} (C_w^a(B_w(t)) + C_w^b(D_w(t))) \\ &+ \sum_{w\in\mathcal{A}_t} \sum_{\{(l,i),(m,j)|l,m\in\mathcal{P}_t, i,j\in w,(l,i)\neq(m,j)\}} C_w^c(V_w^{(l,i),(m,j)}(t)) \\ &+ \sum_{l\in\mathcal{P}_t} (C_l^d(I_l(t)) + \sum_{w\in\mathcal{A}_t} \sum_{\{(l,i)\in\mathcal{P}_t\times w\}} C_w^e(X_{(l,i),w}(t)) \end{aligned}$$

We have the following state equations ($w \in \mathcal{A}_t$):

$$T_w(t+1) = T_w(t) + B_w(t) - D_w(t), \tag{2.14}$$

$$\begin{aligned} X_{(l,i),w}(t+1) &= X_{(l,i),w}(t) \\ &+ \sum_{\{(m,j)\in\mathcal{P}_t\times w|(m,j)\neq(l,i)\}} (V_w^{(m,j),(l,i)}(t) - V_w^{(l,i),(m,j)}(t)), \end{aligned} \tag{2.15}$$

where $l \in \mathcal{P}_t, i \in w, l \neq 0, i \neq 0$,

$$\begin{aligned} I_l(t+1) &= I_l(t) \\ + \min_i &\left\{ \sum_{\{w \in \mathcal{A}_t | F_{(l,i),w} > 0\}} \frac{1}{F_{(l,i),w}} C_{(l,i),w} X_{(l,i),w}(t) \right\} \\ &\qquad -d_l(t),\ l \in \mathcal{P}_t \backslash \{0\}, \end{aligned} \tag{2.16}$$

The constraints are as follows ($w \in \mathcal{A}_t$):

$$\sum_{\{(m,j) \in \mathcal{P}_t \times w | (m,j) \neq (l,i)\}} V_w^{(l,i),(m,j)}(t) \leq X_{(l,i),w}(t), \quad i \in w,\ l \neq 0,\ i \neq 0, \tag{2.17}$$

$$\sum_{\{(m,j) \in \mathcal{P}_t \times w | m \neq 0, j \neq 0\}} V_w^{(0,0),(m,j)}(t) \leq U_w(t) \triangleq K_w T_w(t) - \sum_{\{(m,j) \in \mathcal{P}_t \times w | m \neq 0, j \neq 0\}} X_{(m,j),w}(t),\ w \in \mathcal{A}_t, \tag{2.18}$$

$$V_w^{(l,i),(m,j)}(t) \geq 0, \tag{2.19}$$

$$T_w(t),\ B_w(t) \geq 0, \tag{2.20}$$

$$T_w(t) \geq D_w(t) \geq 0. \tag{2.21}$$

The second term on the RHS of Equation (2.16) is the ‘throughput term’ in the inventory equation and gives the number of ‘finished wafers’ of product l in period t; thus, the bottleneck operation for a particular product for the given state of allocated capacity is essentially the operation(s) yielding the minimum term (i.e., the arg min) in Equation (2.16). There is no machine work-in-process inventory explicitly considered (only finished product inventory), as this is meant to be a higher-level planning model, but the difference between reserve capacity and excess capacity at a non-bottleneck machine is still captured in the last term of the cost function, in the form of an operational charge for allocated capacity (versus no such charge for reserved capacity). In Equation (2.17), the cases $m = 0$ and $j \neq 0$ and $m \neq 0$ and $j = 0$ do not arise, since we have already mentioned that if either of m or j is 0, the other one is automatically 0. Also, note that Constraints (2.17) and (2.19) imply $X_{(l,i),w}(t) \geq 0$, $\forall l \in \mathcal{P}_t, i \in w$, and similarly Constraints (2.18) and (2.19) imply $U_w(t) \geq 0$; hence, these two sets of constraints are not included explicitly as separate constraints in our formulation.

2.2.2 Cost Model

The cost structure for our model is given by $\{C_w^a(x),\ C_w^b(x),\ C_w^c(x),\ C_l^d(y),\ C_w^e(x)\}$, which is designed to capture most cost factors in fab-level decision making. Next, we will identify major cost elements in each cost category, propose ways to estimate them, and discuss various approaches to obtain parametric values of the cost model.

The cost for additional capacity, $C_w^a(x)$, covers equipment purchase cost, equipment installation cost, equipment qualification cost, training cost, and necessary new facility cost (e.g., additional clean room). Installation cost is characterized as a certain percentage of total equipment purchase cost. Qualification costs represent the costs directly involved in the initial processing of wafers to establish that the equipment is performed within specifications; and the cost includes the total labor cost involved and the production revenue lost during the period and the cost of wafer used. Training cost is counted in man-hours spent on training engineers, maintenance men, and operators. New facility cost is proportional to the floor space (square feet) of the clean room.

The cost for discarding capacity, $C_w^b(x)$, is equal to the residual value of the equipment, and can be handled by the so-called straight line depreciation method in accounting. If a tool is discarded within its life time (also called depreciation life), the cost for discarding is the original tool cost multiplied by a ratio which equals (life time - used time)/ life time; otherwise, there is no cost for discarding.

The switch-over cost, $C_w^c(x)$, also depending on products, is a function of the tool set-up time and labor change time due to switch over. Note there is no switch-over cost for batch tools if products are switched among that batch tool group. And there is no switch-over cost for sending capacity to reserve.

Inventory holding cost and backlogging cost are both described in $C_w^d(y)$. Inventory holding cost is related to inventory quantity, item value and length of time the inventory is carried. The cost consists of the cost of capital, variable costs of taxes and insurance on inventories, and the cost involved in storing inventory. Backlogging cost, in our case, is equal to the product's contribution margin, which is the difference between the selling price and unit production cost. Note that $C_w^d(y)$ can take market dynamics into account.

The operating cost, $C_w^e(x)$ covers the handling cost to load and unload product, tool recurring cost, labor cost for operating tools, utility cost, supplies and consumables cost, and material cost (e.g. mask), etc.

Many of the parametric values of the cost model developed are readily available or estimated in the accounting departments or other departments in a company. Labor costs, for instance, are available in the personnel department; tool purchasing costs can be obtained in the purchasing department. Inventory costs can be obtained from warehouses. The lost profit due to backlogging is not known before the total cost and income are calculated, but it can be estimated from accounting records using regression analysis. Total set-up time for each piece of equipment is typically obtained from recipes of processes. In addition, parameters can be obtained from parameters of COO models in industry.

2.2.3 Demand Model

We model the demand in our operational model as stochastic processes and we consider three demand patterns. It is assumed that there are three products A, B, and C, in the same product family. For example, CMOS8, CMOS10 and CMOS12 can be three products in a memory chip product family.

In pattern 1 and pattern 2, we assume that demands are independent among types of products and over periods. In pattern 3, we assume that demands for different products are correlated.

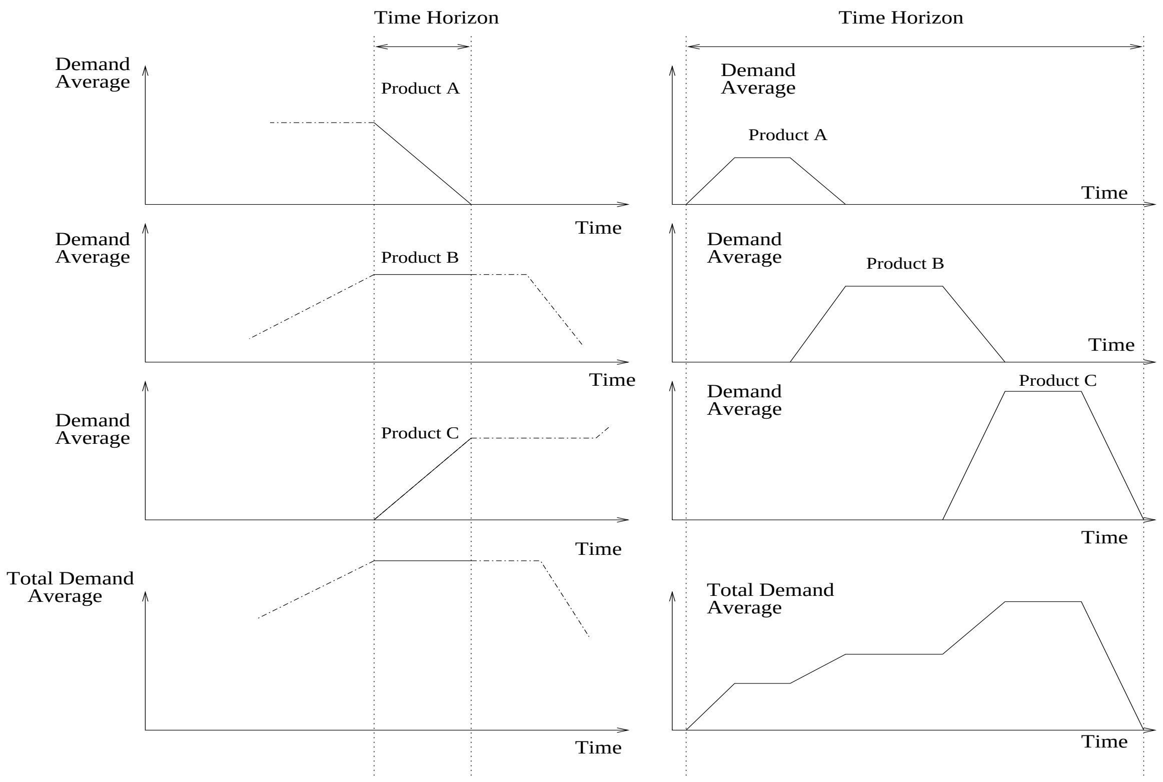


Figure 2.2: Demand Pattern 1 and Demand Pattern 2

In pattern 1, we consider a short time period, with about the duration of the product's fitup ramp, and assume that the demand for product A is decreasing, the demand for product B is steady, and the demand for product C is increasing. It is meant to capture the situation that the technology for these products migrates forward from product A to product C. A special case, in which it is assumed that all demands have normal distribution and their averages are linear in time, is shown in Figure 2.2.

In pattern 2, we consider a longer period that ranges from the time when product A emerges to the time when product C dies out. Demand for each product has a life time. And the new product up-ramp compensates the down-ramp of the old product. The slope is not necessarily the same. If the up-ramp slope always greater than the down-ramp's, we are dealing with increasing total demand. A special case of pattern 2 with linear demand average is also shown in Figure 2.2.

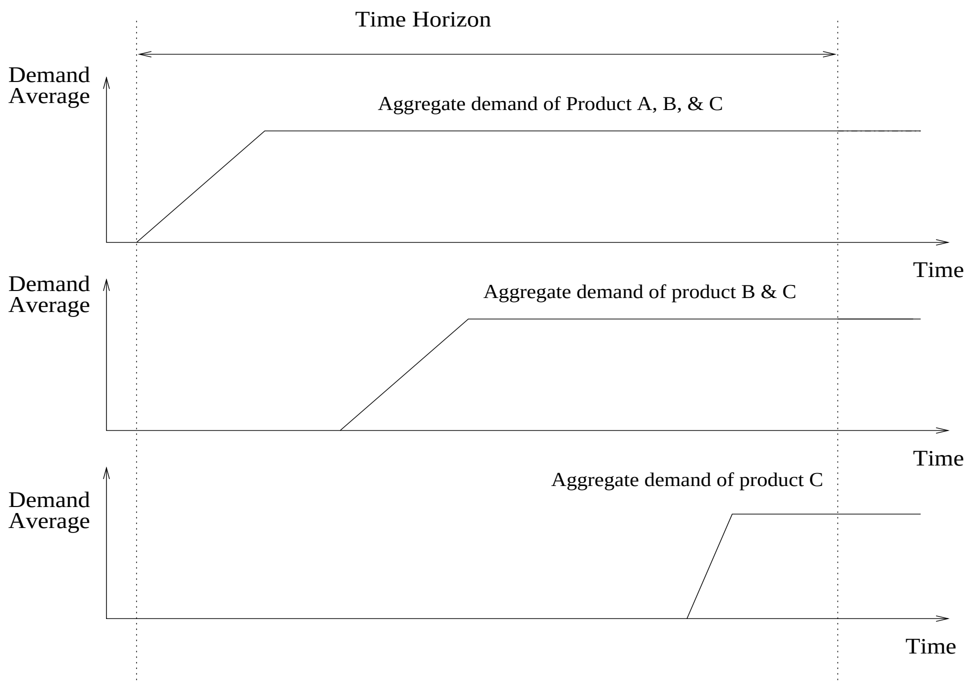


Figure 2.3: Demand Pattern 3

In pattern 3, we consider correlated demand among products. Demands are given as the aggregate demands for products existing in the product family. It is assumed that the aggregate demands first increase, then become stable, and finally die out. It is meant to capture the fab life cycle dynamics. A special case with normal distribution and linear slope is shown in Figure 2.3.

The parameters in the patterns, such as lifetime of products, can be obtained via consulting with industry colleagues.

2.2.4 Simple Example

In order to get a better understanding of how the model would be specified in practice, we provide here a very simple example of a fab producing just two products that each have two operations (litho and etch). The example is meant to be illustrative of the notation and the proposed MDP model, especially with regard to the factory state and capacity expansion/allocation actions. For actual-sized fabs, the notation would all be handled by a computer, as it would be practically infeasible to enumerate the various components of the model. Specifically, the fab will be characterized as follows. There are two products: "A" and "B"; two operations on each: "litho" and "etch", distinguished by product; machines – litho or etch – could be flexible (able to do both A and B operations) or dedicated (only able to do either A or B); operation times, which depend on the product and the machine.

To get a feeling for the computational demands of solving our model, as well as to gain some insight into the structure of optimal expansion and allocation policies we worked through some numerical runs for the simple example to illustrate the application of the model for finding the optimal switching policy between two flexible capacities. The software laboratory **SYSCODE** [59]– which implements several algorithmic schemes for solving dynamic programming models – is used here to obtain the optimal policy.

In these experiments, we also assume the following. During the decision horizon, no machine is purchased, discarded, or sent to reserve, and no maintenance is required. The only actions are to switch flexible machines between different products. For each type of machine (litho or etch), no more than one machine can be switched from one product and/or operation to another in a period. Products A and B are operated in whole unit and half units, respectively. The inventory warehouses for products A and B have capacities of 1 and 0.5 units, respectively. There is a limit on backlogged demand of 1 and 0.5 units for product A and B, respectively. Demand exceeding backlogging limits is lost. The demand process for a given product is independent and identically distributed from period to period, and demand processes are mutually independent between products.

Under the above assumptions, the state vector of our MDP model takes the form $\{X_{(A,1),013},\ X_{(A,2),024},\ I_A,\ I_B\}$, where the first and second components are, respectively, the litho and etch capacities allocated to product A, and the third and fourth components are, respectively, the inventory levels of products A and B. Note that the capacity allocated to product B is simply the remainder of total machine capacity for each tool type (litho or etch), because we have assumed for this simple example that no capacity is ever put into reserve, thus reducing the dimensionality of the state vector from six dimensions to four components. Under our assumptions, the components of the state vector take values in the following sets:

$$\begin{aligned}
X_{(A,1),013} = 2 - X_{(B,3),013} &\in \{0,1,2\},\\
X_{(A,2),024} = 2 - X_{(B,4),024} &\in \{0,1,2\},\\
I_A &\in \{-1,0,1\},\\
I_B &\in \{-0.5,0,0.5\},
\end{aligned}$$

and thus the total number of possible states is 81. We will define a state group as the set of states that have the same capacity allocation $X_{(A,1),013}$ and $X_{(A,2),024}$, i.e., they differ only in their product inventory levels.

The action vector has the form $\{V_{013}^{(A,1),(B,3)},\ V_{024}^{(A,2),(B,4)},\ V_{013}^{(B,3),(A,1)},\ V_{024}^{(B,4),(A,2)}\}$, where the first component is the litho capacity moved from product A to product B, the second component is the etch capacity moved from product A to product B, the third component is the litho capacity moved from product B to product A and the fourth component is the etch capacity moved from product B to product A. Since switching is a two-way interaction, we will have nine possible actions as $A1 = (1,0,1,0)$, $A2 = (1,0,0,0)$, $A3 = (1,0,0,1)$, $A4 = (0,0,1,0)$, $A5 = (0,0,0,0)$, $A6 = (0,0,0,1)$, $A7 = (1,1,1,0)$, $A8 = (0,1,0,0)$, $A9 = (0,1,0,1)$.

A brief summary of our preliminary findings is as follows:

- If one product, say A, is more likely to be short of stock, and it has a higher backlogging penalty, both litho machines and both etch machines would be allocated to produce product A to avoid high cost. To reach this "absorbing" machine allocation state group (2,2), usually the switching actions are taken in one of the following ways:

  - Go to the absorbing state group directly if an admissible action exists.
  - First go to state group (1,1), which allocates one of each type of machine to each product, and then go to the absorbing state group in the next period.
  - First go to the state group that is closest to the absorbing state group, and then go to the absorbing state group.

- Optimal policies that are obtained for various cases match with intuition:
  - If some operating cost is much higher than the others, it is likely that the action leading to this operation will not be performed to avoid high cost.
  - If some switching cost is much higher than others, this switching action is not likely to be performed.
  - For state (1,1,0,0), with one of each type of machine assigned to each product and zero inventory, when the demands are 1 and 0.5 units for products A and B, respectively, the optimal policy is the intuitively obvious one of not switching any capacity, i.e., continuing to run a "balanced" fab with respect to capacity.

2.2.5 "Testbed" Example

For our model, we have provided a simple example and its computational results above. Now we would like to provide a more realistic "testbed" example.

This example originates from the "testbed" in http: // www. eas. asu. edu / masmlab. There are seven data sets with spread sheets for factory, operation, products, tools, and processes etc. We chose data set 4 since it has seven products and it represents a medium-sized fab. However, there are no cost parameters in data set 4. Fortunately, data set 4 is a simple version of ASPEN data in Factory Explorer, which gives us more information, including the cost parameters.

From the original data set, we know that three of the seven products are produced using a common process recipe A with 92 steps, and the other four products are produced using another common process recipe B with 19 steps. Since there are several reentry process steps, we group some reentries into one operation. For example, there are 8 clean steps and they are essentially the same operation. In this way, the 92 process steps in recipe A are aggregated into 31 operations and the 19 process steps in recipe B are aggregated into 11 operation.

So, in this example, the fab is characterized as follows. There are seven products: "A", "B" ,"C", "D", "E" ,"F" and "G"; 137 operations: 31 operations on each of "A" "B" and "E", distinguished by product, and 11 operations on each of "C", "D","F" and "G"; 31 tools: 3 batch tools flexible among all products, where switching-over only happens between ABE batch and CDFG batch, batch 8 tools for product A,B, and E, 12 non-batch tools flexible among A,B, and E, and 8 non-batch tools flexible among all products; operation times, which depend on the product and the tool.

If we discretize every element in the state vector into 10 values, the cardinality of the state space is 10^{175}. If we discretize every element in the control vector into 10 values, the cardinality of the control space is 10^{291}. Hence, the fab-level decision-making MDP problem suffers from both the "curse of dimensionality" and the "curse of modeling".

Obviously, this example cannot be solved by basic MDP algorithms. We need to seek new approach to solve it, such as the simulation-based dynamic programming algorithms.

Note that this problem not only has a large state space as in other problems, but it also has a large control space, unlike many problems solved by simulation-based algorithms. So we need to parameterize the policy or trade off control state complexity with state space complexity. In addition, it is a finite horizon problem; therefore we need to convert it into the stochastic shortest path problem or other form for implementing simulation-based algorithms.

Chapter 3

Simulation-Based Policy Iteration for Unichain Average Cost Problem

In this chapter, we study simulation-based policy iteration algorithms for average cost problems. Some examples of average cost problems are inventory control problems and computer and communication networks, where decisions are made based on throughput rate or average time a job or packet remains in the system [3].

For the average cost problem, results and analyses of policy iteration, as well as simulation-based policy iteration algorithms, depend on the chain structure of the transition matrices of Markov chains generated by stationary policies, whereas those for total cost problems do not. On the basis of the chain structure, MDPs for average cost problems can be classified as recurrent, unichain, or multichain [3], with recurrent the strongest and multichain the weakest. An MDP is *recurrent* if the transition matrix corresponding to every deterministic stationary policy contains a single recurrent class. An MDP is *unichain* if the transition matrix corresponding to every deterministic stationary policy contains one single recurrent class plus a possibly empty set of transient states, whereas it is *multichain* if the transition matrix corresponding to at least one stationary policy consists of two or more recurrent classes. The simulation-based algorithms for the average cost problem studied so far are under either a recurrence assumption or a unichain plus one common recurrent state assumption [3]. Here, we develop a simulation-based policy iteration algorithm for average cost unichain problems and carry out experiments on an inventory control problem.

In the proposed algorithm, we evaluate the average cost first and then evaluate realization factors [12] (the difference between the differential costs) from states to a reference state, instead of the differential costs directly. In this way, our problem is also converted into a stochastic shortest path problem. Using the realization factors gives us the flexibility of choosing the reference state not necessarily the same from iteration to iteration, which leads us to remove the common recurrent state assumption. In addition, we embed the average cost into the stage cost, where the new stage cost is the original stage cost minus the average cost, and apply temporal-difference learning scheme for stochastic shortest path problem in [4]. Thus, the proposed algorithm should be more computationally efficient than the algorithm in [14]. To improve the performance further, transient states are selected as the initial states for sample paths, and the inverse of the visit count is chosen as the step size.

In order to illustrate the application of our algorithm to solving average cost problems under the unichain assumption, we carried out numerical experiments on a single item inventory control problem. We verified that the MDP model of this problem is unichain.

3.1 Simulation-Based Policy Iteration

The basic problem we consider in this chapter is the problem of optimizing a stochastic discrete-time dynamic system with average cost criterion, (refer chapter 2 for detailed MDP model formulation). The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_t : S \to U, \mu_t(i) \in U(i)$ for i and t, the average cost per stage

$$J_\pi(i) = \lim_{T\to\infty} \frac{1}{T} E\left\{\sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i\right\}. \tag{3.1}$$

A stationary policy is an admissible policy of the form $\pi = \{\mu, \mu, \ldots\}$; we denote it by μ_∞.

In an average cost MDP, results and analyses depend on the chain structure corresponding to stationary policies. For simplicity, we consider only unichain MDPs in this chapter.

One method for solving average cost MDP is Policy Iteration, as introduced in Chapter 1. Simulation-based Policy Iteration (SBPI) originates from policy iteration, and the general structure of SBPI is the same as for exact policy iteration. There are two differences, however [4]:

- Given the current stationary policy μ_∞, the corresponding average cost and differential costs are not computed exactly. Instead, an approximate average cost $\hat{\eta}_\mu$ and approximate differential costs $\hat{h}_\mu(i)$ are obtained via simulation. Here, noise from simulation experiments becomes a source of error.

- Once approximate policy evaluation is completed and $\hat{\eta}_\mu$ and $\hat{h}_\mu(i)$ are available, we generate a new policy $\bar{\mu}_\infty$ which is greedy with respect to $\hat{\eta}_\mu$ and $\hat{h}_\mu(i)$, i.e. satisfying (2.10). The greedy policy can be calculated exactly, or it can be approximated which introduces a new source of error.

Here we focus on the policy evaluation step since we use the same policy improvement step as PI. First, we discuss how to approximate the average cost associated with a stationary policy μ_∞ via simulation, which by definition can be written as

$$\eta_\mu = \lim_{T\to\infty} \frac{1}{T} E\left\{\sum_{t=0}^{T-1} g(x_t, \mu(x_t))\right\}. \tag{3.2}$$

Given a stationary policy, if we run one long sample paths with length T, we can obtain an approximation of the average cost via simulation as T becomes large. We may also estimate η_μ iteratively.

Let us discuss how to approximate the differential cost via simulation. Given a stationary policy μ_∞, and assuming the corresponding Markov chain is an aperiodic chain (this can be relaxed), we also have [3]

$$h_\mu(i) = \lim_{T\to\infty} E\left\{\sum_{t=0}^{T-1}(g(x_t,\mu_t(x_t)) - \eta_\mu) \mid x_0 = i\right\}. \tag{3.3}$$

We may refer to the average cost (3.2) as the stationary cost, since it represents cost per stage for a system in steady state. Thus (3.3) allows interpretation of the differential cost as the expected total difference between the total cost and the stationary cost. The differential cost is also known as the bias [3] or the potential [12].

Assume the Markov chain associated with the current stationary policy, starting from state i, encounters state j at time epoch T_{ij} the first time, that is, $T_{ij} = \min\{t : t \geq 0, x_t = j\}$ (note that $T_{ii} = 0$, so that T_{ii} is not the recurrent time of state i); then

$$\begin{aligned}
h_\mu(i) = \quad & \lim_{T\to\infty} E\left\{\sum_{t=0}^{T_{ij}-1}(g(x_t,\mu_t(x_t)) - \eta_\mu) + \sum_{t=T_{ij}}^{T-1}(g(x_t,\mu_t(x_t)) - \eta_\mu) \mid x_0 = i\right\} \\
= \quad & E\left\{\sum_{t=0}^{T_{ij}-1}(g(x_t,\mu_t(x_t)) - \eta_\mu) \mid x_0 = i\right\} \\
& + \lim_{T\to\infty} E\left\{\sum_{t=T_{ij}}^{T-1}(g(x_t,\mu_t(x_t)) - \eta_\mu) \mid x_{T_{ij}} = j\right\} \\
= \quad & b_\mu(i,j) + h_\mu(j),
\end{aligned}$$

where

$$b_\mu(i,j) \equiv E\left\{\sum_{t=0}^{T_{ij}-1}(g(x_t,\mu_t(x_t)) - \eta_\mu) \mid x_0 = i\right\}$$

are called realization factors [12]. This gives us an idea how to approximate $h_\mu(i)$ via simulation.

We know that the unichain Markov chain associated with a stationary policy contains a single recurrent class and a possibly empty set of transient states, so that each state in the recurrent class is reached in a finite number of steps from all initial states with a positive probability. If we choose one such state as reference state r, the differential costs of all other states can be expressed as

$$h_\mu(i) = b_\mu(i,r) + h_\mu(r). \tag{3.4}$$

If the reference state is simply state 0, the state having the differential cost zero, then $h_\mu(i) = b_\mu(i,r)$ since we set $h_\mu(0) = 0$; if not, we can first approximate $b_\mu(0,r)$ via simulation and obtain $h_\mu(i)$ using $h_\mu(i) = b_\mu(i,r) - b_\mu(0,r)$, since $h_\mu(r) = -b_\mu(0,r)$.

So the task of approximating $h_\mu(i)$ reduces to approximating $b_\mu(i,r)$, where

$$b_\mu(i,r) \equiv E\left\{\sum_{t=0}^{T_{ir}-1} (g(x_t, \mu_t(x_t)) - \hat{\eta}_\mu) \mid x_0 = i\right\}$$

for each state i. For brevity, we refer to $b_\mu(i,r)$ as $b_\mu(i)$.

Now the problem is converted into a stochastic shortest path problem, where the new stage cost is $g(x_t, \mu_t(x_t)) - \hat{\eta}_\mu$ and the new termination state is the reference state. Below, we apply the temporal-difference learning scheme for the stochastic shortest path problem [4] to our problem.

The straightforward way is to generate many independent and identically distributed (i.i.d.) sample paths starting from state i and ending with the reference state (used in [14]), and average the corresponding realization factors to obtain an approximation to $b_\mu(i)$. While this can be done separately for each state i, an alternative is to use each sample path to obtain realization factor samples for all states visited by the sample path by considering the realization factor of the trajectory portion that starts at each intermediate state.

To formalize the process, suppose we perform a number of simulation runs, each ending at the reference state r. Consider the lth time the given state i

is encountered, and let $(i, i_1, i_2, \ldots, i_N)$ be the remainder of the corresponding trajectory, where $i_N = r$. Let $c(i,l)$ be the corresponding realization factor of reaching state r, that is, $c(i,l) = (g(i,i_1) - \eta_\mu) + (g(i_1,i_2) - \eta_\mu) + \ldots + (g(i_{N-1}, i_N) - \eta_\mu)$.

Then we can estimate $b_\mu(i)$ by the sample mean

$$\hat{b}_\mu(i) = \frac{1}{L}\sum_{l=1}^{L} c(i,l).$$

We can also iteratively calculate the sample means by using the update formula

$$\hat{b}^l_\mu(i) := \hat{b}^{l-1}_\mu(i) + \gamma_l(c(i,l) - \hat{b}^{l-1}_\mu(i)), l = 1, 2, \ldots, L, \tag{3.5}$$

starting with $\hat{b}^0_\mu(i) = 0$, where stepsize $\gamma_l = 1/l$.

Consider a trajectory $(i_0, i_1, i_2, \ldots, i_N)$ and let m be an integer between 0 and N. We note that this trajectory contains the subtrajectory $(i_m, i_{m+1}, \ldots, i_N)$; this is a sample path with initial state i_m and can therefore be used to update $\hat{b}_\mu(i_m)$ according to equation (3.5). This leads to the following algorithm. At the end of a simulation run that generates the state trajectory $(i_0, i_1, i_2, \ldots, i_N)$, for each $m = 0, 1, \ldots, N-1$, use the formula

$$\hat{b}_\mu(i_m) := \hat{b}_\mu(i_m) + \gamma_{l_m}((g(i_m, i_{m+1}) - \eta_\mu) + \cdots + (g(i_{N-1}, i_N) - \eta_\mu) - \hat{b}_\mu(i_m)). \tag{3.6}$$

Define *temporal differences* (TD) [4]:

$$d_m = (g(i_m, i_{m+1}) - \eta_\mu) + \hat{b}_\mu(i_{m+1}) - \hat{b}_\mu(i_m), \quad m = 0, 1, \ldots, N. \tag{3.7}$$

Note that the mth temporal difference d_m becomes known as soon as the transition from i_m to i_{m+1} is simulated. Then, following the state transition (i_m, i_{m+1}), the cost update formula can be rewritten as follows:

$$\begin{cases} \hat{b}_\mu(i_1) := \hat{b}_\mu(i_1) + \gamma_{l_1} d_m \\ \hat{b}_\mu(i_2) := \hat{b}_\mu(i_2) + \gamma_{l_2} d_m \\ \cdots \\ \hat{b}_\mu(i_m) := \hat{b}_\mu(i_m) + \gamma_{l_m} d_m \end{cases} \tag{3.8}$$

The preceding implementation of the method for evaluating the cost of a policy μ_∞ is known as $TD(1)$ [4]. A generalization of $TD(1)$ is $TD(\lambda)$, $\lambda \in [0, 1]$, which replaces (3.8) by the following:

$$\begin{cases} \hat{b}_\mu(i_1) := \hat{b}_\mu(i_1) + \gamma_{l_1}\lambda^{m-1}d_m \\ \hat{b}_\mu(i_2) := \hat{b}_\mu(i_2) + \gamma_{l_2}\lambda^{m-2}d_m \\ \cdots \\ \hat{b}_\mu(i_m) := \hat{b}_\mu(i_m) + \gamma_{l_m}d_m \end{cases} \tag{3.9}$$

Note that it is possible to encounter state i multiple times within the same sample trajectory. We may use the cost samples of all the subtrajectories to estimate $b_\mu(i)$, and this is called the every-visit method. However, this estimator is biased, since these subtrajectories are dependent. In an alternative method called the first-visit method, we use only the cost sample corresponding to the first visit to that state in a sample trajectory, in which case the corresponding estimator is unbiased. It can be shown that the bias of the every-visit method converges to zero, but there is strong evidence indicating that its mean squared error eventually becomes larger than the mean squared error of the first-visit method [4].

There are also some important practical issues in implementing our proposed algorithm. First, we choose transient states as the initial states, because transient states will be rarely visited starting from a recurrent state, whereas recurrent states are reached from any transient state. Second, we choose the inverse of the visit count to a state i as the stepsize γ_{l_i}. The visit count to a state is the number of times the state is encountered in the sample path.

The SBPI algorithm is as follows:

1. Set $k = 0$, select an arbitrary stationary policy μ_∞^k.

2. **Policy evaluation**:

(a) Estimate the average cost $\hat{\eta}^k$:

(i) Generate one sample trajectory with length T_A and arbitrary initial state (subscript A indicates the "Average Cost");

(ii) Set the initial estimate $\hat{\eta}^k := 0$;

(iii) For the lth sample trajectory $(i_0, i_1, \ldots, i_{T_A})$,

Following the transition (i_t, i_{t+1}), update $\hat{\eta}^k$ by

$\hat{\eta}^k := (1 - 1/t)\ \hat{\eta}^k + 1/t\ g(i_t, \mu^k(i_t))$;

(b) Estimate the differential cost $\hat{h}^k(i)$:

(i) Determine recurrent states and transient states by Fox-Landi algorithm;

(ii) Choose one recurrent state as the reference state r;

(iii) Generate L_D sample trajectories (subscript D indicates the "Differential Cost"),

each starting with a transient state and ending with the first encountered reference

state r;

(iv) For the lth sample trajectory $(i_0, i_1, \ldots, r)$ and $l = 1, \ldots, L_D$,

Following the transition (i_m, i_{m+1}), update $b(i_s)$, for $s = 1, \ldots, m$, by

$\hat{b}(i_s) := \hat{b}(i_s) + \gamma_{is}\lambda^{m-s} d_m$,

where d_m and λ are defined before and γ_{is} is a stepsize;

(v) If $r = 0$, $\hat{h}^k(i) := \hat{b}(i)$; otherwise, $\hat{h}^k(i) := \hat{b}(i) - \hat{b}(0)$.

3. **Policy improvement**:

   Choose a new policy μ_∞^{k+1} that is greedy with respect to the approximate average cost and differential costs, i.e, according to (2.10).

4. If $\mu_\infty^{k+1} = \mu_\infty^k$, then stop and set $\mu_\infty^* = \mu_\infty^k$; otherwise increment k by 1 and return to step 2.

Next we apply our proposed algorithm to an inventory control problem.

3.2 Inventory Control Problem

We consider a single product inventory control problem, where the system state is the inventory level and the stages are the review periods. In a given state, actions correspond to the amount of stock to order. Transition probabilities depend on the quantity ordered and the random demand during the previous period. A decision rule specifies the quantity to be ordered as a function of the stock on hand at the time of review, and a policy consists of a sequence of such re-stocking functions. The objective is to seek a reordering policy that minimizes long-run average ordering cost, inventory cost and shortage cost.

Now let us look at an example. Each week, a manager of a warehouse checks current inventory (stock on hand) of a single product. Based on this information, he decides whether or not to order additional stock from a supplier. In doing so, he is faced with a tradeoff between the costs associated with keeping inventory and the penalties associated with being unable to satisfy customer demand. The manager's objective is to minimize the average cost over the decision-making horizon. [3].

We formulate a basic model for the inventory control problem under the following assumptions:

- The decision to order additional stock is made at the beginning of each week and delivery occurs instantaneously, i.e., no lead-time.
- Demand for the product arrives throughout the week but all orders are filled on the last day of the week.
- If demand exceeds inventory, there is limited backlogging B of unfilled orders and if more than that, the extra demand is lost.

- The cost and the demand distribution do not vary from week to week.
- The product is sold only in whole units.
- The warehouse has capacity of M units.

Let
x_t denote the inventory on hand at the beginning of week t,
u_t the number of units ordered by the inventory manager in week t,
D_t the random demand in week t.
We assume that the demand has a known probability distribution. The inventory at week $t+1$, x_{t+1}, is related to x_t, the inventory at week t, through the system equation

$$x_{t+1} = x_t + u_t - D_t, \tag{3.10}$$

but we restrict the back-logging limit to be B (i.e., $-B \leq x_t \leq M$).
Define the cost at each week as follows:

$$g(x_t, u_t) = C \cdot I(u_t > 0) + c_o \cdot u_t + c_h \cdot max(0, x_t) + c_p \cdot max(0, -x_t), \tag{3.11}$$

where
C = setup cost for placing a order,
c_o = per unit ordering cost,
c_h = holding cost per week per unit of inventory,
c_p = shortage cost per week per unit of inventory,
$I\{\cdot\}$ = the indicator function of the set $\{\cdot\}$.

The MDP formulation is defined as follows:
states:

$$i = x_t + B, \quad i \in \mathbf{S} = \{0, 1, 2, \ldots, B + M\};$$

actions:

$$u \in U(i) = \{0, 1, 2, \ldots, M - (i - B)\};$$

transition probabilities:

$$p_{ij}(u) = \begin{cases} p(D = i + u - j) & if\ j \neq 0; \\ p(D > i + u) & if\ j = 0; \end{cases} \tag{3.12}$$

where D is the generic random variable for D_t.

expected costs:

$$g(i, u) = C \cdot I(u > 0) + c_o \cdot u + c_h \cdot \max(0, i - B) + c_p \cdot \max(0, -(i - B)). \tag{3.13}$$

Note that the state is defined as the inventory level offset by B, since the state is indexed from 0 in our MDP definition.

Now, we provide the details of the application of SBPI to the inventory problem. We first discuss why it is necessary to check whether our MDP model is unichain and how to check it.

There are two reasons why we need to check the unichain assumption. First, if an MDP is not unichain, i.e. multichain, the average cost of a stationary policy is possibly nonconstant, which means we could use neither policy iteration nor SBPI under the unichain assumption. A more complicated policy iteration algorithm for the multichain case is given in [3]. Second, the behavior of the policy iteration algorithm for a unichain MDP involves a recurrent class plus transient states, and the recurrent class changes from iteration to iteration, so the convergence of policy iteration for a unichain MDP depends on differential costs of both the recurrent class and the transient states. The learning schemes provided in [12] [13] only guarantee that the differential cost for states in the recurrent class converges to the true value, in which case, the policy improvement step may not lead to a better policy. Thus, a scheme that guarantees that the estimated differential cost for both recurrent states and transient states converges to the true value is necessary under the unichain assumption. In our algorithm, it is done by running many sample paths with the initial states being transient states, which guarantees the transient states are visited frequently enough.

The standard procedure to verify whether an MDP is unichain is the Fox-Landi algorithm [3]. The basic idea is to construct paths through the state space using the transition probability matrix, producing a labeling and grouping of all states. It also provides a way to find the recurrent classes and transient states. However, this algorithm requires $O(|\mathbf{S}|^2)$ comparisons. An alternative method is: if we can prove an MDP is unichain by other means, a simple heuristic for finding the recurrent states and transient states is to run several sample paths in advance and calculate how often the states have been visited. Those states that are seldom visited are likely to be transient states and are chosen as initial states in the implementation of our algorithm. Furthermore, state 0 is a recurrent state under all policies, so we chose it as the reference state. A necessary and sufficient condition under which the MDP model is unichain is given in the following.

Lemma 3.2.1 *With the transition probabilities defined in (3.12) and $D_{\max}$ the maximum value that the random variable D can take, the MDP is unichain if and only if $D_{\max} \geq B + M$.*

Proof: For $D_{\max} \geq B + M$, the probability from any state i to state 0 with any action u, $p_{i0}(u)$, is positive, so state 0 is recurrent and any other state can reach this recurrent state 0 in a single step with a positive probability, which means that each of the other states is either recurrent in the same recurrent class as state 0 or a transient state. Therefore, the MDP is unichain.

If $D_{\max} < B + M$, then we show that the MDP is multichain by constructing a policy under which there are two recurrent classes. Take the following policy:

$$\mu(i) = \begin{cases} B + M - i, & i \geq B + M - D_{\max}; \\ 0, & \text{otherwise}; \end{cases} \tag{3.14}$$

Then some nonempty subset of $\{0, \ \ldots, \ B + M - D_{\max} - 1\}$ is recurrent, and some nonempty subset of $\{B + M - D_{\max}, \ \ldots, \ B + M\}$ is recurrent, and these two sets themselves do not communicate. □

The proof of the lemma shows that in the unichain case the state 0 is reachable (in one step) from all other states for all policies, so it must be part of the recurrent set for all policies, giving the following result:

Corollary: For the inventory control problem, the MDP is unichain if and only if it is unichain with a common recurrent state (namely state 0).

However, if we limit policies to what one might called "reasonable" monotone ordering policies, $\mu(i) \geq \mu(i+1)$, i.e., you never order less with a lower inventory level, we can find unichain MDPs in which there is no common recurrent state under all policies. Let us look at an example with $D \in \{0,1\}$ and $\mathbf{S} = \{0,1,2\}$. If policies were unrestricted, then by the lemma, the resulting MDP would be multichain, since $D_{max} < 2$. However, under the policy restriction above, the resulting MDP is in fact unichain, with the allowable policies given by the following $(\mu(0), \mu(1))$ pairs, with $\mu(2) = 0$ due to the warehouse limit: (0,0), (1,0), (1,1), (2,0), (2,1), which give respective recurrent sets $\{0\}, \{0,1\}, \{1,2\}, \{0,1,2\}, \{1,2\}$, i.e., state 0 is transient under two policies, one of which is the last policy, which is of the (s, S) form with $s = 0$ and $S = 1$. An (s, S) type policy means that an order is placed only when inventory level falls below the level s, and that the quantity of the order is placed "up to S". The policy that was eliminated by the condition was (0,1), which would have given two recurrent classes $\{0\}$ and $\{1,2\}$.

For the inventory control problem, we performed numerical experiments using SBPI. In our experiments, we consider an example with: $B = 50$; $M = 100$; setup cost $C = 100$, unit cost $c_o = 1$, holding cost $c_h = 1$, and shortage cost c_p = 10. The demands $\{D_t\}$ are i.i.d. with each sample generated as follows [18]:

- sample from a exponential distribution with a given mean 20;
- round off this value to the closest integer;

- restrict the resulting value in a given range from 0 to 250.

Since $D_{\max} = 250 > 150 = B + M$, by the lemma the MDP for this example is unichain.

This problem is not too large, so it can be solved exactly through policy iteration. From a given arbitrary policy, shown in Fig 3.1a, five policy iterations were required to determine the optimal policy (policy for iteration 5 in Fig 3.1b). The optimal average cost is 99.87. The graphical representation makes it obvious that the optimal policy has some kind of structure, in particular, a (s, S) type policy. More interestingly, we found that all policies following the initial policy are of the (s, S) type in this case. Fig 3.1c shows the corresponding s and S for iteration 2 to 5. An open problem is whether or not this finding is true in general. The average costs for each iteration are shown in Fig 3.1d.

Implementation of SBPI requires choosing tunable parameters L_A, T_A, L_D, and λ. Larger L_A, T_A, and L_D provide better accuracy at the cost of additional computation. In the following experiments, we used $L_A = 1$, $T_A = 100000$, $L_D = 10000$ and $\lambda = 1$.

Because SBPI requires the policy evaluation step to provide a good estimate of the average cost and differential cost before implementing the policy improvement step, large T_A and L_D are necessary. This is also related to the visit count of a state, which is the number of times the state is encountered in the sample path. This is even more important in unichain problem, because of the presence of transient states. To see this, Fig 3.2 shows results for an (s, S) type policy where $s = 10$ and S =45, and $T_A = 10^5$, $L_D = 10^4$, and $\lambda = 1$. In this case, the transient states are the states from 95 to 151. If we look at the visit count shown in Fig 3.2b, we can see clearly that these states are less frequently visited, and the differential costs have larger error, as shown in Fig 3.2c. An updated policy is shown in Fig 3.2d.

The stopping criterion is based on the estimated cost difference between two

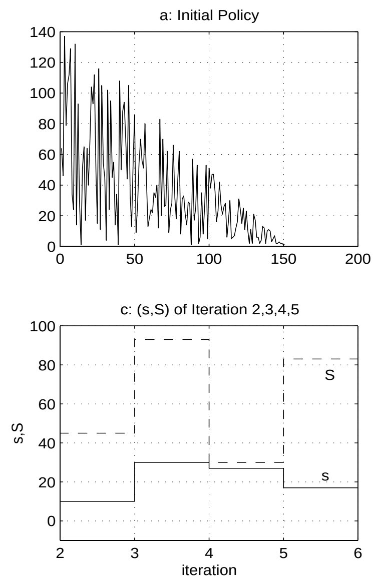


Figure 3.1: PI Policies and Average Cost

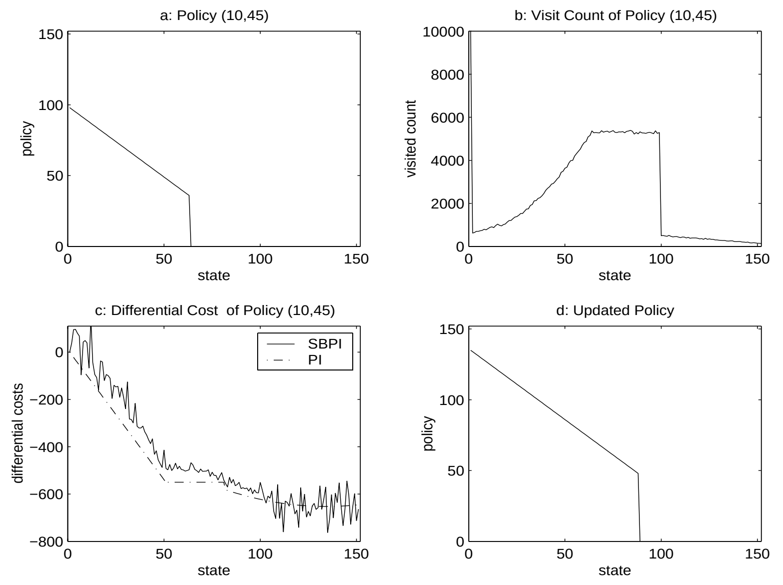


Figure 3.2: SBPI Policy Evaluation for Differential Costs

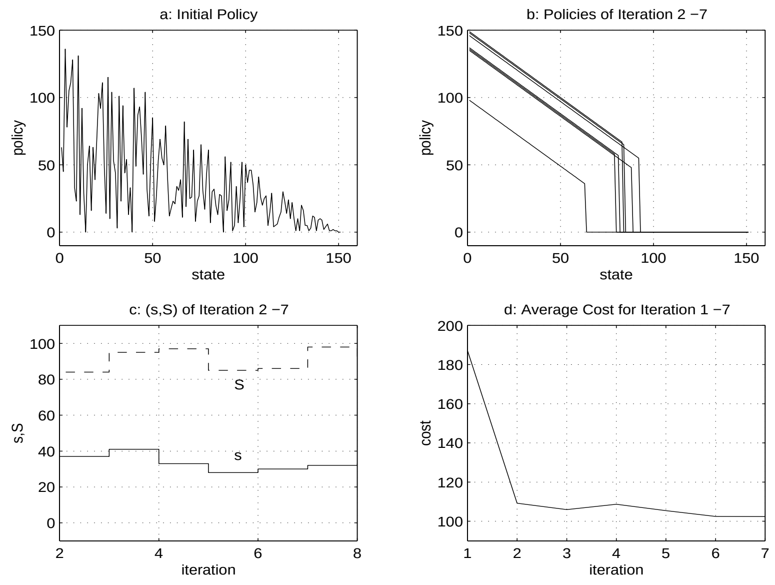


Figure 3.3: SBPI Policies and Average Cost ($T_A = 10^5 \quad L_D = 10^4$)

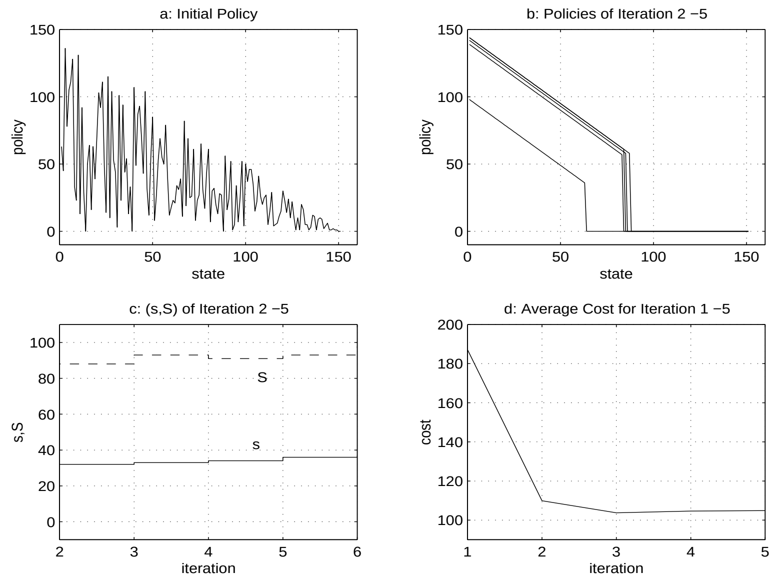


Figure 3.4: SBPI Policies and Average Cost ($T_A = 10^5 \quad L_D = 10^5$)

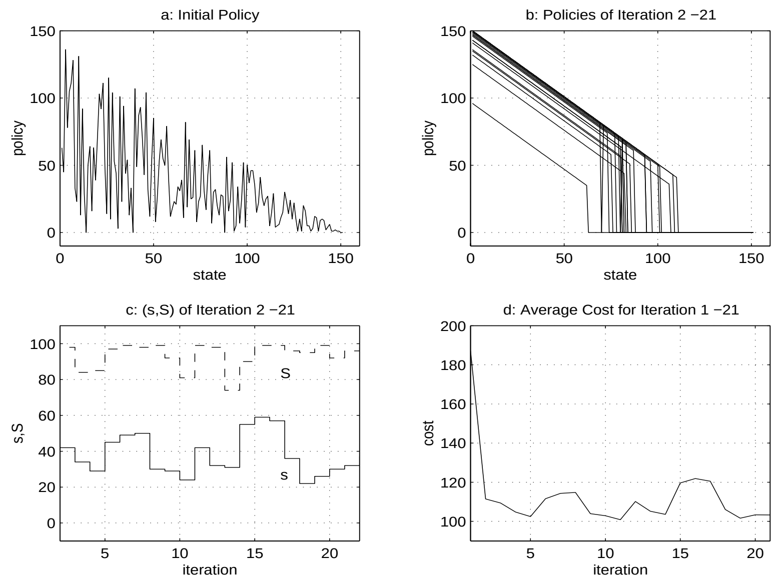


Figure 3.5: SBPI Policies and Average Cost ($T_A = 10^5 \quad L_D = 10^3$)

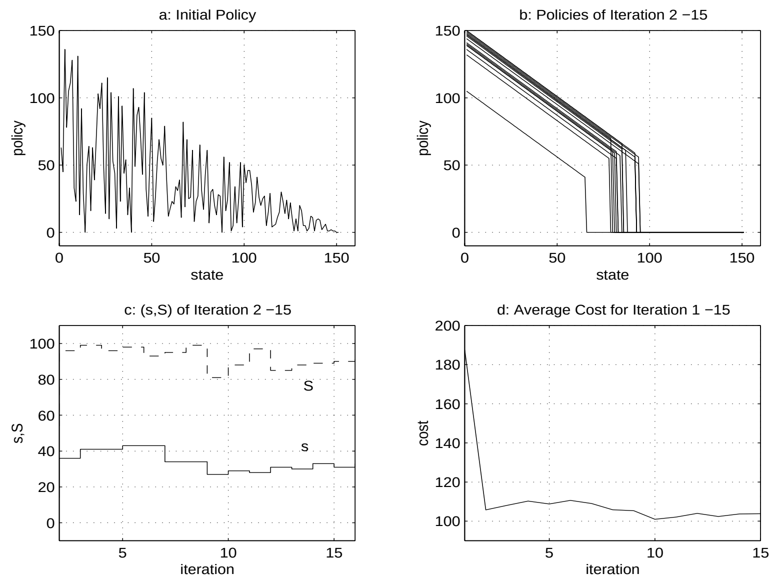


Figure 3.6: SBPI Policies and Average Cost ($T_A = 10^4 \quad L_D = 10^4$)

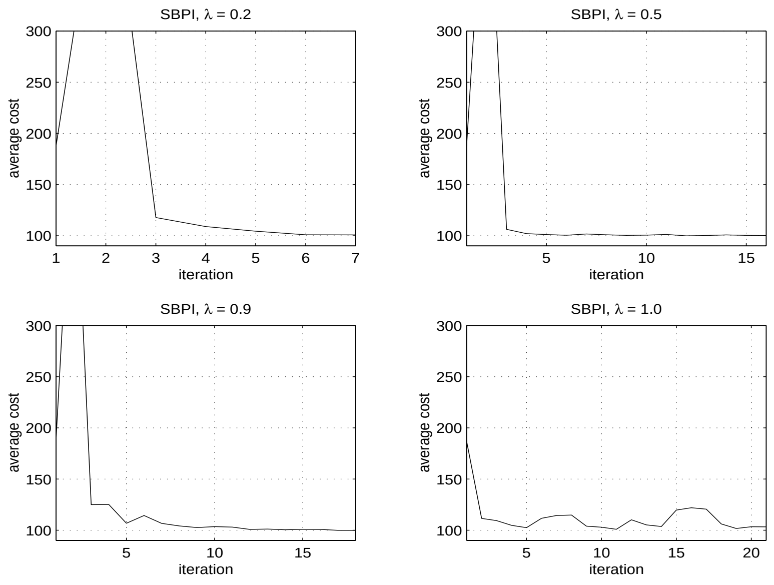


Figure 3.7: SBPI Policies and Average Cost for Different λ

consecutive iterations. The algorithm stops when the difference is less than a threshold, which was 0.25 in our experiments. A more accurate stopping criterion could be based on the difference between consecutive policies, but this requires the definition of a suitable metric on the policy space.

First we consider the SBPI implementation with TD(1), i.e. $\lambda = 1$. In order to determine the setting of the tunable parameters T_A and L_D, we ran a policy evaluation step before the SBPI algorithm is run, and compared the variance and computational time. From the experiments, we determined that $T_A = 10^5$ and $L_D = 10^4$ gave suitably accurate results. Fig 3.3 shows the result of the SBPI implementation with TD(1) for the same case shown for PI in Fig 3.1. Note that SBPI takes 7 iterations to achieve the near optimal policy, whereas PI takes only 5 iterations to achieve the optimal policy (refer to Fig 3.1); however, the average costs associated with them are very close: 102.45 and 99.87. As with PI, we also found that all SBPI policies iterated after the initial policy are (s,S) type. In terms of computational time, it takes about 15 minutes for SBPI in a Sun Ultra 10 to reach the near optimal policy.

Next, we investigated the sensitivity of the algorithm to the various tunable parameters T_A, L_D and λ. If T_A or L_D is chosen to be larger, SBPI would take fewer iterations but longer time overall to reach a near optimal policy. For example: if $L_D = 10^5$ with other parameters unchanged, SBPI takes only 5 iterations but about 2 hours to reach a near optimal policy, starting from the same policy (see Fig 3.4). Compared with the PI implementation, the number of policy iterations is the same, but the average cost for the near optimal policy, 104.63, is even worse than the previous case.

If T_A or L_D is chosen to be smaller, the time to obtain a near optimal policy may be shorten. If $L_D = 10^3$ with other parameters unchanged, it takes 21 iterations and about 7 minutes to reach the near optimal policy, with an average cost of 103.29 (refer to Fig 3.5). If $T_A = 10^4$ with other parameters unchanged, it takes 15 iterations and about 13 minutes to reach the near optimal policy, with an average cost of 103.82(refer to Fig 3.6). Using values of T_A and L_D that are too small, however, may lead to oscillatory behavior, because the evaluated average cost and differential cost in the policy evaluation step are not accurate enough.

The value of λ is related to that of L_D, in that smaller λ allows L_D to be tuned smaller. Fig 3.7 shows a comparison of different λ with the same $L_D = 10^3$. The result shows that generally the smaller the λ,the better the performance. However, if λ is too small, the performance deteriorates drastically when the average cost goes to 500 for each iteration.

The computational experiments led to the following conclusions for the numerical example: Our SBPI algorithm finds a near optimal policy in a few iterations. For the case we considered, the updated policy becomes an (s, S) type policy within one iteration, starting from any randomly picked policy. The performance of SBPI depends on sufficiently good estimates of the differential costs based on the simulated sample paths, which in turn can be sensitive to the choice of the tunable parameters, trading off computational time for accuracy.

3.3 Discussion

In this chapter, we develop a simulation-based policy iteration algorithm for average cost unichain MDPs. In this algorithm, 1) the problem is converted to a stochastic shortest path problem and a reference state can be chosen as any state recurrent under the current policy, in which case the reference state is not necessarily the same from iteration to iteration; 2) the realization factors to a reference state are evaluated, instead of the differential cost being evaluated directly, by a temporal-difference earning scheme; 3) transient states are selected as the initial states for sample paths, and the inverse of the visiting count is chosen as the step size to improve the performance.

There are also other simulatioon-based algorithms for the inventory problem, such as those based on Perturbation Analysis [60], or Simultaneous Perturbation Stochastic Approximation, since the optimal policy for this problem is known to have the (s,S) type structure.

Chapter 4

Simulation-Based Gradient Algorithms for Weighted Cost Problems

In this chapter, we develop a two-timescale simulation-based gradient algorithm based on perturbation analysis for weighted cost problems, and compare it with the simulation-based gradient algorithms proposed by Marbach [20]. In our context, the weighted cost problem is an infinite horizon MDP problem with weighted expected total cost criterion, where the expected total cost is defined in the same way as in the stochastic shortest path problem. This problem is first discussed by Marbach [20] (where it is referred to as the weighted reward-to-go problem), and it is meant to capture the situation where the decision maker wants to minimize the cost-to-go, given a known probability distribution on the initial state. Our fab-level decision making problem is a special case of the weighted cost problem, where the initial state is known. It has been proved that the policy that minimizes the expected total cost from all states in the stochastic shortest path problem also minimizes the weighted expected total cost in the corresponding weighted cost problem. Hence, policy iteration, value iteration, and all simulation-based dynamic programming algorithms for the stochastic shortest path problem can be transferred to the weighted cost problem. However, our fab-level decision-making problem has a very large control space. One way to deal with a large control space is to parameterize the policy, but there are few simulation-based algorithms with parameterized policies for the stochastic shortest path problem, due to the difficulty that expected costs from all states need to be minimized with respect to the parameters. Fortunately, if a policy can be parameterized, the weighted cost problem can be transformed to a problem of finding the optimal parameters minimizing only the weighted expected cost, and thus can be solved by stochastic approximation algorithms based on gradient estimation. Hence, we want to study simulation-based algorithms for the weighted cost problem.

For the weighted cost problem, Marbach [20] derived one form of the gradient of the cost function with respect to the parameters, and developed two simulation-based gradient algorithms. After correcting some errors in the algorithms, we compare the corrected algorithms with our two-timescale algorithm. We carry out numerical experiments on a parking example.

4.1 Weighted Cost Problems

Consider the situation where the decision maker wants to minimize the mean cost over initial states, knowing that the initial state x_0 is equal to a specific state $i \in S$ with probability ξ_i, where S is the state space. If we denote the cost to go from an initial state x_0 with policy π as $J_\pi(x_0)$, the mean cost to go over initial states with policy π is $E[J_\pi(x_0)] = \sum_{i \in S} \xi_i J_\pi(i)$. In the sense that ξ_i acts like a kind of weight, we call the mean cost to go over initial states a *weighted cost to go* and denote it by χ_π. The corresponding problem of minimizing χ_π is called *the weighted cost problem.* Our fab-level decision making problem is a special case of the weighted cost problem, where the initial state is known.

Here, we assume that the weighted cost problem has an infinite horizon. We also assume that there exists a termination cost-free state i^* in the system and the system remains there at no further cost once it reaches that state, as in the stochastic shortest path problem. The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_t : S \to U, \mu_t(i) \in U(i)$ for i and t, the weighted expected total cost,

$$\chi_\pi = \sum_{i \in S} \xi_i J_\pi(i), \tag{4.1}$$

where

$$\begin{aligned}
&\xi_i \geq 0, \quad \textstyle\sum_{i \in S} \xi_i = 1, \\
&J_\pi(i) = E\left\{\textstyle\sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i\right\}, \\
&T = \min\{t > 0 \mid x_t = i^*\},
\end{aligned}$$

$U(i)$ is the set of allowable actions in state i, $p_{ij}(u)$ is the one-step transition probability from state i to j with action u, and $g(i, u)$ is the corresponding stage cost.

If π is a stationary policy with the form $\pi = \{\mu, \mu, \ldots\}$, the corresponding weighted cost and expected total cost from state i are denoted as χ_μ and $J_\mu(i)$, respectively.

If a randomized decision rule μ is considered, which specifies a probability distribution $q_{\mu(i)}(u)$ on the set of actions, the stage cost and transition probabilities become

$$\begin{aligned}
g(i, \mu(i)) &= \textstyle\sum_{u \in U(i)} g(i, u) q_{\mu(i)}(u), \\
p_{ij}(\mu(i)) &= \textstyle\sum_{u \in U(i)} p_{ij}(u) q_{\mu(i)}(u).
\end{aligned}$$

In order to solve this problem, we need to derive its optimality equations. Note that the policy minimizing $J_\mu(i)$ in the corresponding stochastic shortest path problem also minimizes χ_μ in the weighted cost problem, because $\chi^* = \sum \xi_i J^*(i) \leq \sum \xi_i J_\pi(i) = \chi_\pi$, as $J^*(i) \leq J_\pi(i)$. Hence, policy iteration, value iteration, and all simulation-based dynamic programming algorithms for the stochastic shortest path problem can be used for the weighted cost problem.

If the control space of an MDP is very large, the search space would accordingly be large, which requires very long computation time. One way to deal with this difficulty is to parameterize the policy.

If we parameterize the policy by a parameter vector $\theta \in R^p$, the parameterized stationary policy is $\pi(\theta) = \{\mu, \mu, \ldots\}$, with $\mu : S \times R^p \to U$, and the corresponding decision rule is $\mu(i, \theta)$, with its dependence on θ assumed known. For example, in an inventory control problem, an (s, S) policy is specified as follows:

$$\mu(i, \theta) = \begin{cases} S - i & if\ i < s; \\ 0 & if\ i \geq s; \end{cases} \tag{4.2}$$

where $\theta = (\theta_1, \theta_2) \equiv (s, S)$. The (s, S) policy means that an order is placed only when the inventory level falls below the level s, and that the quantity of the order is placed "up to S".

A more general example is described in [20], where the policy is a randomized policy. Let $r_u(i, \theta)$, be a parameterized function that maps states and actions to real numbers, and consider the case where the decision maker chooses at state i action u with probability

$$q_{\mu(i,\theta)}(u) = \frac{\exp(r_u(i, \theta))}{\sum_{v \in U} \exp(r_v(i, \theta))}. \tag{4.3}$$

Note that $q_{\mu(i,\theta)}(u) \geq 0$ and $\sum_{u \in U(i)} q_{\mu(i,\theta)}(u) = 1$. The function $r_u(i, \theta)$ can be thought as a "likelihood" function associated with state i and action u, in the sense that the action with largest $r_u(i, \theta)$ is most likely to be chosen at state i.

Under a randomized policy $\mu(i, \theta)$ characterized by $q_{\mu(i,\theta)}(u)$, the transition probabilities are denoted by $p_{ij}(\theta)$, the one-stage cost at state i is denoted by $g_i(\theta)$ and

$$\begin{aligned} p_{ij}(\theta) &= \textstyle\sum_{u \in U(i)} p_{ij}(u) q_{\mu(i,\theta)}(u), \\ g_i(\theta) &= \textstyle\sum_{u \in U(i)} g(i, u) q_{\mu(i,\theta)}(u). \end{aligned} \tag{4.4}$$

Hence, under policy $\mu(i, \theta)$, the Markov Decision Process we defined earlier degenerates to a Markov process defined by transition probabilities $p_{ij}(\theta)$ and one-stage costs $g_i(\theta)$, which we call a Markov Cost Process (MCP) depending on θ. The termination state is i^*, with $p_{i^* i^*}(\theta) = 1$ and $g_{i^*}(\theta) = 0$.

Furthermore, the original problem of finding an optimal policy for an MDP becomes a problem of finding the optimal θ minimizing $\chi(\theta)$, where

$$\chi(\theta) = \sum_{i \in S} \xi_i J_i(\theta),$$

$$J_i(\theta) := J_{\pi(\theta)}(i) = E\left\{ \sum_{t=0}^{T-1} g_{i_t}(\theta) \mid i_0 = i \right\}, \tag{4.5}$$

and $T = \min\{t > 0 \mid i_t = i^*\}$.

In order for gradient estimation techniques to be applicable, the following assumptions on $p_{ij}(\theta)$ and $g_i(\theta)$ are necessary.

Assumption 4.1.1 (MCP Parameterization) *For all $i, j \in S$, the transition probability $p_{ij}(\cdot)$, and the stage cost $g_i(\cdot)$ are bounded, twice differentiable, and have bounded first and second derivatives. Furthermore, for all states $i, j \in S$, we have*

$$\nabla p_{ij}(\theta) = p_{ij}(\theta) f_{ij}(\theta),$$

where the function $f_{ij}(\theta)$ is bounded and differentiable, with bounded first derivative.

Let $P(\theta)$ be the transition matrix with entries $p_{ij}(\theta)$, let $\mathcal{P} = \{P(\theta) \mid \theta \in \mathcal{R}^K\}$ be the set of all possible transition matrices, and let $\overline{\mathcal{P}}$ be its closure. It can be proved that every element $P \in \overline{\mathcal{P}}$ is a stochastic matrix (see Lemma 1 in [20]).

Assumption 4.1.2 (MCP Termination) *There exists a state $i^* \in S$, such that, for every parameter vector $\theta \in \mathcal{R}^K$, we have*

$$g_{i^*}(\theta) = 0$$

and

$$p_{i^* i^*}(\theta) = 1$$

and for every state $i \in S$ and every transition matrix $P(\cdot) \in \overline{\mathcal{P}}$, we have $p_{ii^*}^N > 0$, where N is the number of states in the state space S.

4.2 Two-Timescale Simulation-Based Gradient Algorithm

In this section, we developed a two-timescale simulation-based gradient algorithm for weighted cost problems using gradient estimation techniques.

We begin with defining a new Markov cost process with transition probabilities

$$p_{\xi,ij}(\theta) = \begin{cases} p_{ij}(\theta) & \text{if } i \neq i^*; \\ \xi_j & \text{if } i = i^*, \end{cases}$$

and one-stage costs $g_{\xi,i}(\theta) = g_i(\theta)$, as in [20]. Note that the new Markov cost process is a renewal process and $\chi(\theta)$ is equal to the expected cumulative cost over a regenerative cycle. By using renewal theory, the following expressions for $\chi(\theta)$ and its gradient can be obtained [20]:

$$\chi(\theta) = E_{\xi,\theta}[T] \sum_{i \in S} \pi_{\xi,i}(\theta) g_{\xi,i}(\theta), \tag{4.6}$$

$$\nabla\chi(\theta) = E_{\xi,\theta}[T] \sum_{i \in S} (\nabla\pi_{\xi,i}(\theta) g_{\xi,i}(\theta) + \pi_{\xi,i}(\theta) \nabla g_{\xi,i}(\theta)), \tag{4.7}$$

where $E_{\xi,\theta}[T]$ is the mean recurrence time and $\pi_{\xi,i}(\theta)$ is the steady state probability distribution of being in state $i \in S$. However, for problems with a large state space, it is generally infeasible to compute this gradient exactly, since this requires the computation, for every $i \in S$, of the steady state probability $\pi_{\xi,i}(\theta)$ and its gradient $\nabla\pi_{\xi,i}(\theta)$. One method to deal with this difficulty is to develop a simulation-based estimator of $\nabla\chi(\theta)$ by techniques such as perturbation analysis.

Before we present a particular gradient estimator for $\nabla\chi(\theta)$, we discuss why we consider two-timescale simulation-based algorithms and how we develop such algorithms.

Let $(i_1, i_2, \ldots)$ be a sample path of the renewal process, t_m be the mth visit to the recurrent state i^*, and $i_{t_m}, i_{t_m+1}, \ldots, i_{t_{m+1}-1}$ be the mth regenerative cycle.

Suppose $\hat{F}(\theta)$ is an unbiased estimator of $\nabla\chi(\theta)$, a simulation-based gradient algorithm based on this estimate is:

$$\theta_{m+1} = \theta_m + \alpha_m \hat{F}_m(\theta_m). \tag{4.8}$$

where $\hat{F}(\theta_m)$ is a sample estimate of $\chi(\theta_m)$ and the step sizes α_m are deterministic, nonnegative, and satisfying $\sum_{m=1}^{\infty} \alpha_m = \infty$ and $\sum_{m=1}^{\infty} \alpha_m^2 < \infty$. This algorithm, which we call the *regenerative-update simulation-based algorithm*, updates at visits to the regenerative state i^*.

And if $\hat{F}_m(\theta_m)$ can be reformulated as:

$$\hat{F}_m(\theta) = \sum_{k=t_m}^{t_{m+1}-1} \hat{R}_k(\theta),$$

another simulation-based gradient algorithm is given by:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{R}_k(\theta_k), \tag{4.9}$$

where the step sizes α_k satisfy the same conditions as before. We call such an algorithm the *every-update simulation-based gradient algorithm.*

Note that the length of each regenerative cycle is unknown, and may be very long, which can lead to infrequent updates of θ for the regenerative-update simulation-based algorithm. On the other hand, if θ is updated at each time epoch, as in the every-update simulation-based gradient algorithm, θ may change too frequently which may not be allowed in a real system. Hence, we propose a general two-timescale simulation-based gradient algorithm as follows:

$$\theta_{l+1} = \theta_l + \sum_{k=n_l}^{n_{l+1}-1} \alpha_k \hat{R}_k(\theta_l), \tag{4.10}$$

where

$$n_{l+1} = \min\{j > n_l | \sum_{k=n_l}^{j-1} \alpha_k \geq \beta_l\}, \tag{4.11}$$

and we assume the following on the two-timescale step sizes α_k and β_l:

Assumption 4.2.1 *The step sizes α_k and β_k are deterministic, nonnegative and satisfy*

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty,$$
$$\sum_{k=1}^{\infty} \beta_k = \infty, \quad \sum_{k=1}^{\infty} \beta_k^2 < \infty,$$
$$\alpha_k/\alpha_{k+1} \to 1, \beta_k/\beta_{k+1} \to 1, \beta_k = o(\alpha_k).$$

Next, we discuss a particular gradient estimator for $\chi(\theta)$.

4.2.1 A Modified Gradient Estimator and its Decomposition

In this section, we review Marbach's results [20] on the gradient of $\chi(\theta)$ and the gradient estimator, and present a modified gradient estimator and a decomposition of this modified gradient estimator.

By using perturbation analysis, Marbach [20] obtained the following expressions for $\nabla\chi(\theta)$, the gradient of $\chi(\theta)$:

$$\nabla\chi(\theta) = E_{\xi,\theta}[T] \sum_{i\in S} \pi_{\xi,i}(\theta)(\nabla g_i(\theta) + \sum_{j\in S} \nabla p_{ij}(\theta) J_j(\theta)), \tag{4.12}$$

based on (4.7).

Note that $\nabla\chi(\theta)$ in Eq. (4.12) can be rewritten as [20]:

$$\nabla\chi(\theta) = E_{\xi,\theta}[T] \sum_{i\in S} \pi_{\xi,i}(\theta) \left(\nabla g_i(\theta) + \sum_{j\in S_i(\theta)} p_{ij}(\theta) \left(\frac{\nabla p_{ij}(\theta)}{p_{ij}(\theta)} J_j(\theta) \right) \right), \tag{4.13}$$

where $S_i(\theta) = \{j \in S \mid \nabla p_{ij}(\theta) \neq 0\}$. Let t_m be the time epoch that state i^* is visited for the mth time and the sequence $\{i_{t_m}, i_{t_m+1}, \ldots, i_{t_{m+1}-1}\}$ be the mth regenerative cycle.

Based on Eq. (4.13), Marbach put forward the following estimate for $\nabla\chi(\theta)$:

$$F_m(\theta) = \sum_{n=t_m}^{t_{m+1}-1} \left(\tilde{J}_{i_n}(\theta) \frac{\nabla p_{i_{n-1}i_n}(\theta)}{p_{i_{n-1}i_n}(\theta)} + \nabla g_{i_n}(\theta) \right), \tag{4.14}$$

where

$$\tilde{J}_{i_n}(\theta) = \begin{cases} \sum_{k=n}^{t_{m+1}-1} g_{i_k}(\theta) & \text{if } t_m < n \le t_{m+1}-1, \\ 0 & \text{if } n = t_m. \end{cases} \tag{4.15}$$

In the proof of unbiasedness for the estimator given by Eq. (4.14) and Eq. (4.15) (Proposition 4 in [20]), it is claimed that $J_{i^*}(\theta) = 0$. However, this claim does not match the definition of $J_i(\theta)$ given by Eq. (4.5). In fact,

$$J_{i^*}(\theta) = E\left\{\sum_{t=0}^{T-1} g_{i_t}(\theta) \mid i_0 = i^*\right\} \tag{4.16}$$

by definition, which suggests that $J_{i^*}(\theta)$ can be estimated by the cumulative cost over a regenerative cycle. As a result, the proof of unbiasedness is incorrect.

In place of Marbach's estimator, we propose the following:

$$\bar{F}_m(\theta) = \sum_{n=t_m}^{t_{m+1}-1} \left(\tilde{J}_{i_{n+1}}(\theta)\frac{\nabla p_{i_n i_{n+1}}(\theta)}{p_{i_n i_{n+1}}(\theta)} + \nabla g_{i_n}(\theta)\right), \tag{4.17}$$

where

$$\tilde{J}_{i_n}(\theta) = \begin{cases} \sum_{k=n}^{t_{m+1}-1} g_{i_k}(\theta), & \text{if } t_m < n \le t_{m+1}-1, \\ \sum_{k=t_m}^{t_{m+1}-1} g_{i_k}(\theta), & \text{if } n = t_{m+1}. \end{cases} \tag{4.18}$$

Comparing the two estimators with Eq. (4.13), we argue that the corrected estimator corresponds better to Eq. (4.13), since i and j in Eq. (4.13) are consistent with i_n and i_{n+1} in Eq. (4.17), respectively, and $\tilde{J}_{i_{t_{m+1}}}(\theta)$ in Eq. (4.18) matches the definition of $J_{i^*}(\theta)$, which is the cumulative cost over a regenerative cycle. Since the cumulative cost over the next regenerative cycle is not available at t_{m+1}, in our estimator the current regenerative cycle from t_m to $t_{m+1}-1$ is used for $\tilde{J}_{i_{t_{m+1}}}(\theta)$ estimating, instead of the next regenerative cycle from t_{m+1} to $t_{m+2}-1$ suggested by Eq. (4.13). Using the corrected estimator, we can otherwise follow proof of Proposition 4 in [20] to prove that our estimator (4.17) is an unbiased estimate of the gradient.

Note that we can decompose (4.17) as

$$\bar{F}_m(\theta) = \sum_{n=t_m}^{t_{m+1}-1} \left(\tilde{J}_{i_{n+1}}(\theta)\frac{\nabla p_{i_n i_{n+1}}(\theta)}{p_{i_n i_{n+1}}(\theta)} + \nabla g_{i_n}(\theta)\right)$$

$$= \sum_{n=t_m}^{t_{m+1}-2} \frac{\nabla p_{i_n i_{n+1}}(\theta)}{p_{i_n i_{n+1}}(\theta)} \sum_{k=n+1}^{t_{m+1}-1} g_{i_k}(\theta) + \frac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)} \tilde{J}_{i_{t_{m+1}}}(\theta) + \sum_{n=t_m}^{t_{m+1}-1} \nabla g_{i_n}(\theta)$$
$$= \sum_{h=t_m+1}^{t_{m+1}-1} \frac{\nabla p_{i_{h-1} i_h}(\theta)}{p_{i_{h-1} i_h}(\theta)} \sum_{k=h}^{t_{m+1}-1} g_{i_k}(\theta) + \frac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)} \tilde{J}_{i_{t_{m+1}}}(\theta) + \sum_{n=t_m}^{t_{m+1}-1} \nabla g_{i_n}(\theta)$$
$$= \sum_{k=t_m+1}^{t_{m+1}-1} g_{i_k}(\theta) \sum_{h=t_m+1}^{k} \frac{\nabla p_{i_{h-1} i_h}(\theta)}{p_{i_{h-1} i_h}(\theta)} + \frac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)} \tilde{J}_{i_{t_{m+1}}}(\theta) + \sum_{n=t_m}^{t_{m+1}-1} \nabla g_{i_n}(\theta)$$
$$= \sum_{k=t_m+1}^{t_{m+1}-1} \left(\nabla g_{i_k}(\theta) + g_{i_k}(\theta) \sum_{h=t_m+1}^{k} \frac{\nabla p_{i_{h-1} i_h}(\theta)}{p_{i_{h-1} i_h}(\theta)} \right) + \frac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\theta)} \sum_{h=t_m}^{t_{m+1}-1} g_{i_h}(\theta)$$
$$= \sum_{k=t_m+1}^{t_{m+1}-1} \left(\nabla g_{i_k}(\theta) + g_{i_k}(\theta) z_k + \frac{\nabla p_{i_k i_{k+1}}(\theta)}{p_{i_k i_{k+1}}(\theta)} (L_k + g_{i_k}(\theta)) I_{\{k=t_{m+1}-1\}} \right)$$

where $I_{\{\cdot\}}$ is the indicator function,

$$z_{k+1} = \begin{cases} 0, & \text{if} \quad i_{k+1} = i^*; \\ z_k + \frac{\nabla p_{i_k i_{k+1}}(\theta)}{p_{i_k i_{k+1}}(\theta)} & \text{otherwise,} \end{cases} \tag{4.19}$$

and

$$L_{k+1} = \begin{cases} 0, & \text{if} \quad i_{k+1} = i^*; \\ L_k + g_{i_k}(\theta), & \text{otherwise.} \end{cases} \tag{4.20}$$

Note that at $k = t_m$, both $\nabla g_{i_k}(\theta)$ and $g_{i_k}(\theta)$ are zero. Hence, $\hat{R}_k(\theta)$ in Equations (4.9) (4.10) can be substituted by

$$R(x_k, \theta) = \nabla g_{i_k}(\theta) + g_{i_k}(\theta) z_k + \frac{\nabla p_{i_k i_{k+1}}(\theta)}{p_{i_k i_{k+1}}(\theta)} (L_k + g_{i_k}(\theta)) I_{\{i_{k+1}=i^*\}}, \tag{4.21}$$

where $x_k = (i_k, z_k, L_k)$.

4.2.2 Two-Timescale Simulation-Based Gradient Algorithm

Now we present our special two-timescale simulation-based gradient algorithm, which updates at some given time epoch n_l, defined by two-timescale step sizes α_k and β_l. In this algorithm, the parameter θ_l is updated as follows:

$$\theta_{l+1} = \theta_l + \sum_{k=n_l}^{n_{l+1}-1} \alpha_k \left(\nabla g_{i_k}(\theta_l) + g_{i_k}(\theta_l) z_k + \frac{\nabla p_{i_k i_{k+1}}(\theta_l)}{p_{i_k i_{k+1}}(\theta_l)} (L_k + g_{i_k}(\theta_l)) I_{\{i_{k+1}=i^*\}} \right), \tag{4.22}$$

where n_l satisfies (4.11), and z_k and L_k are given by (4.19) and (4.20), respectively. Note that z_k and L_k are updated at every time k, which is a faster timescale; θ_l is updated at time n_l, which is a slower timescale.

To prove its convergence, we make additional assumptions as in [20].

Assumption 4.2.2 *The step sizes α_k are non-increasing. Furthermore, there exist a positive integer p and a positive scalar A such that*

$$\sum_{k=n}^{n+t}(\alpha_n - \alpha_k) \leq At^p \alpha_n^2,$$

for all positive integers n and t.

Assumption 4.2.3 (MCP Strong Termination) *There exist a state $i^* \in S$ and a positive integer N_0, such that, for every parameter vector $\theta \in \mathcal{R}^K$, we have $g_{i^*}(\theta) = 0$, and $P_{i^*i^*}(\theta) = 1$, and, for every state $i \in S$ and every collection $\{P_1, \ldots, P_{N_0}\}$ of N_0 matrices in the set $\overline{\mathcal{P}}$, we have $Q_{ii^*} > 0$, where the matrix Q is given by $Q = P_1 \cdots P_{N_0}$.*

We have the following convergence result for the two-timescale simulation-based gradient algorithm.

Proposition 4.2.1 *Let Assumptions 4.1.1, 4.2.1, 4.2.2, and 4.2.3 hold, and let $\{\theta_l\}$ be the sequence of parameter vectors generated by the two-timescale simulation-based gradient algorithm (4.22). Then, $\{\chi(\theta_l)\}$ converges and*

$$\lim_{l\to\infty} \parallel \nabla\chi(\theta_l) \parallel = 0$$

with probability 1.

Proof: See Appendix.

4.3 Parking Problem

This case study involves a well-known academic example which has been used in [4] to demonstrate the approximate policy iteration method. Here we adopt this example to illustrate the simulation-based gradient algorithms presented in the last two sections.

A driver is looking for a low-priced parking space on the way to his destination. The parking area contains N spaces. The driver starts at space N and crosses the parking spaces from space s to space $s-1$, $s = N, N-1, \ldots, 1$. The destination is parking space 0. Each parking space is empty with probability p independently of whether other parking spaces are empty or not. The driver can spot whether a parking space is empty only when he reaches it, and then, if it is empty, he makes a decision whether or not to park in that space. If he parks in space s, $s = N$, $N-1$, $\ldots$, 1, he incurs a cost $c(s) > 0$. If he reaches the destination without having parked, he must park in the destination's garage, which is expensive, and costs $C > 0$. The objective is to find the optimal parking policy.

In [4], this problem is formulated as a stochastic shortest path problem with termination state i^*. In addition to the termination state, state 0 corresponds to reaching the expensive garage, state (s, F) $(s = 1, \ldots, N)$ corresponds to space s being empty ("free"), and state $(s, \bar{F})$ $(s = 1, \ldots, N)$ corresponds to space s being not empty ("not free"). Two possible actions, u_p and u_n, represent the actions "park" and "do not park", respectively.

An MDP formulation for this stochastic shortest path problem follows.

States:

$$i \in S = \{0, i^*, (s,F), (s,\bar{F})\}, \quad \text{for all } s.$$

Actions:

$$u(i) = U(i) = \begin{cases} \{u_n, u_p\}, & i = (s,F), \quad \text{for all } s. \\ \{u_n\}, & i = (s,\bar{F}), \quad \text{for all } s. \\ \{u_p\}, & i = 0, \\ \emptyset, & i = i^*. \end{cases}$$

Transition probabilities:

$$\begin{aligned} p_{(s,F)i^*}(u_p) &= 1, \quad \text{for all } s, \\ p_{(1,j)0}(u_n) &= 1, \quad j \in \{F, \bar{F}\}, \end{aligned}$$

$$
\begin{aligned}
p_{(s,j)(s-1,F)}(u_n) &= p, \quad s = 2, \ldots, N, \;\; j \in \{F, \bar{F}\}, \\
p_{(s,j)(s-1,\bar{F})}(u_n) &= 1-p, \quad s = 2, \ldots, N, \;\; j \in \{F, \bar{F}\}, \\
p_{ii^*}(u_p) &= 1, \quad i \in \{0, i^*\}, \;\; u \in U(i).
\end{aligned}
$$

One-stage cost:

$$
\begin{aligned}
g((s,F), u_p) &= c(s), \quad \text{for all } s, \\
g((s,j), u_n) &= 0, \quad \text{for all } s, \;\; j \in \{F, \bar{F}\}, \\
g(0, u_p) &= C, \\
g(i^*, u) &= 0.
\end{aligned}
$$

Objective function:

The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_t : S \to U, \mu_t(i) \in U(i)$ for i and t, the total cost from state i,

$$
J_\pi(i) = E\left\{ \sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i \right\}, \tag{4.23}
$$

where T is defined as in Eq. (4.5).

The optimality equations for this stochastic shortest path problem are [4]:

$$
\begin{aligned}
&J^*(s) = p \min\{c(s), J^*(s-1)\} + (1-p)J^*(s-1), \qquad s = 1, \ldots, N, \\
&J^*(0) = C.
\end{aligned} \tag{4.24}
$$

An optimal policy has the form:

$$
\mu^*(s) = \begin{cases} u_p, & \text{if space } s \text{ is free and } c(s) \le J^*(s-1), \\ u_n, & \text{otherwise.} \end{cases}
$$

Furthermore, if $c(s)$ is monotonically increasing in s, there is an integer s^* such that it is optimal to park at space s if and only if s is free and $s \le s^*$, since $J^*(s)$ is monotonically nonincreasing in s [4]. Thus, the optimal policy is a threshold policy, i.e., to park at the first available space after a threshold space is reached.

This parking problem can also be formulated as a weighted cost problem. The difference between the weighted cost formulation and the stochastic shortest path formulation is Eq. (4.1) in place of Eq. (4.23). In addition, the optimal policy for the weighted cost problem is the same as that of the stochastic shortest path problem, a threshold policy.

Following Eq. (4.3), we can parameterize the policy for the weighted cost problem as a randomized policy:

$$\begin{aligned} q_{\mu((s,F),\theta)}(u_n) &= \frac{1}{1+\exp(\theta - s)} \\ q_{\mu((s,F),\theta)}(u_p) &= 1 - q_{\mu((s,F),\theta)}(u_n), \end{aligned}$$

where θ is the threshold and the problem is converted to finding the optimal θ.

Under the randomized policy and using Eq. (4.4), the transition probabilities are:

$$\begin{aligned} p_{(s,F)j}(\theta) &= p_{(s,F)j}(u_p) \;\; q_{\mu((s,F),\theta)}(u_p) + p_{(s,F)j}(u_p) \;\; q_{\mu((s,F),\theta)}(u_n), \\ p_{(s,\bar{F})j}(\theta) &= p_{(s,\bar{F})j}(u_n), \\ p_{0i^*}(\theta) &= p_{0i^*}(u_p), \\ p_{i^*j}(\theta) &= p_{i^*j}(u); \end{aligned}$$

and the corresponding one-stage costs are:

$$\begin{aligned} g_{(s,F)}(\theta) &= g((s,F),u_p) \;\; q_{\mu((s,F),\theta)}(u_p) + g((s,F),u_n) \;\; q_{\mu((s,F),\theta)}(u_n), \\ g_{(s,\bar{F})}(\theta) &= g((s,\bar{F}),u_n), \\ g_0(\theta) &= g(0,u_p), \\ g_{i^*}(\theta) &= g(i^*,u). \end{aligned}$$

The objective function is: $\chi(\theta) = \xi_1 J_{(N,F)}(\theta) + \xi_2 J_{(N,\bar{F})}(\theta)$, where $\xi_1 = p$, $\xi_2 = 1-p$, and $J_i(\theta)$ is defined as in Eq. (4.5).

Now, given a fixed $\bar{\theta}$, the state sequence $i_0(\bar{\theta})$,. . .,$i_T(\bar{\theta})$ generated according to the above transition probabilities and one-stage costs is a Markov cost process.

However, for the simulation-based gradient algorithms, we need to define a new Markov cost process with new transition probabilities

$$p_{\xi,ij}(\theta) = \begin{cases} p_{ij}(\theta) & \text{if } i \neq i^*; \\ \xi_j & \text{if } i = i^*, \end{cases}$$

and new one-stage costs $g_{\xi,i}(\theta) = g_i(\theta)$.

The new Markov cost process is a renewal process. So we need only one single sample path when we apply simulation-based gradient algorithms, where the initial state is i^*.

4.4 Numerical Experiments

We now compare the three algorithms on the parking problem: the two-timescale algorithm [given by Eq. (4.22)], the (corrected) regenerative-update simulation-based gradient algorithm [Eq. (4.8) with $\bar{F}_m(\theta_m)$ given by Eq. (4.17) in place of $\hat{F}_m(\theta_m)$], and the (corrected) every-update simulation-based gradient algorithm [Eq. (4.9) with $R_k(x_k, \theta_k)$ given by (4.21) in place of $\hat{R}_k(\theta_k)$]. Note that the convergence of the latter two algorithms can be proved following the same arguments as in the proofs of Proposition 5 and Proposition 14 in [20].

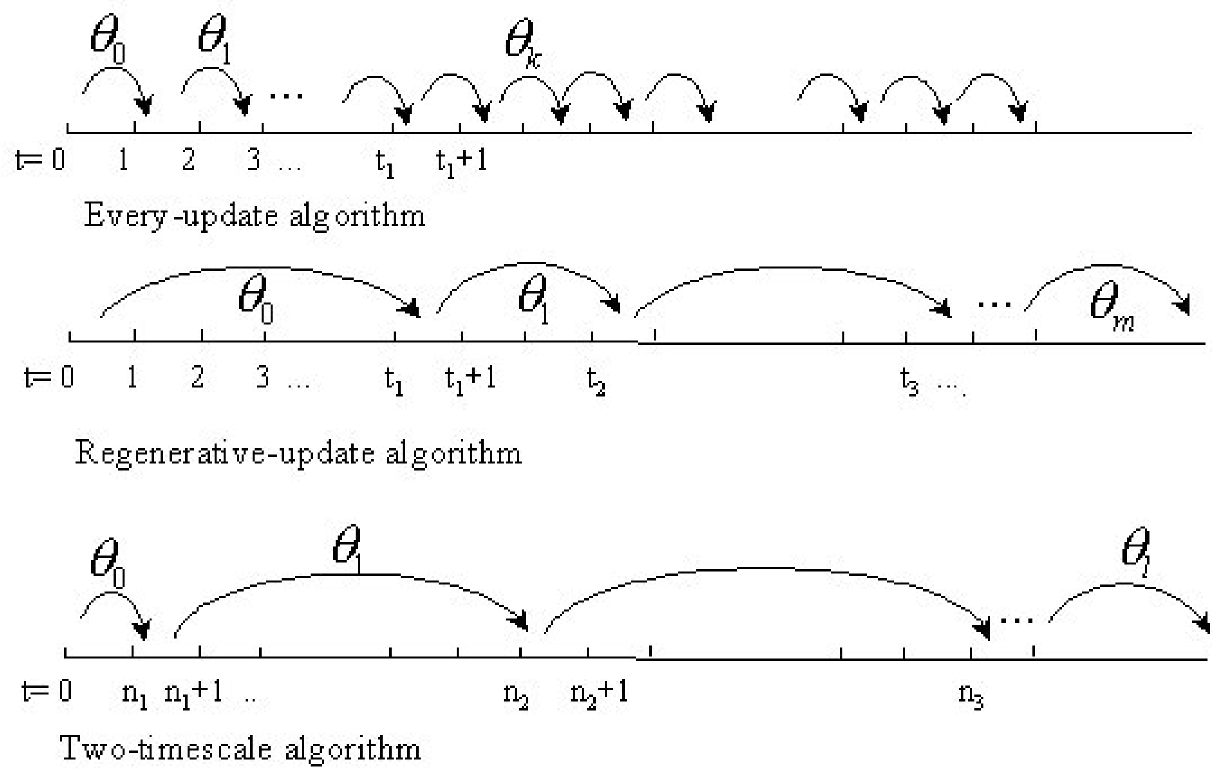
The three algorithms are different in the manner of estimating gradients and updating θ, which is shown in Figure 4.1.

In our numerical experiments, we consider the same case as in [4] where

$$p = 0.05, \quad c(s) = s, \quad C = 100, \quad N = 200.$$

The optimal policy for this case is to park in the first available space after the parking space 35 is reached. Fig. 4.2 shows the expected cost $\chi(\theta)$ as a function of threshold θ. The optimal cost $\chi(\theta^*) = 35.7639$ when $\theta^* = 35$.

All three algorithms were implemented in C, and experiments were conducted on a Sun Microsystems ULTRA10 running Solaris 2.6 operating system. For all



θ_0
θ_1
θ_k
t= 0 1 2 3 … t_1 t_1+1
Every-update algorithm
θ_0
θ_1
θ_m
t= 0 1 2 3 … t_1 t_1+1 t_2 t_3 …
Regenerative-update algorithm
θ_0
θ_1
θ_l
t= 0 n_1 n_1+1 .. n_2 n_2+1 n_3
Two-timescale algorithm

Figure 4.1: The Comparison of the Three Algorithms

the experiments, we selected initial parameter $\theta_0 = 100$, for which $\chi(\theta_0) = 81.7045$. For each result, we simulated four sample paths using independent seeds and show the mean and standard deviation of θ. When we implemented these three algorithms, we used step sizes $\alpha_j = a\ j^c$ (where j is the updates index) and $\beta_l = b\ l^d$ (required for the two-timescale algorithm), where a, b, c, and d are tunable positive parameters. We also set $n_0 = 1$ for the two-timescale algorithm.

regenerative-update algorithm

Since we dealt with a renewal process, we only needed to simulate one single sample path starting from state i^*. At every time epoch n, we calculated the one-stage cost differential $\nabla g_{i_n}(\theta_m)$, transition probability $p_{i_n i_{n+1}}(\theta_m)$, transition

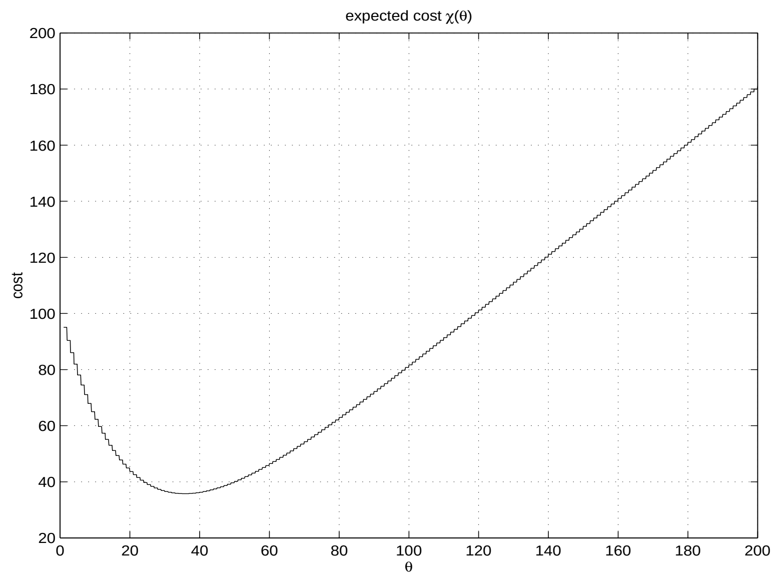


Figure 4.2: Expected Cost $\chi(\theta)$

probability differential $\nabla p_{i_n i_{n+1}}(\theta_m)$, and cumulative costs $\tilde{J}_{i_{n+1}}(\theta_m)$, where time epoch n is in the mth regenerative cycle. At the end of the mth regenerative cycle, we computed the gradient estimate using Eq. (4.17) and updated θ_m.

First we consider the regenerative-update algorithm implementation with fixed a and c. Figures 4.3(a) and 4.3(b) show how θ_m and $\hat{\chi}(\theta_m)$ converge to a near-optimal value, with $a = 2.0$ and $c = 0.662$. Note that we plot the progressions as a function of time epochs rather than as a function of iterative (regenerative) updates to facilitate comparison with the other algorithms. At time epoch n, $\theta_{n,1}$ to $\theta_{n,4}$ are four samples, their mean is $\bar{\theta}_n = (\sum_{i=1}^{4} \theta_{n,i})/4$, and their sample standard deviation (std) is calculated by $\sqrt{\sum_{i=1}^{4}(\theta_{n,i} - \bar{\theta}_n)^2/3}$. The trajectories shown in Figure 4.3(a) (or b) are the mean, the mean plus the standard deviation, the mean minus standard deviation, and the optimal value of θ_n (or $\chi(\theta_n)$). Note that θ converges to a value with near-optimal cost within $1,000,000$ time epochs. After that, θ_n improves more slowly, with some small oscillations. The final value $\theta_{5,000,000} = 37.34 \pm 1.82$ and $\chi(\theta_{5,000,000}) = 35.89 \pm 0.16$.

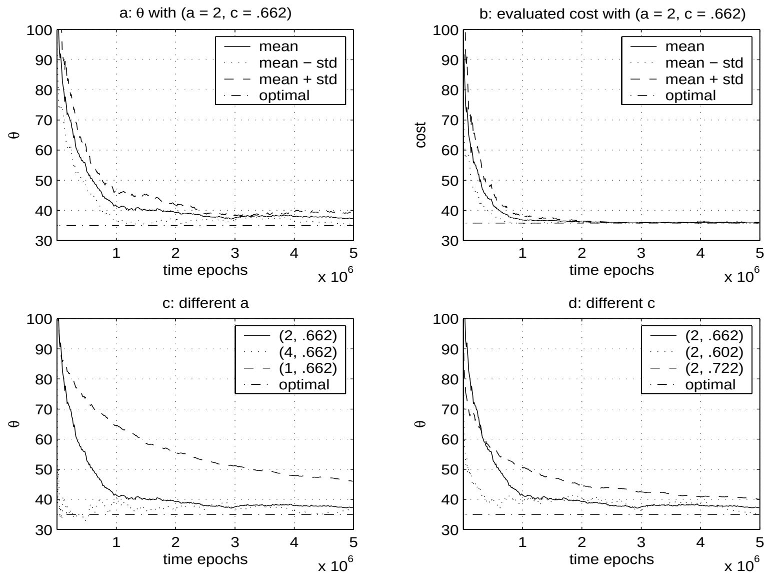


Figure 4.3: Simulation-Based Regenerative-Update Gradient Algorithm

Next, we investigated the sensitivity of this algorithm to a and c. From Fig. 4.3(c), we can observe that too small a value of a leads to slow convergence, but larger values cause overshoots. In contrast, Fig. 4.3(d) indicates the effect is opposite for c, i.e., too large a value leads to slow convergence, whereas small values can result in fluctuations. This type of sensitivity is typical of stochastic approximation algorithms.

Note that implementation of the regenerative-update algorithm requires storage of the accumulated costs $\tilde{J}_{i_{n+1}}(\theta)$ for all time epochs in a regenerative cycle before estimating the gradient estimate. If the expected length of a regenerative cycle is large, the storage requirements of this algorithm might make it impractical.

every-update algorithm

Implementation of the every-update algorithm requires calculation at each time epoch k of the one-stage cost $g_{i_k}(\theta_k)$, one-stage cost differential $\nabla g_{i_k}(\theta_k)$, transition probability $p_{i_k i_{k+1}}(\theta_k)$, and transition probability differential $\nabla p_{i_k i_{k+1}}(\theta_k)$,

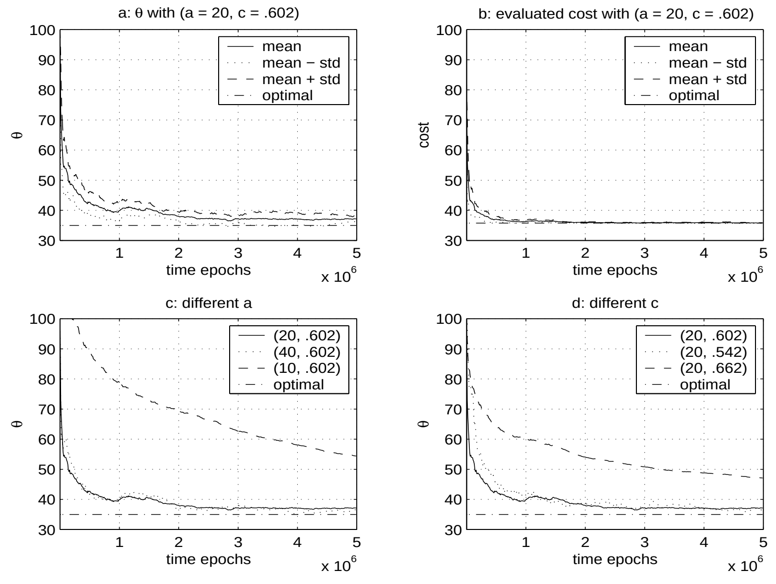


Figure 4.4: Simulation-Based Every-Update Gradient Algorithm

and then updating θ_k, L_k and z_k using (4.9), (4.20), and (4.19). At the end of each regenerative cycle, L_k and z_k must be reset.

With $a = 20.0$ and $c = 0.662$, Fig. 4.4(a) and 4.4(b) show how θ_k and $\hat{\chi}(\theta_k)$ converge to the optimal value. Again, θ_k is near-optimal within about $1,000,000$ time epochs, converging more slowly to the optimal after that, with $\theta_{5,000,000} = 37.12 \pm 0.70$ and $\chi(\theta_{5,000,000}) = 35.82 \pm 0.08$. Sensitivity analysis of this algorithm w.r.t. a and c, as seen in Fig. 4.4(c) and 4.4(d), is basically the same as for the regenerative-update algorithm.

two-timescale algorithm

Implementation of the two-timescale algorithm is similar to the implementation of the every-update algorithm, the main difference being that θ is updated at a slower scale, instead of at every time epoch. The number of time epochs before the next update of θ is characterized by n_l (see Eq. (4.11)) where l is the number of updates so far. At each time epoch k of the θ_l update, the quantities $g_{i_k}(\theta_l)$, $\nabla g_{i_k}(\theta_l)$, $p_{i_k i_{k+1}}(\theta_l)$, $\nabla p_{i_k i_{k+1}}(\theta_l)$ must be calculated, and L_k and z_k updated. At

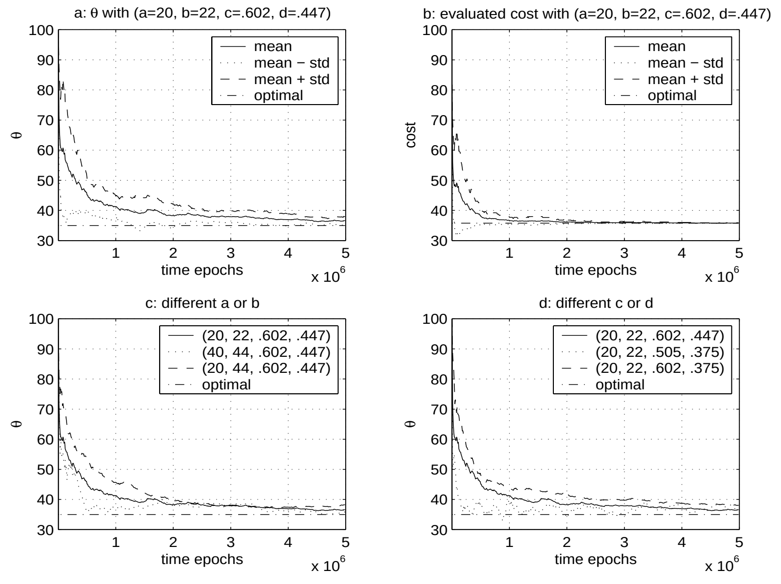


Figure 4.5: Simulation-Based Two-Timescale Gradient Algorithm

the end of each regenerative cycle, L_k and z_k are reset.

From Fig. 4.5(a) and 4.5(b), we can see how θ_l and $\hat{\chi}(\theta_l)$ converge to a near-optimal value, with $a = 20$, $b = 22$, $c = 0.602$, and $d = 0.447$. Again, θ_l proceeds rapidly to near optimality within $1,000,000$ time epochs, with final value $\theta_{5,000,000} = 36.62 \pm 1.35$ and $\chi(\theta_{5,000,000}) = 35.83 \pm 0.09$.

Next, we investigated the sensitivity of this algorithm to a, b, c and d. We can see the influence of both a and b is similar to that of a in the regenerative-update algorithm from Fig. 4.5(c), and the influence of both c and d is similar to that of c in the regenerative-update algorithm from 4.5(d). Note that b and d affect the convergence behavior of the two-timescale algorithm only through n_l.

Comparing these three algorithms, only the the regenerative-update algorithm's implementation requires storing interim values. Comparing the every-update algorithm and the two-timescale algorithm, the former requires more computations since it updates θ at each time epoch, but the latter contains more tunable parameters.

4.5 Discussion

In this chapter, we proposed a new two-timescale simulation-based algorithm based on perturbation analysis for weighted cost problems, and compared it with two simulation-based algorithms proposed by Marbach [20] via numerical experiments on a parking example. Implementation of the regenerative-update algorithm requires storing interim values before each regenerative point. The numerical experiments show that the performance of the two-timescale algorithm is close to that of the every-update algorithm. However, in some situations when we need frequent estimation but infrequent control, the two-timescale algorithm is a better choice than the every-update algorithm.

Note that all the three algorithms mentioned in this chapter need information about transition probabilities and gradient of stage costs and sometimes may not be applicable, in which situation SPSA algorithms discussed in the next chapter may be better choices. However, when algorithms based on perturbation analysis are applicable, they will outperform SPSA algorithms.

Chapter 5

Simultaneous Perturbation Stochastic Approximation Algorithms

In this chapter, we first describe two Simultaneous Perturbation Stochastic Approximation (SPSA) algorithms for weighted cost problems and carry out numerical experiments on the parking problem, comparing them with simulation-based gradient algorithms discussed in Chapter 4. Second, we consider a general SPSA algorithm for function minimization and show that the SPSA algorithm converges under a weaker assumption – the function does not have to be differentiable.

The simulation-based gradient algorithms for weighted cost problems in Chapter 4 require explicit model information about the transition probabilities. The SPSA algorithms proposed in this chapter do not use this information. One of the SPSA algorithms for weighted cost problems is a special case of the SPSA algorithm proposed by Spall [35]; the other is extended from two-timescale SPSA for average cost problems proposed by Bhatnagar et al. [32] [33]. We will describe these two algorithms in Section 5.1.

Convergence of SPSA has been analyzed under various conditions. Much of the literature assumes the function be three times differentiable [35] [38] [39] [40] [41] [42] [43], though weaker assumptions are found as well, e.g. [44] [45] [46] [47]

[48]. However, all of them require that the function be at least differentiable. Among the weakest assumptions on the minimized function, Fu and Hill [45] assume that the function is differentiable and convex; Chen et al. [44] assume that the function is differentiable and the gradient satisfies a Lipschitz condition. In the fab-level decision-making problem presented in Chapter 2, we found that the one-step cost function is continuous and convex with respect to the action, but is nondifferentiable, so that the problem of finding the one-step optimal action requires minimizing a continuous and convex function. So the question is: does the SPSA algorithm converge in this setting? The answer is affirmative, and the details will be presented in Section 5.2.

Gerencsér et al. [61] have discussed non-smooth optimization. However, they approximate the non-smooth function by a smooth enough function, and then optimize the smooth function by SPSA. Thus, they take an indirect approach.

5.1 SPSA Algorithms for Weighted Cost Problems

In Chapter 4, we discussed simulation-based gradient algorithms for weighted cost problems, provided that explicit information about the transition probabilities is given. However, in many situations, this is not the case, but SPSA algorithms can be easily applied. As in Section 4.1, we parameterize the policy by a parameter vector $\theta \in R^p$, so the original problem of finding an optimal policy for an MDP becomes one of finding the optimal θ minimizing $\chi(\theta)$, where

$$\chi(\theta) = \sum_{i \in S} \xi_i J_i(\theta),$$

and

$$J_i(\theta) = E\left\{ \sum_{t=0}^{T-1} g_{i_t}(\theta) \mid i_0 = i \right\}.$$

The parameterized stationary policy is $\pi(\theta) = \{\mu, \mu, \ldots\}$, with $\mu : S \times R^p \to U$, and the corresponding decision rule is $\mu(i, \theta)$, with its dependence on θ assumed known.

First, as in Chapter 4, we define a new Markov process with new transition probabilities

$$p_{\xi,ij}(\theta) = \begin{cases} p_{ij}(\theta) & \text{if } i \neq i^*; \\ \xi_j & \text{if } i = i^*, \end{cases}$$

and new one-stage costs $g_{\xi,i}(\theta) = g_i(\theta)$. The new Markov process is a renewal process. Then,

$$\chi(\theta) = E\{\sum_{t=t_n}^{t_{n+1}-1} g_{i_t}(\theta)\},$$

where $\{i_{t_n}, i_{t_n+1}, \ldots, i_{t_{n+1}-1}\}$ is the nth renewal cycle. Note that $\chi(\theta)$ can be interpreted as the expected cost over a single renewal cycle.

SPSA algorithm

Let $\{\Delta_m\}$ be a sequence of mutually independent column vectors with zero-mean i.i.d. random components $(\Delta_{m,i}, i = 1, \ldots, p)$, and let c_m be a positive gain sequence that goes to zero. An SPSA algorithm for weighted cost problems is:

$$\theta_{m+1} = \theta_m + \alpha_m [\Delta_m^{-1}] \frac{y_m^+ - y_m^-}{2c_m}, \tag{5.1}$$

where

$$y_m^+ = \textstyle\sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m + c_m \Delta_m),$$
$$y_m^- = \textstyle\sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m - c_m \Delta_m),$$

and $\{\alpha_m\}$ is a sequence of step sizes. Note that $y_m^\pm$ are sample estimates of $\chi(\theta_m \pm c_m \Delta_m)$.

SPSA was proposed by Spall [35] and has been implemented in many problems [36]. Like stochastic approximation algorithms based on finite difference methods, SPSA is also a Kiefer-Wolfowitz-type stochastic approximation algorithm [37], requiring no detailed knowledge of the system dynamics. The chief merit of SPSA is that it requires only two sample estimates to calculate a gradient estimate of the objective function, regardless of the dimension of the parameter vector.

The convergence of SPSA can be proved under appropriate assumptions placed on the following : 1) the objective function $\chi(\theta)$; 2) the step size α_m; 3) bias and variance of the gradient estimate [45]. For 1) and 2), we have $\chi(\theta)$ continuously differentiable under Assumption 4.1.1 and $\{\alpha_m\}$ are deterministic, nonnegative, and satisfy $\sum_{m=1}^{\infty} \alpha_m = \infty$ and $\sum_{m=1}^{\infty} \alpha_m^2 < \infty$, as in Chapter 4.

For 3), first define

$$\begin{aligned} G_m &:= [\Delta_m^{-1}]\frac{y_m^+ - y_m^-}{2c_m}, &(5.2)\\ b_m &:= E[G_m \mid \theta_m] - \nabla\chi(\theta_m), &(5.3)\\ e_m &:= G_m - E[G_m \mid \theta_m], &(5.4)\end{aligned}$$

where G_m is the gradient estimate, b_m is the bias of the estimate, and e_m is the noise term.

Lemma 5.1.1 *If Δ_m has bounded second moments and $E[|\Delta_m^{-1}|]$ is uniformly bounded, then $b_m \to 0$ with probability 1.*

Proof: $E\left[[\Delta_m^{-1}]\frac{y_m^+ - y_m^-}{2c_m}\right] =$

$$\begin{aligned} & E\left[[\Delta_m^{-1}]\frac{\sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m + c_m\Delta_m) - \sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m - c_m\Delta_m)}{2c_m}\right] \\ = \; & E\left[E\left[[\Delta_m^{-1}]\frac{\begin{array}{c}\sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m + c_m\Delta_m) \\ - \sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m - c_m\Delta_m\end{array}}{2c_m} \mid \Delta_m\right]\right] \\ = \; & E\left[[\Delta_m^{-1}]\frac{\begin{array}{c}E[\sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m + c_m\Delta_m) \mid \Delta_m] \\ - E[\sum_{t=t_m}^{t_{m+1}-1} g_{i_t}(\theta_m - c_m\Delta_m) \mid \Delta_m]\end{array}}{2c_m}\right]\end{aligned}$$

$$
\begin{aligned}
&= E\left[[\Delta_m^{-1}]\frac{E[\chi(\theta_m + c_m\Delta_m) \mid \Delta_m] - E[\chi(\theta_m - c_m\Delta_m) \mid \Delta_m]}{2c_m}\right] \\
&= E\left[[\Delta_m^{-1}]\frac{\begin{array}{c}(E[\chi(\theta_m)] + c_m\Delta_m^T\nabla\chi(\theta_m) + O(|c_m|^2|\Delta_m|^2) \\ -\ (E[\chi(\theta_m)] - c_m\Delta_m^T\nabla\chi(\theta_m) + O(|c_m|^2|\Delta_m|^2)\end{array}}{2c_m}\right] \\
&= E\left[[\Delta_m^{-1}]\frac{(2c_m\Delta_m^T\nabla\chi(\theta_m) + O(|c_m|^2|\Delta_m|^2)}{2c_m}\right] \\
&= E\left[[\Delta_m^{-1}]\Delta_m^T\nabla\chi(\theta_m) + O(|c_m||\Delta_m|^2)\right] \\
&= \nabla\chi(\theta_m) + E\left[([\Delta_m^{-1}]\Delta_m^T - I)\right]\nabla\chi(\theta_m) + E[O(|c_m||\Delta_m|^2)] \\
&= \nabla\chi(\theta_m) + O(|c_m|).
\end{aligned}
$$

Hence,

$$b_m = E[G_m \mid \theta_m] - \nabla\chi(\theta_m) = O(|c_m|).$$

Since c_m is a positive gain sequence that goes to zero, $b_m \to 0$ with probability 1. $\square$

Lemma 5.1.2 *Suppose $E[|\Delta_m^{-1}|^2]$ is uniformly bounded, and $y_m^{\pm}$ have uniformly bounded second moments. Then, $E[e_m^T e_m]$ is $O(c_m^{-2})$.*

Proof:

$E[e_m^T e_m] =$

$$E\left[\left([\Delta_m^{-1}]\frac{y_m^+ - y_m^-}{2c_m} - E\left[[\Delta_m^{-1}]\frac{y_m^+ - y_m^-}{2c_m}\right]\right)^T\left([\Delta_m^{-1}]\frac{y_m^+ - y_m^-}{2c_m} - E\left[[\Delta_m^{-1}]\frac{y_m^+ - y_m^-}{2c_m}\right]\right)\right]$$

$$
\begin{aligned}
&= E\left[\left(\frac{y_m^+ - y_m^-}{2c_m}\right)^2[\Delta_m^{-1}]^T[\Delta_m^{-1}]\right] + E\left[E^2\left[\left(\frac{y_m^+ - y_m^-}{2c_m}\right)\right][\Delta_m^{-1}]^T[\Delta_m^{-1}]\right] \\
&\quad -2E\left[\left(\frac{y_m^+ - y_m^-}{2c_m}\right)E\left[\left(\frac{y_m^+ - y_m^-}{2c_m}\right)\right][\Delta_m^{-1}]^T[\Delta_m^{-1}]\right] \\
&\leq BE[|\Delta_m^{-1}|^2](c_m^{-2}) \\
&= O(c_m^{-2}).
\end{aligned}
$$

□

Before we present the main convergence result for (5.1), we cite a general convergence result by L'Ecuyer et al. [62] as follows:

Proposition 5.1.1 *Consider a problem of finding the optimal parameter vector θ to minimize $J(\theta) = E[L(\theta, \omega)]$, where $L(\theta, \omega)$ are measurements. Suppose a gradient stochastic algorithm has the form: $\theta_{m+1} = \theta_m + \alpha_m \tilde{G}_m(\theta_m)$, where α_m is the step size satisfying $\sum_{m=1}^{\infty} \alpha_m = \infty$. Assume*

1) $J(\theta)$ is differentiable for each θ, and either convex or unimodal,

2) $\tilde{b}_m \to 0$ with probability 1, and

3) $\sum_{m=1}^{\infty} E[\tilde{e}_m^T \tilde{e}_m]\alpha_m^2 < \infty$ with probability 1, where definitions of $\tilde{b}_m$ and $\tilde{e}_m$ are similar to (5.3) and (5.4) respectively except for the "". Then, $\theta_m \to \theta^*$ with probability 1,

Here is the main convergence result for (5.1):

Proposition 5.1.2 *Suppose $\chi(\theta)$ is differentiable for each θ, and either convex or unimodal, $\sum_{m=1}^{\infty} \alpha_m = \infty$, $\sum_{m=1}^{\infty} (\frac{\alpha_m}{c_m})^2 < \infty$, and $E[|\Delta_m^{-1}|]$ is uniformly bounded. Then, for the SPSA algorithm (5.1), $\theta_m \to \theta^*$ with probability 1.*

Proof: By Lemma 5.1.1, Lemma 5.1.2, and $\sum_{m=1}^{\infty} (\frac{\alpha_m}{c_m})^2 < \infty$, we have $b_m \to 0$ with probability 1, and $\sum_{m=1}^{\infty} E[e_m^T e_m]\alpha_m^2 < \infty$. Then, the result follows by Proposition 5.1.1. □

Remark: The condition that the function be differentiable will be relaxed in the next section.

There is another way to deal with the weighted cost problem. First, let us define a different new Markov process with a The objective function becomes:

$$\chi(\theta) = E\left[\sum_{t=-1}^{T-1} g(x_t, \mu(x_t, \theta)) \mid x_{-1} = z\right], \tag{5.5}$$

where $T = \min\{t > 0 \mid x_t = n\}$.

With this new formulation, we propose an SPSA algorithm with the form (5.1) but the two measurements being

$$\bar{y}_m^+ = \textstyle\sum_{t=-1}^{T-1} g_{i_t}(\theta_m + c_m\Delta_m),$$
$$\bar{y}_m^- = \textstyle\sum_{t=-1}^{T-1} g_{i_t}(\theta_m - c_m\Delta_m),$$

with $i_{-1} = z$ and $T = \min\{t > 0 \mid i_t = i^*\}$.

In this new algorithm, we treat ξ_i as a transition probability and simulate two sample paths ended with termination state i^*. Results similar to Lemma 5.1.1, Lemma 5.1.2 and Proposition 5.1.2 can be obtained.

Two-Timescale SPSA algorithm

Two-timescale SPSA algorithm is proposed by Bhatnagar et al. on average cost problems[32]. Here we extend it to the weighted cost problem, and leave the proof of its convergence for the weighted cost problem as a future research.

Let $\{\Delta_l\}$ be a sequence of mutually independent column vectors with zero-mean i.i.d. random components $(\Delta_{l,i}, i = 1, \ldots, p)$, and let c be a given positive constant. The two-timescale SPSA algorithm is:

$$\theta_{l+1} = \theta_l + \sum_{t=n_l}^{n_{l+1}-1} \alpha_t[\Delta_l^{-1}]\frac{g_{i_t}(\theta_l + c\Delta_l) - g_{i_t}(\theta_l - c\Delta_l)}{2c}, \tag{5.6}$$

where $n_{l+1} = \min\{j > n_l | \sum_{k=n_l}^{j-1} \alpha_k \geq \beta_l\}$, and α_t and β_l are step sizes.

Now we perform numerical experiments using SPSA algorithms on the same parking problem as in Chapter 4 and compare them with simulation-based gradient algorithms in Chapter 4.

Numerical Experiments

As in Chapter 4, the two SPSA algorithms are implemented in C and experiments are conducted on a Sun Microsystems ULTRA10. For all the experiments, we also select the same initial $\hat{\theta} = 100$ and $\chi(\hat{\theta}) = 81.7045$. Cost performances

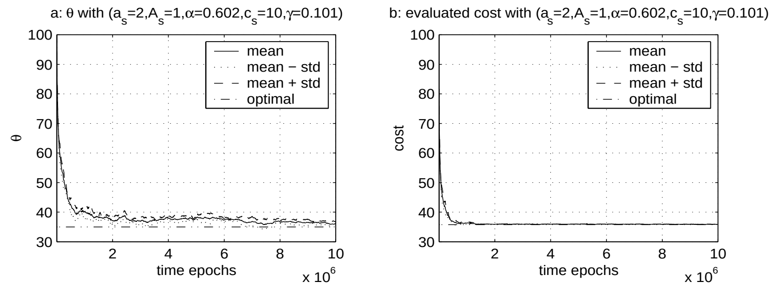


Figure 5.1: SPSA Algorithm

are evaluated by averaging results from 33 independently seeded 1000 renewal cycles replications. We also use four sample paths and show means and confident intervals.

SPSA algorithm

When we implement the proposed SPSA algorithm for weighted cost problems, we calculate two perturbed one-stage costs $g_{i_t}(\theta_m \pm c_m\Delta_m)$ at every time epoch t in the renewal cycle m, and compute a gradient estimate using (5.1) as well as update θ_m at the end of mth renewal cycle.

As with the gradient algorithms presented in Chapter 4, the choice of the step size α_m and gain sequences c_m is also crucial to the performance of SPSA algorithms. In [63], Spall provides a guideline on how to select gain sequences (and step sizes). Following the guideline, we implement SPSA algorithm for the parking problem using $\alpha_m = a_s\ (A_s + m + 1)^{\alpha}$, with ($a_s = 2$, $A_s = 1$, $\alpha = 0.602$),and $c_m = c_s\ (m+1)^{\gamma}$ with ($c_s = 10$, $\gamma = 0.101$). We also chose $\Delta_{m,i}$ using a Bernoulli distribution with probability of 1/2 for each ± 1 outcome. Fig. 5.1(a)&(b) show how θ_m and $\hat{\chi}(\theta_m)$ converge to near optimal values, with x label is still time epochs t.

Note that θ_m converges fast to a value with evaluated cost close to optimal one within 10^6 iterates. After that, θ_m only makes slow improvement with small oscillation. θ_m at t = 10^7 is 36.05 and the evaluated cost is 35.85.

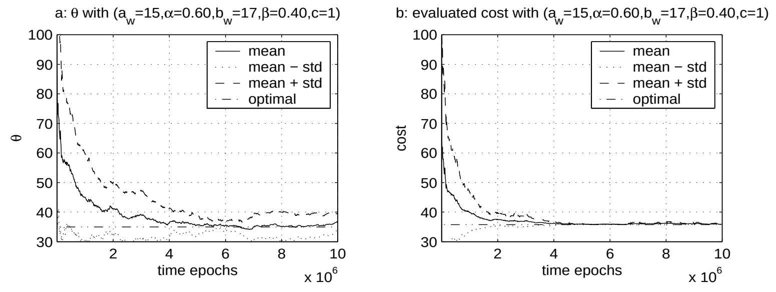


Figure 5.2: Two-Timescale SPSA Algorithm

If we compare the result with the regenerative-update algorithm, we found that both converge to near optimal values, with the variance of the θ resulting from the regenerative-update algorithm decreases and the variance of the θ resulting from the SPSA algorithm keeps at a small value.

Two-Timescale SPSA algorithm

When we implement the proposed two-timescale SPSA algorithm for weighted cost problems, we still calculate two perturbed one-stage costs $g_{i_t}(\theta_l \pm c\Delta_l)$ at every time epoch t, with each component of Δ_l chosen according to a Bernoulli distribution. However, we update θ_l using (5.6) only when we reach the end of the current n_l. Then we compute n_{l+1} for the next update.

As in [32], we chose $\alpha_t = a_w \; t^{\alpha}$, $\beta_l = b_w \; l^{\beta}$. However, the choice of the coefficients of the step sizes and gain c is by trial and error. Fig. 5.2(a)&(b) show convergence to the optimal value with a selected setting ($a_w = 15$,$\alpha = 0.60$, $b_w = 17$, $\beta = 0.40$, $c = 1$).

Compared to the SPSA algorithm, the two-timescale SPSA algorithm leads to a result with even bigger variance. It may be improved by choosing better step sizes and gain. However, for the two-timescale SPSA algorithm, it is not necessary to know when i^* is reached, which is an advantage to the SPSA algorithm.

5.2 Convergence of SPSA for Nondifferentiable Convex Function Optimization

In this section, we consider function minimization and show that the SPSA algorithm converges for nondifferentiable convex functions, which is especially important when the function is not differentiable at the minimizing point. First, similar to [48], we decompose the SPSA algorithm into four terms: a subgradient term, a bias term, a random direction noise term and an observation noise term. In our setting, the subgradient term replaces the gradient term in [48], since we assume that the function does not have to be differentiable. Hence, we need to show the asymptotic behavior of the algorithm follows a differential inclusion instead of an ordinary differentiable equation. Kushner and Yin [64] state a theorem (Theorem 5.6.2) for convergence of a Kiefer-Wolfowitz algorithm in a nondifferentiable setting. However, this theorem is not general enough to cover our SPSA algorithm. We will prove a more general theorem to establish convergence of SPSA.

The general approach for proving convergence for these types of algorithms requires showing that the bias term vanishes asymptotically. In the differentiable case, a Taylor series expansion or the mean value theorem is used to establish this. These tools are not applicable in our more general setting, but we are able to use convex analysis for this task. For the random direction noise term, we use a similar argument as in [48] to show the noise goes to zero with probability 1, except that now the term is a function of the subgradient instead of the gradient. For the observation noise term, the conditions for general Kiefer-Wolfowitz algorithms given in [64, pp. 113 - 114] are used, and we also show it goes to zero with probability 1. For more conditions on the observation noise term, see [48].

To be more specific, we want to minimize the function $E[F(\theta, \chi)] = f(\theta)$

over the parameter $\theta \in H \subset R^r$, where $f(\cdot)$ is continuous and convex, χ is a random vector and H is the feasible region of θ. Let θ_k denote the kth estimate of the minimum, and let Δ_k be an independent sequence of column random vectors with $\Delta_k = [\Delta_{k,1}, \ldots, \Delta_{k,r}]^T$. $\Delta_{k,1}, \ldots, \Delta_{k,r}$ are not necessary mutually independent. The two-sided SPSA algorithm to update θ_k is as follows:

$$\theta_{k+1} = \Pi_H \left(\theta_k - \alpha_k [\Delta_k^{-1}] \frac{F_k^+ - F_k^-}{2c_k} \right), \tag{5.7}$$

where Π_H is a projection on H, $F_k^{\pm}$ are observations taken at parameter values $\theta_k \pm c_k \Delta_k$, c_k is a positive sequence converging to zero, α_k is the step size, and $[\Delta_k^{-1}]$ is defined as $[\Delta_k^{-1}] := [\Delta_{k,1}^{-1}, \ldots, \Delta_{k,r}^{-1}]^T$.

Write the observation in the form

$$F_k^{\pm} = f(\theta_k \pm c_k \Delta_k) + \phi_k^{\pm},$$

where $\phi_k^{\pm}$ are observation "noises", and define

$$G_k := \frac{f(\theta_k + c_k\Delta_k) - f(\theta_k - c_k\Delta_k)}{2c_k}. \tag{5.8}$$

Then the algorithm (5.7) can be written as:

$$\theta_{k+1} = \Pi_H \left(\theta_k - \alpha_k [\Delta_k^{-1}] G_k + \alpha_k [\Delta_k^{-1}] \frac{\phi_k^- - \phi_k^+}{2c_k} \right). \tag{5.9}$$

The convergence of the SPSA algorithm (5.7) has been proved under various conditions [36]. One of the weakest conditions is that $f(\cdot)$ be differentiable and convex [45]. Under this condition instead of the condition that $f(\cdot)$ is continuous and convex, we can use a Taylor series expansion or the mean value theorem to get $f(\theta_k \pm c_k\Delta_k) = f(\theta_k) \pm c_k \Delta_k^T \nabla f(\theta_k) + O(|c_k|^2 |\Delta_k|^2)$. Therefore, $G_k = \Delta_k^T \nabla f(\theta_k) + O(|c_k||\Delta_k|^2)$, which means G_k can be approximated by $\Delta_k^T \nabla f(\theta_k)$. Then, suppose $H = R^r$, the algorithm (5.9) can be written as:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla f(\theta_k) + \alpha_k (I - [\Delta_k^{-1}]\Delta_k^T) \nabla f(\theta_k) + \alpha_k [\Delta_k^{-1}] \frac{\phi_k^- - \phi_k^+}{2c_k},$$

where a standard argument of the ODE method implies that the trajectory of θ_k follows the ODE

$$\dot{\theta} = -\nabla f(\theta).$$

In our context, however, we only assume that $f(\cdot)$ is continuous and convex – $\nabla f(\cdot)$ may not exist at some points, so a Taylor series expansion or the mean value theorem is not applicable. Instead, using convex analysis we show that G_k is close to the product of Δ_k^T and a *subgradient* of $f(\cdot)$.

5.2.1 Subgradient and Reformulation of the SPSA Algorithm

First, we introduce some definitions and preliminary results on convex analysis, with more details in [65].

Let h be a real-valued convex function on R^r; a vector $sg(x)$ is a *subgradient* of h at a point x if $h(z) \geq h(x) + (z-x)^T sg(x), \forall z$. The set of all subgradients of h at x is called the *subdifferential of h at x* and is denoted by $\partial h(x)$ [65, p. 214]. If h is a convex function, the set $\partial h(x)$ is a convex set, which means that $\lambda z_1 + (1-\lambda) z_2 \in \partial h(x)$ if $z_1 \in \partial h(x)$, $z_2 \in \partial h(x)$ and $0 \leq \lambda \leq 1$.

The *one-sided directional derivative* of h at x with respect to a vector y is defined to be the limit

$$h'(x;y) = \lim_{\lambda \downarrow 0} \frac{h(x+\lambda y) - h(x)}{\lambda}. \tag{5.10}$$

According to Theorem 23.1 in [65, p. 213], if h is a convex function, $h'(x;y)$ exists for each y. Furthermore, according to Theorem 23.4 in [65, p. 217], at each point x, the subdifferential $\partial h(x)$ is a non-empty closed bounded convex set, and for each vector y the directional derivative $h'(x;y)$ is the maximum of the inner products $\langle sg(x), y \rangle$ as $sg(x)$ ranges over $\partial h(x)$. Denote the set of $sg(x)$ on which $h'(x;y)$ attains its maximum by $\partial h_y(x)$. Thus, for all $sg_y(x) \in \partial h_y(x)$

and $sg(x) \in \partial h(x)$,

$$h'(x;y) = y^T sg_y(x) \geq y^T sg(x).$$

Now let us discuss the relationship between G_k defined by (5.8) and subgradients.

Lemma 5.2.1 *Consider the algorithm (5.7), assume $f(\cdot)$ is a continuous and convex function, and $\lim_{k\to\infty} c_k = 0$. Then $\forall \varepsilon > 0$, $\exists$ $\widetilde{sg}(\theta_k) \in \partial f(\theta_k)$ and finite K, such that*

$$|G_k - \Delta_k^T \widetilde{sg}(\theta_k)| < \varepsilon, \qquad \forall k \geq K.$$

Proof: Since $f(\cdot)$ is a continuous and convex function, both $f'(\theta_k; \Delta_k)$ and $f'(\theta_k; -\Delta_k)$ exist. By (5.10) and $\lim_{k\to\infty} c_k = 0$, $\forall \varepsilon > 0$, $\exists$ $K < \infty$ such that for all $k \geq K$,

$$|f'(\theta_k;\Delta_k) - \frac{f(\theta_k + c_k\Delta_k) - f(\theta_k)}{c_k}| < \varepsilon,$$
$$|f'(\theta_k;-\Delta_k) - \frac{f(\theta_k - c_k\Delta_k) - f(\theta_k)}{c_k}| < \varepsilon.$$

Since $G_k = (f(\theta_k + c_k\Delta_k) - f(\theta_k - c_k\Delta_k))/(2c_k)$,

$$|G_k - \frac{1}{2}(f'(\theta_k;\Delta_k) - f'(\theta_k;-\Delta_k))| < \varepsilon. \tag{5.11}$$

In addition, for $f'(\theta_k;\Delta_k)$, there exists $sg_{\Delta_k}(\theta_k) \in \partial f_{\Delta_k}(\theta_k)$ such that

$$f'(\theta_k;\Delta_k) = \Delta_k^T sg_{\Delta_k}(\theta_k). \tag{5.12}$$

Similarly, for $f'(\theta_k;-\Delta_k)$, there exists $sg_{-\Delta_k}(\theta_k) \in \partial f_{-\Delta_k}(\theta_k)$ such that

$$f'(\theta_k;-\Delta_k) = (-\Delta_k)^T sg_{-\Delta_k}(\theta_k). \tag{5.13}$$

Combining (5.11), (5.12) and (5.13), we conclude that $\forall$ $\varepsilon > 0$, $\exists$ finite K and $sg_{\Delta_k}(\theta_k)$, $sg_{-\Delta_k}(\theta_k) \in \partial f(\theta_k)$ such that

$$|G_k - \Delta_k^T(\frac{1}{2}sg_{\Delta_k}(\theta_k) + \frac{1}{2}sg_{-\Delta_k}(\theta_k))| < \varepsilon, \quad k \geq K. \tag{5.14}$$

Note that $\partial f(\theta_k)$ is a convex set, so $\widetilde{sg}(\theta_k) := \frac{1}{2} sg_{\Delta_k}(\theta_k) + \frac{1}{2} sg_{-\Delta_k}(\theta_k) \in \partial f(\theta_k)$, and $|G_k - \Delta_k^T \widetilde{sg}(\theta_k)| < \varepsilon$. $\square$

Define $\delta_k = \Delta_k^T \widetilde{sg}(\theta_k) - G_k$. The SPSA algorithm (5.9) can be decomposed as

$$\theta_{k+1} = \Pi_H \left(\theta_k - \alpha_k \widetilde{sg}(\theta_k) + \alpha_k (I - [\Delta_k^{-1}]\Delta_k^T)\widetilde{sg}(\theta_k) + \alpha_k [\Delta_k^{-1}]\delta_k + \alpha_k [\Delta_k^{-1}] \frac{\phi_k^- - \phi_k^+}{2c_k} \right). \tag{5.15}$$

Suppose $H = R^r$, and if we can prove that the third, fourth, and fifth terms inside of the projection go to zero as k goes to infinity, the trajectory of θ_k would follow the differential inclusion [64, p. 16]

$$\dot{\theta} \in -\partial f(\theta).$$

According to [65, p. 264], the necessary and sufficient condition for a given x to belong to the minimum set of f (the set of points where the minimum of f is attained) is that $0 \in \partial f(x)$.

5.2.2 Basic Constrained Stochastic Approximation Algorithm

Kushner and Yin [64, p. 124] state a theorem (Theorem 5.6.2) for convergence of a Kiefer-Wolfowitz algorithm in a nondifferentiable setting. However, this theorem is not general enough to cover the SPSA algorithm given by (5.15). So, we establish a more general theorem.

Note that the SPSA algorithm given by (5.15) is a special case of the stochastic approximation algorithm:

$$\begin{aligned} \theta_{k+1} &= \Pi_H(\theta_k + \alpha_k \ \widetilde{sf}(\theta_k) + \alpha_k \ b_k + \alpha_k \ e_k) & (5.16) \\ &= \theta_k + \alpha_k \ \widetilde{sf}(\theta_k) + \alpha_k \ b_k + \alpha_k \ e_k + \alpha_k \ Z_k, & (5.17) \end{aligned}$$

where Z_k is the reflection term, b_k is the bias term, e_k is the noise term, and

$\widetilde{sf}(\theta_k)$ can be any element of $-\partial f(\theta_k)$. Similar to (5.15), we need to show that b_k, e_k and Z_k go to zero.

As in [64, p.90], let $m(t)$ denote the unique value of k such that $t_k \leq t < t_{k+1}$ for $t \geq 0$, and set $m(t) = 0$ for $t < 0$, where the time scale t_k is defined as follows: $t_0 = 0$, $t_k = \sum_{i=0}^{k-1} \alpha_i$.

Define the shifted continuous-time interpolation $\theta^k(t)$ of θ_k as follows:

$$\theta^k(t) = \theta_k + \sum_{i=k}^{m(t+t_k)-1} \alpha_i \widetilde{sf}(\theta_i) + \sum_{i=k}^{m(t+t_k)-1} \alpha_i b_i + \sum_{i=k}^{m(t+t_k)-1} \alpha_i e_i + \sum_{i=k}^{m(t+t_k)-1} \alpha_i Z_i. \tag{5.18}$$

Define $B^k(t) = \sum_{i=k}^{m(t+t_k)-1} \alpha_i b_i$, and define $M^k(t)$ and $Z^k(t)$ similarly, with e_k and Z_k respectively in place of b_k. Since $\theta^k(t)$ is piecewise constant, we can rewrite (5.18) as

$$\theta^k(t) = \theta_k + \int_0^t \widetilde{sf}(\theta^k(s))ds + B^k(t) + M^k(t) + Z^k(t) + \rho^k(t), \tag{5.19}$$

where $\rho^k(t) = \int_{t_{m(t+t_k)}}^t \widetilde{sf}(\theta^{m(t+t_k)}(s))ds \leq \alpha_{m(t+t_k)} |\widetilde{sf}(\theta_{m(t+t_k)})|$. Note that $\rho^k(t)$ is due to the replacement of the first summation in (5.18) by an integral, and $\rho^k(t) = 0$ at the jump times t_k of the interpolated process, and $\rho^k(t) \to 0$, since α_k goes to zero as k goes to infinity.

We use similar conditions as for Theorem 5.3.1 in [64, pp. 88-108], which are listed as follows:

(5.A.1) $\alpha_k \to 0$, $\sum \alpha_k = \infty$;

(5.A.2) the feasible region H is a hyperrectangle. In other words, there are numbers $a_i < b_i$, $i = 1, \ldots, r$, such that $H = \{x : a_i \leq x_i \leq b_i\}$;

(5.A.3) for some positive number T_1, $\lim_{k\to\infty} \sup_{|t|\leq T_1} |B^k(t)| = 0$ w.p. 1;

(5.A.4) for some positive number T_2, $\lim_{k\to\infty} \sup_{|t|\leq T_2} |M^k(t)| = 0$ w.p. 1.

For $x \in H$ satisfying (5.A.2), define the set $C(x)$ as follows. For $x \in H^0$, the interior of H, $C(x)$ contains only the zero element; for $x \in \partial H$, the boundary of

H, let $C(x)$ be the infinite convex cone generated by the outer normals at x of the faces on which x lies [64, p. 77].

Proposition 5.2.1 *For the algorithm given by (5.17), assume $\widetilde{sf}(\theta_k)$ is uniformly bounded and (5.A.1)-(5.A.4). Suppose that $f(\cdot)$ is continuous and convex, but not constant. Consider the differential inclusion*

$$\dot{\theta} \in -\partial f(\theta) + z, \quad z(t) \in -C(\theta(t)), \tag{5.20}$$

and let S_H denote the set of stationary points of (5.20), i.e. points in H where $0 \in -\partial f(\theta) + z$. Then, for almost all ω, $\{\theta_k(\omega)\}$ converges to a point in S_H, which attains the minimum of f.

Proof: see Appendix. □

5.2.3 Convergence of the SPSA algorithm

In this section, we prove convergence of the SPSA algorithm given by (5.15) using Proposition 5.2.1.

We first rewrite the SPSA algorithm given by (5.15) as follows:

$$\theta_{k+1} = \theta_k - \alpha_k \widetilde{sg}(\theta_k) + \alpha_k [\Delta_k^{-1}]\delta_k + \alpha_k (I - [\Delta_k^{-1}]\Delta_k^T)\widetilde{sg}(\theta_k) + \alpha_k [\Delta_k^{-1}] \frac{\phi_k^- - \phi_k^+}{2c_k} + \alpha_k \tilde{Z}_k, \tag{5.21}$$

where $\tilde{Z}_k$ is the reflection term.

Note that the counterparts of b_k and e_k in (5.17) are $[\Delta_k^{-1}]\delta_k$ and $e_{r,k} + e_{o,k}$, respectively, where the latter quantity is decomposed into a random direction noise term $e_{r,k} := (I - [\Delta_k^{-1}]\Delta_k^T)\widetilde{sg}(\theta_k)$ and an observation noise term $e_{o,k} := \frac{[\Delta_k^{-1}]}{2c_k}(\phi_k^- - \phi_k^+)$.

Lemma 5.2.2 *Assume 1) $E[\frac{\Delta_{k,i}}{\Delta_{k,j}} \mid \Delta_0, \ldots, \Delta_{k-1}, \theta_0, \ldots, \theta_{k-1}] = 0$, $i \neq j$; 2) $\widetilde{sg}(\theta_k)$ is uniformly bounded by a finite B. Then, $e_{r,k} = (I - [\Delta_k^{-1}]\Delta_k^T)\widetilde{sg}(\theta_k)$ is a martingale difference.*

Proof: Using a similar argument as in [48], define $M_k^B = \sum_{l=0}^{k}(I-[\Delta_l^{-1}]\Delta_l^T)\widetilde{sg}(\theta_l)$, so $e_{r,k} = M_k^B - M_{k-1}^B$.

$$\begin{aligned} E[M_{k+1}^B \mid M_l, l \le k] &= E[M_k^B + (I - [\Delta_{k+1}^{-1}]\Delta_{k+1}^T)\widetilde{sg}(\theta_{k+1}) \mid M_l, l \le k] \\ &= M_k^B + E[e_{r,k+1} \mid M_l, l \le k], \end{aligned}$$

and the absolute value of $E[e_{r,k+1} \mid M_l, l \le k]$

$$\begin{aligned} |E[e_{r,k+1} \mid M_l, l \le k]| &\le |B| \cdot \; |E[(I - [\Delta_{k+1}^{-1}]\Delta_{k+1}^T)\bar{\mathbf{1}} \mid M_l, l \le k]| \\ &= |B| \cdot \; E\left|\left[\begin{bmatrix} 0 & \frac{\Delta_{k+1,1}}{\Delta_{k+1,2}} & \cdots & \frac{\Delta_{k+1,1}}{\Delta_{k+1,r}} \\ \frac{\Delta_{k+1,2}}{\Delta_{k+1,1}} & 0 & \cdots & \frac{\Delta_{k+1,2}}{\Delta_{k+1,r}} \\ \vdots & \vdots & & \vdots \\ \frac{\Delta_{k+1,r}}{\Delta_{k+1,}} & \cdots & \frac{\Delta_{k+1,r}}{\Delta_{k+1,2}} & 0 \end{bmatrix} \bar{\mathbf{1}} \mid M_l, l \le k \right]\right| \\ &= 0, \end{aligned}$$

where $\bar{\mathbf{1}}$ is a column vector with each element being 1. Thus, $e_{r,k}$ is a martingale difference. □

Lemma 5.2.3 *Assume Δ_k is independent of $\theta_k, \theta_{k-1} \dots, \theta_0$. Then $\phi_k^- - \phi_k^+$ is a martingale difference.*

Proof: We know $E[F(\theta_k \pm c_k\Delta_k, \chi_k) \mid \theta_k, \Delta_k] = f(\theta_k \pm c_k\Delta_k)$ since $E[F(\theta, \chi) \mid \theta] = f(\theta)$. Thus, $E[\phi_k^- - \phi_k^+ \mid \theta_k, \Delta_k] = 0$.

Note that $\phi_k^- - \phi_k^+$ depends on $\phi_{k-1}^- - \phi_{k-1}^+, \dots, \phi_0^- - \phi_0^+$ only via θ_k and Δ_k. So

$$E[\phi_k^- - \phi_k^+ \mid \theta_k, \Delta_k, \phi_{k-1}^- - \phi_{k-1}^+, \dots, \phi_0^- - \phi_0^+] = E[\phi_k^- - \phi_k^+ \mid \theta_k, \Delta_k] = 0,$$

which implies that $E[\phi_k^- - \phi_k^+ \mid \phi_{k-1}^- - \phi_{k-1}^+, \dots, \phi_0^- - \phi_0^+] = 0$, i.e., $\phi_k^- - \phi_k^+$ is a martingale difference. □

For the observation noise term $e_{o,k}$, we assume the following conditions [64, pp. 109-113]:

(5.B.1) $\phi_k^- - \phi_k^+$ is a martingale difference;

(5.B.2) For each $\mu > 0$, $\sum_k e^{-\mu c_k^2/\alpha_k} < \infty$;

(5.B.3) For some $T > 0$, there is a $c_1(T) < \infty$ such that for all k,

$$\sup_{k \le i < m(t_k+T)} \frac{\alpha_i/c_i^2}{\alpha_k/c_k^2} \le c_1(T);$$

(5.B.4) There is a $K < \infty$ such that for a small γ, all k, and each component $(\phi_k^- - \phi_k^+)_j$ of $(\phi_k^- - \phi_k^+)$,

$$E_k e^{\gamma(\phi_k^- - \phi_k^+)_j} \le e^{\gamma^2 K/2}.$$

Lemma 5.2.4 *Assume (5.A.1), (5.B.1) - (5.B.4) and assume that $[\Delta_k^{-1}]$ is uniformly bounded. Then, for some positive T, $\lim_{k\to\infty} \sup_{|t|\le T} |M^{o,k}(t)| = 0$ w.p. 1, where $M^{o,k}(t) := \sum_{i=k}^{m(t+t_k)-1} \alpha_i e_{o,i}$.*

Proof: Define $\tilde{M}^k(t) := \sum_{i=k}^{m(t+t_k)-1} \frac{\alpha_i}{2c_i}(\phi_i^- - \phi_i^+)$. By (5.A.1) and (5.B.1) - (5.B.4), there exists a positive T such that $\lim_{k\to\infty} \sup_{|t|\le T} |\tilde{M}^k(t)| = 0$ w.p. 1, following the same argument as the one in the proof of Theorem 5.3.2 and Theorem 5.3.3 in [64, pp. 108-110].

Suppose $[\Delta_k^{-1}]$ is uniformly bounded by D, then $|M^{o,k}(t)| \le D|\tilde{M}^k(t)|$.

Thus, $\lim_{k\to\infty} \sup_{|t|\le T_1} |M^{o,k}(t)| = 0$ w.p. 1. □

Proposition 5.2.2 *Consider the SPSA algorithm (5.15), where $\widetilde{sg}(\theta_k)$ is uniformly bounded and $\widetilde{sg}(\theta_k) \in \partial f(\theta_k)$. Assume (5.A.1)-(5.A.2), (5.B.2)-(5.B.4), and assume that Δ_k is bounded, $[\Delta_k^{-1}]$ is uniformly bounded, Δ_k is independent of $\theta_k, \theta_{k-1} \ldots, \theta_0$, and $E[\frac{\Delta_{k,i}}{\Delta_{k,j}} \mid \Delta_0, \ldots, \Delta_{k-1}, \theta_0, \ldots, \theta_{k-1}] = 0$ for $i \ne j$. Suppose that $f(\cdot)$ is continuous and convex, but not constant. Consider the differential inclusion*

$$\dot{\theta} \in -\partial f(\theta) + z, \quad z(t) \in -C(\theta(t)), \tag{5.22}$$

and let S_H denote the set of stationary points of (5.22), i.e. points in H where $0 \in -\partial f(\theta) + z$. Then, for almost all ω, $\{\theta_k(\omega)\}$ converges to a point in S_H, which attains the minimum of f.

Proof: Since $[\Delta_k^{-1}]$ are uniformly bounded and $\lim_{k\to\infty} \delta_k = 0$ by Lemma 5.2.1, $\lim_{k\to\infty}[\Delta_k^{-1}]\delta_k = 0$, which implies that (5.A.3) holds.

By Lemma 5.2.3, (5.B.1) holds. By Lemma 5.2.4, $\lim_{k\to\infty} \sup_{|t|\le T} |M^{o,k}(t)| = 0$ w.p. 1. So (5.A.4) holds for $e_{o,k}$.

Since Δ_k is bounded and both $\widetilde{sg}(\theta_k)$ and $[\Delta_k^{-1}]$ are uniformly bounded, $E|e_{r,k}|^2 < \infty$. Using the martingale convergence theorem and Lemma 5.2.2, we get $\lim_{k\to\infty} \sup_{|t|\le T} |M^{r,k}(t)| = 0$ w.p. 1, where $M^{r,k}(t) := \sum_{i=k}^{m(t+t_k)-1} \alpha_i e_{r,i}$. So (5.A.4) holds for $e_{r,k}$.

So all conditions of Proposition 5.2.1 are satisfied, and all its conclusions hold. □

5.3 Discussion

In this chapter, we developed two SPSA algorithms for weighted cost problems and carried out numerical experiments. Compared to the simulation-based algorithms in Chapter 4, the SPSA algorithms are easy to use and general applicable, though less efficient.

Then we used convex analysis to establish convergence of constrained SPSA for the setting in which the objective function is not necessarily differentiable. The analysis can be extended to SPSA algorithms with nondifferentiable constraints, as well as other stochastic approximation algorithms for nondifferentiable function optimization. In particular, we intend to consider global optimization of nondifferentiable functions along the lines of [41].

Chapter 6

Fab-Level Decision-Making Application

In this chapter, we tackle the fab-level decision making MDP problem presented in Chapter 2. From its "testbed" example of Section 2.2.5, we have seen that the fab-level decision making MDP problem suffers from both "curse of dimensionality" and "curse of modeling". In order to solve such problems, our methodology is first to derive the structure of optimal policies for some special cases, then to use this structure to construct parameterized heuristic policies for more general cases and implement simulation-based algorithms to determine parameters of the heuristic policies. Sometimes, we also use simulation-based policy iteration to gain insight of optimal policies.

First, in Section 6.1, we present a simple example of the fab-level decision making problem, and apply simulation-based policy iteration to this example. Numerical experiments indicate that a near-optimal policy, which matches intuition, can often be obtained within ten iterations. The resulting policies also give us some insight of optimal policies. However, this algorithm cannot be extended to more complicated examples of the fab-level decision making problem, since the control space explodes very quickly.

The crucial part of our methodology is to find classes of parameterized policies that are near optimal. Then, we can use simulation-based algorithms introduced in Chapter 4 and Chapter 5 to estimate the optimal parameters of the heuristic policy. Hence, in the second part of this chapter, we focus on deriving the structure of an optimal policy for a special one-machine, two-product case. Note that the resulting policies do not depend on how elements in the states and the control vectors are discretized.

Finally, we discuss how to implement simulation-based algorithms on the testbed example, including reformulation of the finite horizon problem to a weighted cost-to-go problem, and parameterization of the structural optimal policies for the one-machine, two-product case and several more complex cases.

6.1 Simulation-Based Policy Iteration for a Simple Example

In order to get a better understanding of how the MDP model of the fab-level decision-making problem would be specified in practice and how to use simulation-based algorithms to obtain some insight into an optimal policy, we provide here a very simple example of a fab with just two machines and producing two products that each have two operations (litho an etch). We will cast the simple example as an infinite horizon discounted cost problem, and apply simulation-based policy iteration to this problem.

6.1.1 Problem Setup

Similar to the one-machine, two-product example, in this two-machine, two-product example, the fab is characterized as follows:

- two products: “A” and “B”;
- two operations on each: “litho” and “etch”, distinguished by product;
- two machines – litho or etch – could be flexible (able to do the respective

operation on both products A and B) or dedicated (only able to do the respective operation on one of A or B);

- operation times, which depend on the product and the machine.

We begin by defining products, operations, and machines.

Products l are chosen from the set

$$\mathcal{P}_t = \{0, A, B\},$$

where 0 corresponds to 'no product'.

Operations i are chosen from the set $\mathcal{X}_t = \{1, 2, 3, 4\}, \mathcal{N}_t = 4$, where

$$\begin{aligned} 1 &\longleftrightarrow \text{lithoA}, \\ 2 &\longleftrightarrow \text{etchA}, \\ 3 &\longleftrightarrow \text{lithoB}, \\ 4 &\longleftrightarrow \text{etchB}. \end{aligned}$$

Thus, operations 1 and 2 correspond to operations on product A, whereas operations 3 and 4 correspond to operations on product B.

Machines are words w chosen from the set $\mathcal{Z}_t$:

$$\begin{aligned} 01 &= \text{machine dedicated to lithoA operation}, \\ 02 &= \text{machine dedicated to etchA operation}, \\ 03 &= \text{machine dedicated to lithoB operation}, \\ 04 &= \text{machine dedicated to etchB operation}, \\ 013 &= \text{flexible litho machine}, \\ 024 &= \text{flexible etch machine}, \\ &\cup \; \{012, 034, 014, 023, 0123, 0124, 0134, 0234, 01234\}, \end{aligned}$$

where 0 corresponds to ‘no operation’ and the last set’s elements do not correspond to feasible machines in the real model.

Other model input (system) parameters that must be defined are $C_{(l,i),w}$ and $F_{(l,i),w}$, which essentially specify the operation times for a particular product on a particular machine.

$$F_{(l,i),w} = \begin{cases} 1 & (A,1),013; (A,2),024; (B,3),013; (B,4),024; \\ & (A,1),01; (A,2),02; (B,3),03; (B,4),04; \\ 0 & \text{otherwise,} \end{cases}$$

i.e., all products require a single operation on the appropriate machine.

$$C_{(l,i),w} = \begin{cases} 1 & (A,1),013; (A,2),024; \\ 0.5 & (B,3),013; (B,4),024; \\ 1.2 & (A,1),01; (A,2),02; \\ 0.6 & (B,3),03; (B,4),04; \\ 0 & \text{otherwise,} \end{cases}$$

i.e., a flexible etch or litho machine completes one operation on product A in an hour, product B takes twice as long on both operations, and a flexible machine is 20% slower than a dedicated one (e.g., 60 minutes versus 50 minutes for product A).

For simplicity, we will take $K_w = 1$ for all w, i.e., availability is 100% for all machines.

Assumptions

Following [66], the following assumptions are made:

- During the decision horizon, no machine is purchased, discarded, or sent to reserve, and no maintenance is required. The only actions are to switch flexible machines between different products.

- For each type of machine (litho or etch), no more than one machine can be switched from one product and/or operation to another in a period.
- Products A and B are operated in whole unit and half units, respectively.
- The inventory warehouses for products A and B have capacities of 1 and 0.5 units, respectively.
- There is a limit on backlogged demand of 1 and 0.5 units for product A and B, respectively. Demand exceeding backlogging limits is lost.
- The demand process for a given product is independent and identically distributed from period to period, and demand processes are mutually independent between products.

States

Under the above assumptions, the state vector of our MDP model takes the form

$$\overline{X}(t) = \{(X_{(A,1),013}(t), X_{(A,2),024}(t), I_A(t), I_B(t)\},$$

where the first and second components are, respectively, the litho and etch capacities allocated to product A, and the third and fourth components are, respectively, the inventory levels of products A and B. Note that the capacity allocated to product B is simply the remainder of total machine capacity for each machine type (litho or etch), because we have assumed for this two-machine, two product example that no capacity is ever put into reserve, thus reducing the dimensionality of the state vector from six dimensions to four components. Under our assumptions, the components of the state vector take values in the following sets:

$$\begin{aligned} X_{(A,1),013}(t) = 2 - X_{(B,3),013}(t) &\in \{0,1,2\}, \\ X_{(A,2),024}(t) = 2 - X_{(B,4),024}(t) &\in \{0,1,2\}, \end{aligned}$$

$$\begin{aligned} I_A &\in \{-1, 0, 1\}, \\ I_B &\in \{-0.5, 0, 0.5\}, \end{aligned}$$

and thus the total number of possible states is 81 and the throughput for each product is given as follows:

$$TP_A(t) = \min\left\{X_{(A,1),013}(t), X_{(A,2),024}(t)\right\},$$

$$TP_B(t) = 0.5 \min\left\{X_{(B,3),013}(t), X_{(B,4),024}(t)\right\},$$

where the argmin gives the bottleneck operation for the product (1 or 2 for A; 3 or 4 for B).

Note that for our problem, states can be divided into *state groups*, which are defined as the sets of those states that have the same capacity allocation for both $X_{(A,1),013}(t)$ and $X_{(A,2),024}(t)$. In other words, if two states differ only in their product inventory levels, we say they belong to the same state group.

Actions

For the action vector, we have two formulations. One formulation corresponds to the action vector with four-dimensional form

$$\{V_{013}^{(A,1),(B,3)}(t), V_{024}^{(A,2),(B,4)}(t), V_{013}^{(B,3),(A,1)}(t), V_{024}^{(B,4),(A,2)}(t)\},$$

where the first component is the litho capacity moved from product A to product B, the second component is the etch capacity moved from product A to product B, the third component is the litho capacity moved from product B to product A, and the fourth component is the etch capacity moved from product B to product A. In this formulation, $V_w^{(l,i),(m,j)}(t) = 0$, if some type w capacity is switched over from product m and operations of type j to product l and operations of type i.

Given that we consider only allocation and not expansion, and do not allow capacity to be sent to reserve, the action vector can be reduced to the twodimensional form

$$\overline{U}(t) = (V_{013}^{(A,1),(B,3)}, V_{024}^{(A,2),(B,4)}),$$

which is the other formulation, since the amount of capacity moved from A to B is the negative of that moved from B to A:

$$V_{013}^{(A,1),(B,3)}(t) = -V_{013}^{(B,3),(A,1)}(t) \in \{-1, 0, 1\},$$
$$V_{024}^{(A,2),(B,4)}(t) = -V_{024}^{(B,4),(A,2)}(t) \in \{-1, 0, 1\}.$$

We will label the resulting nine possible actions as follows: A1=(-1,-1), A2=(-1,0), A3=(-1,+1), A4=(0,+1), A5=(0,0), A6=(0,+1), A7=(+1,-1), A8(+1,0), A9=(+1,+1). However, note that for a given state group, not all actions are admissible.

We have the following state equations:

$$X_{(A,1),013}(t+1) = X_{(A,1),013}(t) - V_{013}^{(A,1),(B,3)}(t), \tag{6.1}$$

$$X_{(A,2),024}(t+1) = X_{(A,2),024}(t) - V_{024}^{(A,2),(B,4)}(t), \tag{6.2}$$

$$I_A(t+1) = I_A(t) + TP_A(t) - d_A(t), \tag{6.3}$$

$$I_B(t+1) = I_B(t) + TP_B(t) - d_B(t). \tag{6.4}$$

Demand Distribution and Transition Probability

The demands for product A and B are modeled as Bernoulli process, in which demand for product A can only be 1 or 2 units and demand for product B can only be 0.5 and 1 unit. The probabilities of demand for product A to be 1 and of demand for product B to be 0.5 are denoted as p_A and p_B.

For each action, we first build a state group transition probability matrices involving only state elements in the form of $X_{(A,j),w}(t)$, based on state equations (6.1) and (6.2). It is not difficult to see that each entry of state group transition probability matrices is either zero or one.

To get a complete transition probability matrices, we fill in each entry of the state group matrices: each entry with value zero is replaced by a zero matrix; and each entry with value one is replaced by a joint transition probability matrix $P(A, B)$, which only involves the state elements $I_l(t)$.

$P(A, B)$ is constructed as follows. Knowing demand distributions, we first build marginal transition probability matrices $P(A)$ and $P(B)$ (3×3) for each product A and B for different throughputs, based on the state equations (6.3) and (6.4), respectively. Then, we build $P(A, B)$ (9×9) with regard to each throughput pairs:

$$P(A, B) = P(A) \otimes P(B)$$

where $\otimes$ is the kronecker product. Suppose X and Y are two matrices, the result of $X \otimes Y$ is a large matrix formed by taking all possible products between the elements of X and those of Y. For example, if X is 2 by 3, then $X \otimes Y$ is

$$\begin{bmatrix} X(1,1) * Y & X(1,2) * Y & X(1,3) * Y \\ X(2,1) * Y & X(2,2) * Y & X(2,3) * Y \end{bmatrix}.$$

Cost Structure

Here we only consider a linear cost structure. So the inventory cost and backlogging cost are proportional to the inventory level; the operation cost is proportional to the allocated capacity for different products on different machines; the switch-over cost is proportional to the switched capacity, which is also an action variable.

Correspondingly, the parameters are unit inventory cost and backlogging cost for both products A and B; unit operating cost on the litho machine for product A, on the litho machine for product B, on the etch machine for product A, and on the etch machine for product B; and the unit switch-over cost on both litho and etch machines.

Objective Function

We formulate the two-machine, two-product example as a discounted cost problem, where the objective function is:

$$J_\pi(i) = \lim_{T\to\infty} E\left\{\sum_{t=0}^{T-1} \alpha^t g(\overline{X}(t), \mu_t(\overline{X}(t))) \mid \overline{X}(0) = i\right\}. \tag{6.5}$$

where $\alpha \in (0,1)$ is the discount factor.

6.1.2 Simulation-Based Policy Iteration Algorithm

The simulation-based policy iteration algorithm for the two-machine, two-product example is as follows:

1. Set $k = 0$, select an arbitrary stationary policy μ^k.

2. **Policy evaluation**: Calculate approximate cost-to-go function $\tilde{J}_\mu(i, r)$, where r is a vector of tunable parameters:

   (i) Generate $M(i)$ sample trajectories with length T for each initial state i, where the mth sample path starting with state i is denoted by $c(i, m)$;

   (ii) solve
   $$\min_r \sum_i \sum_{m=1}^{M(i)} (\tilde{J}(i, r) - c(i, m))^2, \tag{6.6}$$
   to obtain the parameter vector $\bar{r}$ for calculating the approximation $\tilde{J}_\mu(i, \bar{r})$ of the cost function.

3. **Policy improvement**:

   Choose a new policy μ^{k+1} that is greedy with respect to the approximate cost-to-go,
   $$\mu^{k+1}(i) = argmin_{u\in U(i)} \sum_{j=0}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, \bar{r})), \quad i = 0, \ldots, n-1,$$

4. If $\mu^{k+1} = \mu^k$, then stop and set $\mu^* = \mu^k$; otherwise increment k by 1 and return to step 2.

The above simulation-based policy iteration algorithm presented closely follows approximate policy iteration in [4].

For the approximate cost-to-go function $\tilde{J}_\mu(i,r)$, first we need to choose an *approximation architecture*, that is, a certain functional form involving a number of free parameters. Broadly, approximation architectures can be classified into two main categories: linear and nonlinear. A linear architecture is of the general form

$$\tilde{J}(i,r) = \sum_{k=0}^{K} r(k)\phi_k(i),$$

where $r(k)$,$k = 1,\ldots,K$, are the components of the parameter vector r, and ϕ_k are fixed, easily computable functions. A common nonlinear architecture is a multilayer perceptron [4] with a single hidden layer which has the following form

$$\tilde{J}(i,r) = \sum_{k=0}^{K} r(k)\sigma(\sum_{l=1}^{L} r(k,l)x_l(i)),$$

where $r(k,l)$ and $r(k)$ are coefficients for the input layer and output layer, respectively; $\sigma(\cdot)$ is a sigmoidal function such as the logistic function

$$\sigma(\xi) = \frac{1}{1+e^{-\xi}}.$$

It is often the case that the approximation architecture is too complicated for state representation, and one considers the use of some structural pieces to represent states. These structural pieces are called *features*, which are fed into the approximation architecture instead of the state itself. Usually, these features are handcrafted, based on the particular problem. Some example of features include state variables, heuristic cost-to-go and/or past cost-to-go.

Since we are dealing with an infinite horizon problem, and simulation can only run for a finite number of steps, we need to make sure that the cost of the finite approximation is close enough to that of the infinite horizon cost. In fact, if we choose T as the sample length, the error between the expected value of the resulting sample cost-to-go and J^μ is within $\frac{G\alpha^T}{(1-\alpha)}$, where G is an upper bound on $|g(i,u,j)|$. So, if we specify the error requirement, the choice of T follows. With T set to be sufficiently large, the effects of using a finite trajectory can be made arbitrarily small. We can choose $M(i)$ by balancing exploration and exploitation.

One set of computational methods for the least squares problem (6.6) is to simulate a number of trajectories, collect the results, formulate the least squares problem and then solve it in batch mode.

Another way to solve the least square problem is the incremental gradient method. Given a sample state trajectory $(i_0, i_1, \ldots, i_N)$ generated using the policy μ, the parameter vector r is updated by

$$r := r - \gamma \sum_{k=0}^{N-1} \nabla \tilde{J}(i_k, r)(\tilde{J}(i_k, r) - \sum_{m=k}^{N-1} g(i_m, \mu(i_m), i_{m+1})),$$

where γ is a stepsize. A key advantage of incremental algorithms is that they allow us to decide whether or not to simulate more trajectories, depending on the quality of the results obtained so far.

6.1.3 Numerical Experiments

To illustrate the effectiveness of simulation-based policy iteration, a case study is provided.

The simulation-based policy iteration is implemented as follows:

- Linear architecture is chosen for cost-to-go approximation architecture.
- Features are state variables $\{(X_{(A,1),013}(t), X_{(A,2),024}(t), I_A(t), I_B(t)\}$ and their squares, plus $\phi_1(x) = 1$; there are nine features.
- For the sample paths, random variates are constructed using the inverse transform method [67]; the simulation length and number of replication are chosen to be 100 and 20, balancing between accuracy and simulation speed.

Model specifications	
$D_A = 1$	0.4
$D_A = 2$	0.6
$D_B = 0.5$	0.7
$D_B = 1$	0.3
unit inventory cost for product A	2
unit inventory cost for product B	1
unit backlog cost for product A	10
unit backlog cost for product B	5
unit operating cost on litho machine for product A	0.2
unit operating cost on litho machine for product B	0.2
unit operating cost on etch machine for product A	0.1
unit operating cost on etch machine for product B	0.1
unit switch cost on litho machine	3
unit switch cost on etch machine	3
discounted factor	0.95

Table 6.1: Model Specifications

- Batch mode least squares is adopted.

The example simulated has the specifications shown in Table 6.1.

Starting with an initial policy (see the policy in the first iteration of Figure 6.1, which is an arbitrarily chosen admissible policy), after 7 iterations the program terminates with a near optimal policy. See Figures 6.1 and 6.2 for the policy trajectories, respectively, and cost trajectories at each iteration.

Compared to exact policy iteration for the same case shown in Figure 6.3, there are several interesting points:

- Simulation-based policy iteration takes 7 iterations to obtain its optimal policy, whereas exact policy iteration takes only 4 iterations to obtain the optimal policy.

- The near optimal policy is different from the optimal policy. Even if we use the optimal policy as the initial policy, simulation-based policy iteration still converges to the near optimal policy. This is due to the feature

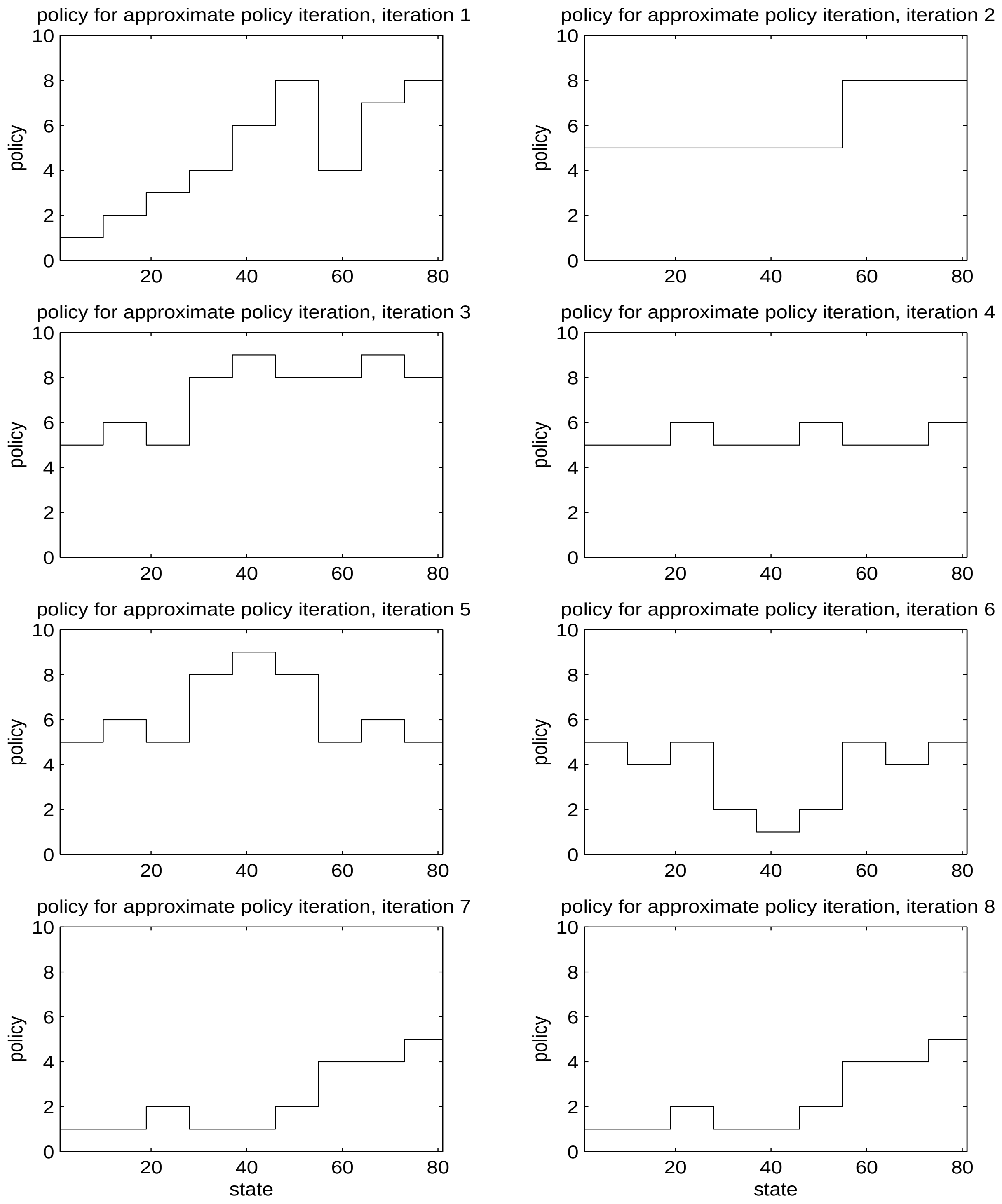


Figure 6.1: Policy for Simulation-Based Policy Iteration

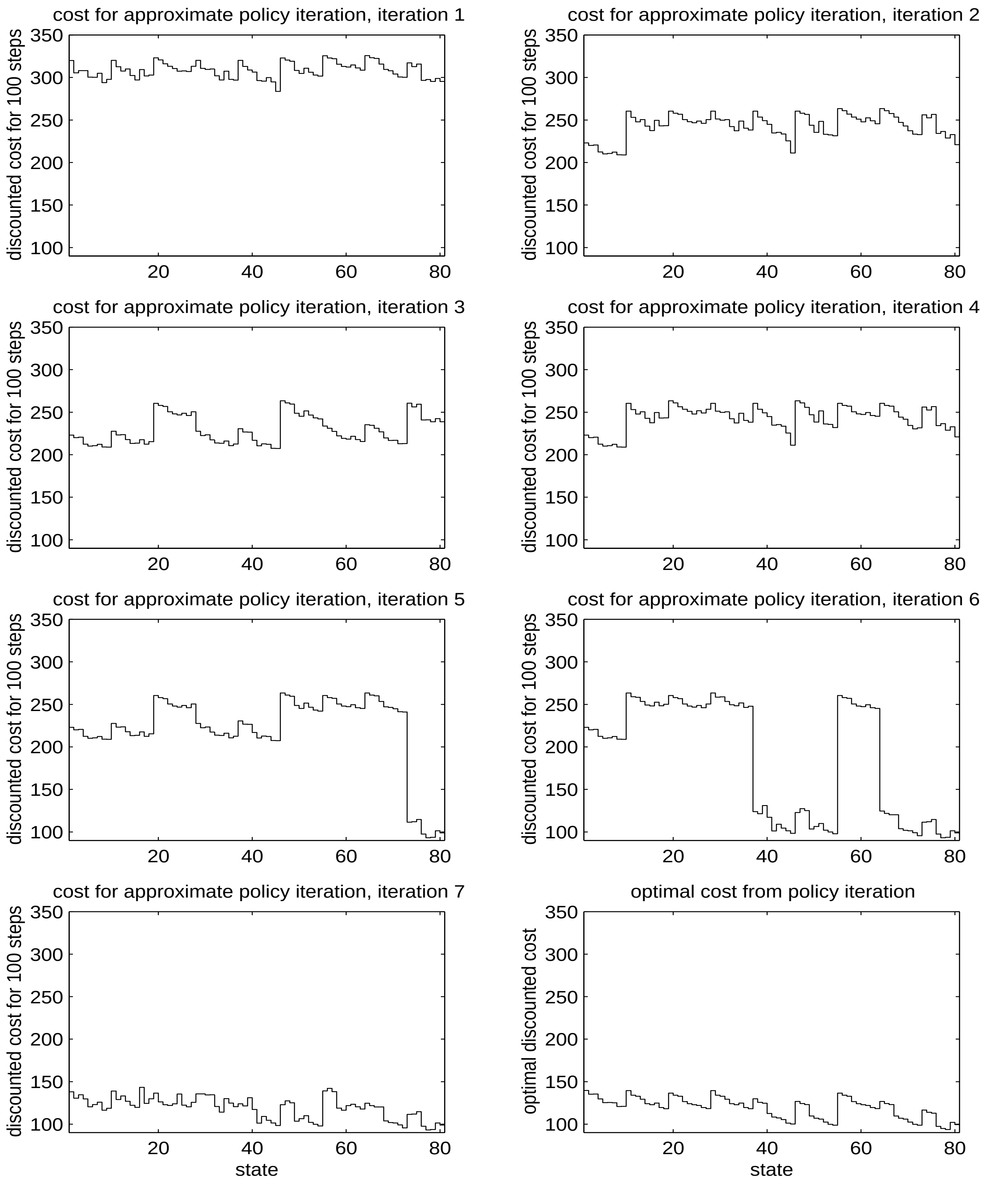


Figure 6.2: Cost Function for Simulation-Based Policy Iteration

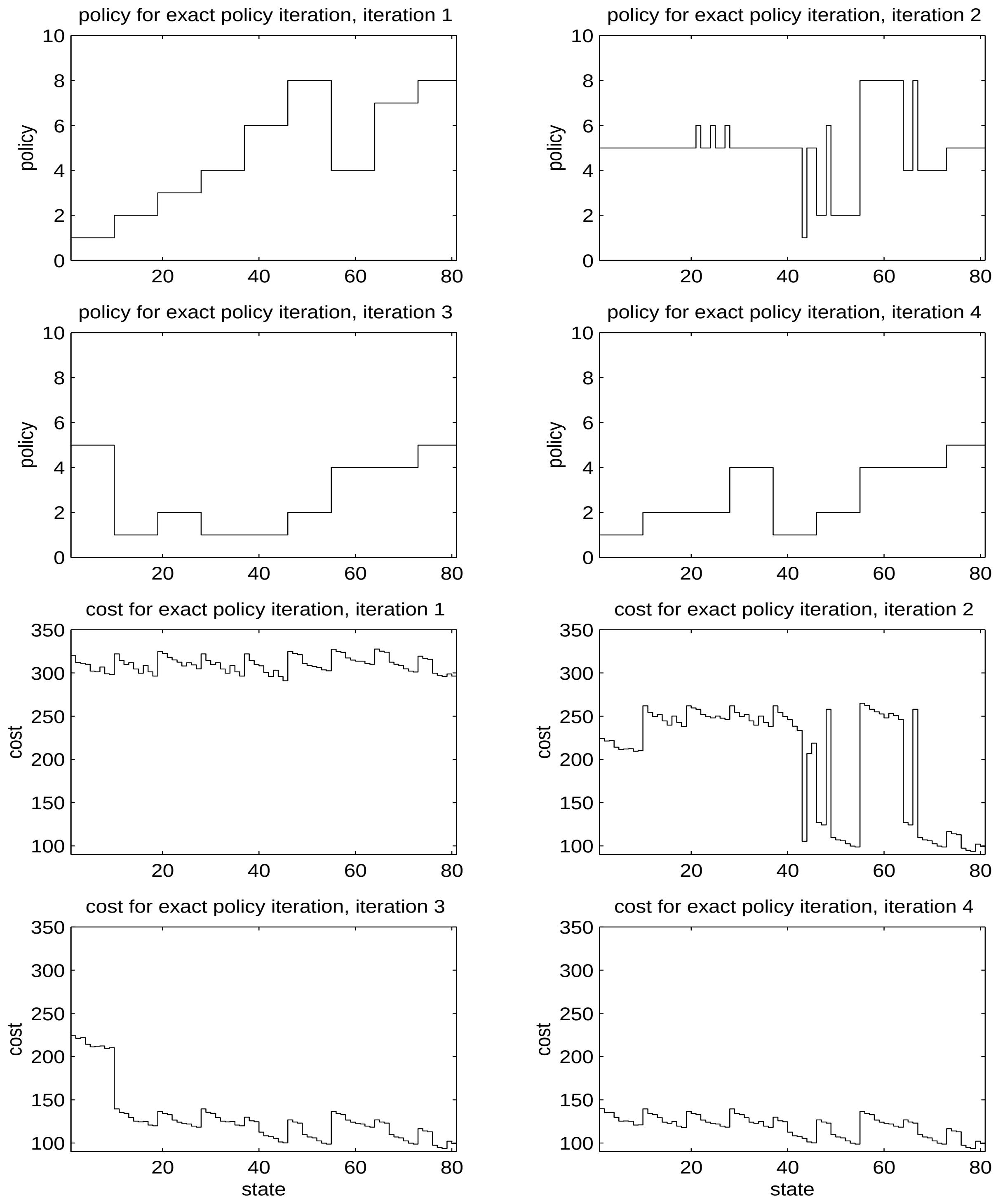


Figure 6.3: Policy and Cost Function for Exact Policy Iteration

approximation.

- Although the paths to the optimal policy are different for simulation-based policy iteration and policy iteration, there are some similarities. It appears that simulation-based policy iteration improves slower than exact policy iteration, with the same trend.
- The cost-to-go of approximate policy iteration stalls at some policy for three steps, which cause the delay.

6.2 Structure of Optimal Policy for a One-Machine, Two-Product Example

In this section, we show how to find the structure of an optimal policy for a one-machine, two-product example of the fab-level decision-making problem.

In Chapter 4 and 5, we developed several simulation-based algorithms and carried out numerical experiments on a parking example. However, for the parking example, we know the structure of an optimal policy, whereas this is not the case for the fab-level decision-making MDP. Hence, we first need to find the optimal policy structure or heuristic policy structure before we apply simulation-based algorithms.

For MDP problems, there are a few cases whose optimal policy structures are known: the parking problem used in Chapter 4 and Chapter 5, the one product inventory control problem with (s, S) optimal policy, and some simple maintenance problems with threshold type optimal policies, etc. As for many other cases, it is difficult to find the structures of optimal policies. In the case of the fab-level MDP problem, we can interpret it as a combination of capacity allocation and multiproduct inventory control problem, so it is even harder to obtain a simple structure of optimal policies. Here, we only try to derive the structure of an optimal policy for a one-machine, two-product case. In fact, even for such small example, we can only get results for some special cases.

We use the backwards induction method of dynamic programming to derive the structure of an optimal policy for the one-machine, two-product example. Assuming the cost structure to be linear and demands to be deterministic, and under some constraints, we can show that the function to be minimized at each stage of dynamic programming is piecewise linear convex. Then, using properties of piecewise linear convex functions, we divide the state space into several partitions, obtain an optimal policies for the partitions, and combine the optimal policy for each partition to form an optimal policy for entire state space. We illustrate the details for time $T-1$ and $T-2$, present results of some scenarios, and verify policy optimality by numerical experiments.

If demands are stochastic, we can show that the function to be minimized at each stage of dynamic programming is convex. Using the properties of convex functions, we can also obtain an optimal policy for some special cases. For some other cases, if demands can only take a finite number of values, we can obtain sub-optimal policy based on results from the deterministic cases. However, for more complicated stochastic cases, constructing near-optimal policies with simple structure is still a subject of ongoing research.

6.2.1 Problem Setup

Here, we consider a special case of the fab-level decision-making problem with two products, one machine and no capacity expansion or discard. Specifically, this problem can be characterized as follows:

- two products: "A" and "B";
- one machine, say etch machine, a flexible machine which is able to perform both A and B operations;

- two operations: "etchA" and "etchB";
- assume no capacity expansion or discard, and no sending from or to reservation;
- linear cost structure;
- demands for products are mutually independent, and are independent over time.

We begin by defining products, operations, and machines.
Products l are chosen from the set $\mathcal{P}_t = \{0, A, B\}$, where 0 corresponds to 'no product'.
Operations i are chosen from the set $\mathcal{X}_t = \{1, 2\}, \mathcal{N}_t = 2$, where

$$\begin{aligned} 1 &\longleftrightarrow \text{etchA}, \\ 2 &\longleftrightarrow \text{etchB}. \end{aligned}$$

Machines are words w chosen from the set $\mathcal{Z}_t = \{01, 02, 012\}$, where

$$\begin{aligned} 01 &= \text{machine dedicated to etchA operation}, \\ 02 &= \text{machine dedicated to etchB operation}, \\ 012 &= \text{flexible etch machine}, \end{aligned}$$

and 0 corresponds to 'no operation'.

Other model input parameters that must be defined are $C_{(l,i),w}$ and $F_{(l,i),w}$. Here we use production rates $P_{(l,i),w} = C_{(l,i),w}/F_{(l,i),w}$ for brevity.
The state vector at time t is

$$\overline{X}(t) = \{X_{(A,1),012}(t), X_{(B,2),012}(t), I_A(t), I_B(t)\},$$

and the action vector at time t is

$$\overline{U}(t) = \{V_{012}^{(A,1),(B,2)}(t)\}.$$

For simplicity, we will drop "012" and use "$A \to B$" to replace "$(A, 1), (B, 2)$" henceforth. Note that $V^{(A\to B)}(t) = -V^{(B\to A)}(t)$.

The total cost we want to minimize is

$$\begin{aligned} J^0 = E \ [\ \textstyle\sum_{t=1}^{T} \ \{ \ & C^c_{A\to B} \max(0, V^{(A\to B)}(t)) + C^c_{B\to A} \max(0, -V^{(A\to B)}(t)) \\ + \ & C^e_{1,A}(X_{(A,1)}(t)) + C^e_{2,B}(X_{(B,2)}(t)) \\ + \ & (C^d_{A,inv} \max(0, I_A(t)) + (C^d_{A,back} \max(0, -I_A(t)) \\ + \ & (C^d_{B,inv} \max(0, I_B(t)) + (C^d_{B,back} \max(0, -I_B(t)) \ \} \ ]. \end{aligned} \tag{6.7}$$

We have the following state equations:

$$X_{(A,1)}(t+1) = X_{(A,1)}(t) - V^{(A\to B)}(t), \tag{6.8}$$

$$X_{(B,2)}(t+1) = X_{(B,2)}(t) + V^{(A\to B)}(t), \tag{6.9}$$

$$I_A(t+1) = I_A(t) + X_{(A,1)}(t)P_{(A,1)} - d_A(t), \tag{6.10}$$

$$I_B(t+1) = I_B(t) + X_{(B,2)}(t)P_{(B,2)} - d_B(t). \tag{6.11}$$

The constraints are as follows:

$$X_{(A,1)}(t) \geq 0, \tag{6.12}$$

$$X_{(B,2)}(t) \geq 0, \tag{6.13}$$

$$V^{(A\to B)}(t) \geq -X_{(B,2)}(t), \tag{6.14}$$

$$V^{(A\to B)}(t) \leq X_{(A,1)}(t). \tag{6.15}$$

6.2.2 Optimality Equation

In order to derive the Optimality Equation for this Markov decision process, we break J^0 into two parts:

$$J^0 = J + J^c,$$

where

$$J = E\left[\sum_{t=1}^{T-1} g_t(\overline{X}(t), \overline{U}(t)) + g_T(\overline{U}(T))\right],$$

with

$$
\begin{aligned}
&g_t(\overline{X}(t), \overline{U}(t)) = C^c_{A\to B}\max(0, V^{(A\to B)}(t)) + C^c_{B\to A}\max(0, -V^{(A\to B)}(t)) \\
+ \quad & C^e_{1,A}(X_{(A,1)}(t) - V^{(A\to B)}(t)) + C^e_{2,B}(X_{(B,2)}(t) + V^{(A\to B)}(t)) \\
+ \quad & C^d_{A,inv}\max(0, \quad I_A(t) + 2X_{(A,1)}(t)P_{(A,1)} - V^{(A\to B)}(t)P_{(A,1)} - d_A(t) - d_A(t+1)) \\
+ \quad & C^d_{A,back}\max(0, -I_A(t) - 2X_{(A,1)}(t)P_{(A,1)} + V^{(A\to B)}(t)P_{(A,1)} + d_A(t) + d_A(t+1)) \\
+ \quad & C^d_{B,inv}\max(0, \quad I_B(t)) + 2X_{(B,2)}(t)P_{(B,2)} + V^{(A\to B)}(t)P_{(B,1)} - d_B(t) - d_B(t+1)) \\
+ \quad & C^d_{B,back}\max(0, -I_B(t)) - 2X_{(B,2)}(t)P_{(B,2)} - V^{(A\to B)}(t)P_{(B,1)} + d_B(t) + d_B(t+1)), \\
& t = 1, \ldots, T-2,
\end{aligned}
$$

$$
\begin{aligned}
&g_{T-1}(\overline{X}(T-1), \overline{U}(T-1)) = \\
&C^c_{A\to B}\max(0, V^{(A\to B)}(T-1)) + C^c_{B\to A}\max(0, -V^{(A\to B)}(T-1)) \\
&+C^e_{1,A}(X_{(A,1)}(T-1) - V^{(A\to B)}(T-1)) + C^e_{2,B}(X_{(B,2)}(T-1) + V^{(A\to B)}(T-1)),
\end{aligned}
$$

$$
g_T(\overline{U}(T)) = C^c_{A\to B}\max(0, V^{(A\to B)}(T)) + C^c_{B\to A}\max(0, -V^{(A\to B)}(T)),
$$

and

$$
\begin{aligned}
J^c \quad = \quad & C^e_{1,A}\ X_{(A,1)}(1) + C^e_{2,B}\ X_{(B,2)}(1) \\
+ \quad & C^d_{A,inv}\ \max(0, I_A(1)) + C^d_{A,back}\ \max(0, -I_A(1)) \\
+ \quad & C^d_{B,inv}\ \max(0, I_B(1)) + (C^d_{B,back}\ \max(0, -I_B(1)) \\
+ \quad & E_{d_A(1)}\ [\ C^d_{A,inv}\ \max(0, \quad I_A(1) + X_{(A,1)}(1)P_{(A,1)} - d_A(1)) + \\
& \qquad C^d_{A,back}\ \max(0, -I_A(1) - X_{(A,1)}(1)P_{(A,1)} + d_A(1))\ ] \\
+ \quad & E_{d_B(1)}\ [\ C^d_{B,inv}\ \max(0, \quad I_B(1) + X_{(B,2)}(1)P_{(B,2)} - d_B(1)) + \\
& \qquad C^d_{B,back}\ \max(0, -I_B(1) - X_{(B,2)}(1)P_{(B,2)} + d_B(1))\ ].
\end{aligned}
$$

Note that $g_{T-1}(\overline{X}(T-1), \overline{U}(T-1))$ contains no inventory and backlog costs and $g_T(\overline{U}(T))$ only consists of switching costs. Because J^c is independent of the control policy, the problem of minimizing J^0 is equivalent to that of minimizing J.

By applying the backwards induction method of dynamic programming, we get the Optimality Equation

$$\begin{aligned} J_T^* &= \min_{\overline{U}(T)} \{g_T(\overline{U}(T))\}, \\ J_t^*(\overline{X}(t)) &= \min_{\overline{U}(t)} \{Q_t(\overline{X}(t), \overline{U}(t))\}, \\ & t = 1, \ldots, T-1, \end{aligned}$$

where

$$\begin{aligned} & Q_{T-1}(\overline{X}(T-1), \overline{U}(T-1)) = g_{T-1}(\overline{X}(T-1), \overline{U}(T-1)) + J_T^*, \\ & Q_t(\overline{X}(t), \overline{U}(t)) = E_{d_A(t), d_A(t+1), d_B(t), d_B(t+1)} \left[g_t(\overline{X}(t), \overline{U}(t)) \mid \overline{X}(t), \overline{U}(t)) \right] \\ & \quad + J_{t+1}^*(\{X_{(A,1)}(t) - V^{(A\to B)}(t), X_{(B,2)}(t) + V^{(A\to B)}(t), \\ & \qquad I_A(t) + X_{(A,1)}(t) P_{(A,1)} - d_A(t), I_B(t) + X_{(B,2)}(t) P_{(B,2)} - d_B(t)\}), \\ & \quad t = 1, \ldots, T-2. \end{aligned}$$

$J_t^*(\overline{X}(t))$ is the optimal cost-to-go up to time t, and $Q_t(\overline{X}(t), \overline{U}(t))$ is known as the Q-factor at time t corresponding to $(\overline{X}(t), \overline{U}(t))$.

At time T, $g_T(\cdot)$ is convex with respect to $\overline{U}(T)$, and obviously, the optimal policy at time T is $\mu_T^* = 0$ and $J_T^* = 0$.

At time $T-1$, $Q_{T-1}(\overline{X}(T-1), \overline{U}(T-1)) = g_{T-1}(\overline{X}(T-1), \overline{U}(T-1))$ since $J_T^* = 0$. We can show that $Q_{T-1}(\overline{X}(T-1), \overline{U}(T-1))$ is piecewise linear and convex with respect to $\overline{U}(T-1)$.

Before we present the optimal policy at time $T-1$, we first discuss some properties of piecewise linear convex functions.

6.2.3 Properties of Piecewise Linear Convex Functions

A piecewise linear function $f : I = [a, b] \to R$ with N linear pieces can be defined in the following way:

$$
\begin{aligned}
&f(x) = u_i + J_i(x - n_i), \qquad && x \in [n_i, n_{i+1}), i = 0, \ldots, N-1 \\
&f(b) = u_{N-1} + J_{N-1}(b - n_{N-1}), &&
\end{aligned} \tag{6.16}
$$

where $a = n_0 < n_1 < \ldots < n_N = b$ are called *partition points*, following the definition of partition points of a piecewise constant function in [68], J_i, $i = 0, \ldots, N-1$, are called *slopes*, and $J_i = J_{i+1}$ only when $u_{i+1} \neq u_i + J_i(n_{i+1} - n_i)$. Then, the function $f(x)$ is determined by the triplets $\{u_i, J_i, n_i\}$. The form (6.16) is referred to as the standard form of $f(x)$.

If $f(x)$ is also convex, then piecewise linearity ensures that $f(x)$ is continuous on $[a, b]$, and the left derivative $f'_-(x)$ and right derivative $f'_+(x)$, defined by $\lim_{\delta > 0} \frac{f(x) - f(x-\delta)}{\delta}$ and $\lim_{\delta > 0} \frac{f(x+\delta) - f(x)}{\delta}$ respectively, exist and are increasing [69]. Note that $f'_-(n_i) = J_{i-1}$ for $i = 1, \ldots, N$ and $f'_+(n_i) = J_i$ for $i = 0, \ldots, N-1$, so J_i is increasing with i. In addition, $J_i \neq J_{i+1}$, as $u_{i+1} = u_i + J_i(n_{i+1} - n_i)$ from continuity of $f(x)$. so, J_i is strictly increasing with i.

Because $f(x)$ is continuous on $[a, b]$, by the extreme value theorem, $f(x)$ attains a minimum value. Given a point $x_0 \in [a, b]$, suppose that there is some $\delta_1 > 0$ such that $f(x) \geq f(x_0)$ for all $x \in [a, b]$ such that $|x - x_0| < \delta_1$, then we said at point x_0 a local minimum value of $f(x)$ is attained. Due to convexity of $f(x)$, at point x_0, a global minimum value of $f(x)$ is also attained. We call such a point x_0 a *minimizer* for $f(x)$, and denote the smallest of all possible minimizers on $[a, b]$ as x^*. In words, the minimum is achieved at a partition point.

It is straightforward to show that

$$
x^* = \begin{cases}
a & \text{if} \quad J_0 > 0, \\
n_i & \text{if} \quad J_i = 0, \\
n_{i+1} & \text{if} \quad J_i < 0 < J_{i+1}, \\
b & \text{if} \quad J_{N-1} < 0.
\end{cases} \tag{6.17}
$$

Note that when $J_i = 0 \ \forall i$, n_i is the smallest of $[n_i, n_{i+1})$ in which all points are minimizers for $f(x)$.

Lemma 6.2.1 *Given the standard form of piecewise linear convex function $f(x)$ in (6.16), there exists i^* such that $n_{i^*} = x^*$. For a particular $\{J_i\}$ sequence, i^* is unique.*

This lemma directly follows from (6.17).

If the $\{J_i\}$ sequence is known, i^* can be obtained directly by (6.17).

In many cases, however, the $\{J_i\}$ sequence is not available. But we can still obtain a minimizer for $f(x)$ by minimizing $f(x)$ over the set of partition points $S = \{n_0, \ldots, n_N\}$, according to (6.17). Usually, the set S can be constructed using some known sets. For example, $f(x)$ is defined as a linear combination $\lambda_1 f_1(x) + \ldots + \lambda_d f_d(x)$ of piecewise linear convex functions $f_1(x), \ldots, f_d(x)$ with non-negative coefficients $\lambda_1, \ldots, \lambda_d$, where $f_1(x), \ldots, f_d(x)$ are called *basis functions of* $f(x)$. We can show that $f(x)$ is still piecewise linear convex, and S_f is a subset of the union of S_{fi}, where S_f is the set of partition points of $f(x)$, and S_{fi} is the set of partition points of $f_i(x)$. In this case, a minimizer can be obtained by minimizing $f(x)$ over S_f, without the knowledge of the $\{J_i\}$ sequence.

Let us consider a non-standard form of the piecewise linear convex function $f(x)$. Suppose $\{q_0, \ldots, q_K\}$ is a set of K points on $[a, b]$ such that it contains all partition points and $a = q_0 < \ldots < q_K = b$. Then the non-standard form of $f(x)$ is as follows:

$$\begin{aligned} &f(x) = v_k + L_k(x - q_k), \qquad && x \in [q_k, q_{k+1}), k = 0, \ldots, K-1, \\ &f(b) = v_{K-1} + L_{K-1}(b - q_{K-1}), && \end{aligned} \tag{6.18}$$

where q_k, $k = 0, \ldots, K$, are called *pseudo partition points*, and L_k, $k = 0, \ldots, K-1$, are called *pseudo slopes*. It is not necessary that $L_k \neq L_{k+1}$ when $v_{k+1} = v_k + L_k(q_{k+1} - q_k)$.

Similar to (6.17), we have

$$x^* = \begin{cases} a & \text{if} \quad L_0 > 0, \\ q_k & \text{if} \quad L_k = 0 \text{ and } L_{k-1} < 0 \text{ (if exists)}, \\ q_{k+1} & \text{if} \quad L_k < 0 < L_{k+1}, \\ b & \text{if} \quad L_{K-1} < 0. \end{cases} \tag{6.19}$$

Lemma 6.2.2 *Given the non-standard form of piecewise linear convex function $f(x)$ in (6.18), there exists k^* such that $q_{k^*} = x^*$. For a particular $\{L_k\}$ sequence, k^* is unique.*

This lemma directly follows from (6.19).

Now we consider a function $g(\bar{y}, x)$ with $\bar{y} = (y_1, \ldots, y_K) \in D_{\bar{y}}$, $a(\bar{y}) \leq x \leq b(\bar{y})$, where $D_{\bar{y}}$ is the domain of $\bar{y}$, and the function has the following properties:
(6.A.1) for all $\bar{y}_a \in D_{\bar{y}}$, $g(\bar{y}_a, x)$ is a piecewise linear convex function with respect to x, and can be written in the standard form:

$$\begin{aligned} &g(\bar{y}_a, x) = u_i(\bar{y}_a) + J_i(\bar{y}_a)(x - n_i(\bar{y}_a)), \quad x \in [n_i(\bar{y}_a), n_{i+1}(\bar{y}_a)), i = 0, \ldots, N(\bar{y}_a) - 1, \\ &g(\bar{y}_a, b(\bar{y}_a)) = u_{N(\bar{y}_a)-1}(\bar{y}_a) + J_{N(\bar{y}_a)-1}(\bar{y}_a)(b(\bar{y}_a) - n_{N(\bar{y}_a)-1}(\bar{y}_a)), \end{aligned} \tag{6.20}$$

where $a(\bar{y}_a) = n_0(\bar{y}_a) < n_1(\bar{y}_a) < \ldots < n_N(\bar{y}_a) = b(\bar{y}_a)$;
(6.A.2) there exists a set of functions $\{q_0(\bar{y}), \ldots, q_M(\bar{y})\}$ such that for all $\bar{y}_a \in D_{\bar{y}}$, the set of partition points of $g(\bar{y}_a, x)$, $\{n_0(\bar{y}_a), \ldots, n_{N(\bar{y}_a)}\bar{y}_a\}$ is a subset of $\{q_0(\bar{y}_a), \ldots, q_M(\bar{y}_a)\}$. Functions $q_k(\bar{y})$, $k = 0, \ldots, M$, are referred to as *pseudo partition functions*.

We want to discover some properties of a minimizer $\text{argmin}_x g(\bar{y}, x)$. The smallest of all possible minimizers is denoted as $\chi^*(\bar{y})$. Note that for a given $\bar{y}_a \in D_{\bar{y}}$, there exists $i^* \in 0, \ldots, N_a$ such that $\chi^*(\bar{y}_a) = n_{a,i^*}$, from the piecewise linear convex property.

Note that for all $\bar{y}_a \in D_{\bar{y}}$, $a(\bar{y}_a) \leq q_k(\bar{y}_a) \leq b(\bar{y}_a)$ $\forall\, k = 0, \ldots, M$, so, without loss of generality, we set $q_0(\bar{y}) = a(\bar{y})$ and $q_M(\bar{y}) = b(\bar{y})$. For the rest of pseudo partition functions, $q_k(\bar{y}_a)$, $k = 1, \ldots, M-1$, their order depends on the value of $\bar{y}$, so there are $(M-1)!$ permutations of $\{q_0(\bar{y}), \ldots, q_M(\bar{y})\}$. For $\bar{y}_a, \bar{y}_b \in D_{\bar{y}}$, if the order of $q_k(\bar{y}_a)$, is the same as the order of $q_k(\bar{y}_b)$ $k = 1, \ldots, M-1$, we say $\bar{y}_a$ and $\bar{y}_b$ belong to the same partition of $D_{\bar{y}}$.

For the mth permutation of $\{q_0(\bar{y}), \ldots, q_M(\bar{y})\}$, $m \in \{1, \ldots, (M-1)!\}$, we reorder them into $Q^m = \{q_0^m(\bar{y}), \ldots, q_M^m(\bar{y})\}$ with $q_k^m(\bar{y})$ increasing with i. The set of all $\bar{y}_b$ satifying $q_0^m(\bar{y}_b) \leq \ldots \leq q_M^m(\bar{y}_b)$, denoted as D^m, is one of $(M-1)!$ partitions of $D_{\bar{y}}$.

We are going to show that, for a given partition, the form of minimizers is the same for all $\bar{y}$ in this partition.

By (6.A.1) and (6.A.2), for all $\bar{y}_b \in D^m$, we have non-standard forms of $g(\bar{y}_b, x)$ as follows:

$$\begin{aligned} g(\bar{y}_b, x) &= v_k^m(\bar{y}_b) + L_k^m(\bar{y}_b)(x - q_k^m(\bar{y}_b)), \quad x \in [q_k^m(\bar{y}_b), q_{k+1}^m(\bar{y}_b)), k = 0, \ldots, M-1, \\ g(\bar{y}_b, b(\bar{y}_b)) &= v_{M-1}^m(\bar{y}_b) + L_{M-1}^m(\bar{y}_b)(b(\bar{y}_b) - q_{M-1}^m(\bar{y}_b)), \end{aligned} \tag{6.21}$$

and we assume (6.A.3) for the mth permutation Q^m, $L_k^m(\bar{y}_b)$ is the same, say L_k^m, for all $\bar{y}_b \in D^m$, $m = 1, \ldots, (M-1)!$. L_k^m is called the kth *pseudo slope* for permutation Q^m.

Lemma 6.2.3 *Consider the function $g(\bar{y}, x)$ and assume (6.A.1) - (6.A.3), for a permutation Q^m and the corresponding partition D^m, then there exists a unique $k^* \in \{0, \ldots, M\}$, $\chi^*(\bar{y}_b) = q_{k^*}^m(\bar{y}_b)$ for all $\bar{y}_b \in D^m$.*

Because the $\{L_k^m(\bar{y}_b)\}$ sequence is the same for all $\bar{y}_b \in D^m$, by Lemma 6.2.2 the conclusion follows.

6.2.4 Optimal policy with deterministic demands

It is straightforward to show that $Q_{T-1}(\overline{X}(T-1), \overline{U}(T-1))$ is a piecewise linear and convex function with pseudo partition functions

$$\{q_0(\overline{X}(T-1)), q_1(\overline{X}(T-1)), q_2(\overline{X}(T-1))\} = \{-X_{(B,2)}(T-1), 0, X_{(A,1)}(T-1)\}$$

and pseudo slopes

$$\{L_{T-1,1}, L_{T-1,2}\} = \{-(C^e_{1,A} - C^e_{2,B} + C^c_{B\to A}), -(C^e_{1,A} - C^e_{2,B} - C^c_{A\to B})\},$$

and (A.1) - (A.3) are satisfied.

By applying Lemma 6.2.3 directly, an optimal policy at time $T-1$ is:

$$\mu^*_{T-1}(\overline{X}(T-1)) = \begin{cases} X_{(A,1)}(T-1) & L_{T-1,2} < 0, \\ 0 & L_{T-1,1} < 0 \leq L_{T-1,2}, \\ -X_{(B,2)}(T-1) & L_{T-1,1} \geq 0. \end{cases} \tag{6.22}$$

This optimal policy result at time $T-1$ can be paraphrased as:

- if the unit operating cost for product A minus that for product B is greater than the unit switching cost from A to B, the optimal policy is to switch as much capacity from A to B as possible;
- if the unit operating cost for product B minus that for product A is greater than the unit switching cost from B to A, the optimal policy is to switch as much capacity from A to B as possible;
- if in between, do not switch any capacity at all.

In addition,

$$\begin{aligned} J^*_{T-1}(\overline{X}(T-1)) &= Q_{T-1}(\overline{X}(T-1), \mu^*_{T-1}(\overline{X}(T-1))) \\ &= C^{mix}_{A,T-1}X_{(A,1)}(T-1) + C^{mix}_{B,T-1}X_{(B,2)}(T-1) \end{aligned}$$

with

$$\{C^{mix}_{A,T-1}, C^{mix}_{B,T-1}\} = \begin{cases} \{C^e_{2,B} + C^c_{A\to B}, C^e_{2,B}\}, & L_{T-1,2} \le 0, \\ \{C^e_{1,A}, C^e_{2,B}\}, & L_{T-1,1} < 0 < L_{T-1,2}, \\ \{C^e_{1,A}, C^e_{1,A} + C^c_{B\to A}\}, & L_{T-1,1} \ge 0. \end{cases}$$

At time $T-2$, if demands are deterministic, the Q-factor (subscript 'd' refers to deterministic) is given by

$$\begin{aligned} &Q_{d,T-2}(\overline{X}(T-2), \overline{U}(T-2)) \\ &= \; g_{T-2}(\overline{X}(T-2), \overline{U}(T-2)) \\ &+ \; J^*_{T-1}(\{X_{(A,1)}(T-2) - V^{(A\to B)}(T-2), X_{(B,2)}(T-2) + V^{(A\to B)}(T-2), \\ &\qquad I_A(T-2) + X_{(A,1)}(T-2)P_{(A,1)} - d_A(T-2), \\ &\qquad I_B(T-2) + X_{(B,2)}(T-2)P_{(B,2)} - d_B(T-2)\}), \end{aligned}$$

and, since $Q_{d,T-2}(\overline{X}_a(T-2), \overline{U}(T-2))$ is a linear combination with non-negative coefficients of several piecewise linear convex functions, we can show the following remark is true.

Remark 6.2.1 $Q_{d,T-2}(\overline{X}(T-2), \overline{U}(T-2))$ *is a piecewise linear convex function with respect to its second argument, with pseudo partition functions* $\{-X_B(T-2), 0, S_{A,T-2}(\overline{X}(T-2)), S_{B,T-2}(\overline{X}(T-2)), X_A(T-2)\}$ *assuming* $-X_B(T-2) \le S_{A,T-2}(\overline{X}(T-2)) \le X_A(T-2)$ *and* $-X_B(T-2) \le S_{B,T-2}(\overline{X}(T-2)) \le X_A(T-2)$*, where*

$$S_{A,T-2}(\overline{X}(T-2)) = 2X_{(A,1)}(T-2) + (I_A(T-2) - d_A(T-2) - d_A(T-1))/P_{(A,1)},$$

$$S_{B,T-2}(\overline{X}(T-2)) = -2X_{(B,2)}(T-2) - (I_B(T-2) - d_B(T-2) - d_B(T-1))/P_{(B,2)},$$

and with pseudo slopes $\{L_{T-2,1}, L_{T-2,2}, L_{T-2,3}, L_{T-2,4}, L_{T-2,5}, L_{T-2,6}, L_{T-2,7}, L_{T-2,8}\}$, *where*

$$\begin{aligned}
L_{T-2,1} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} - C^{c}_{B\to A} \\
&\quad -P_{(A,1)}C^{d}_{A,inv} - P_{(B,2)}C^{d}_{B,back}, \\
L_{T-2,2} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} - C^{c}_{B\to A} \\
&\quad -P_{(A,1)}C^{d}_{A,inv} + P_{(B,2)}C^{d}_{B,inv}, \\
L_{T-2,3} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} - C^{c}_{B\to A} \\
&\quad +P_{(A,1)}C^{d}_{A,back} - P_{(B,2)}C^{d}_{B,back}, \\
L_{T-2,4} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} - C^{c}_{B\to A} \\
&\quad +P_{(A,1)}C^{d}_{A,back} + P_{(B,2)}C^{d}_{B,inv}, \\
L_{T-2,5} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} + C^{c}_{A\to B} \\
&\quad -P_{(A,1)}C^{d}_{A,inv} - P_{(B,2)}C^{d}_{B,back}, \\
L_{T-2,6} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} + C^{c}_{A\to B} \\
&\quad -P_{(A,1)}C^{d}_{A,inv} + P_{(B,2)}C^{d}_{B,inv}, \\
L_{T-2,7} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} + C^{c}_{A\to B} \\
&\quad +P_{(A,1)}C^{d}_{A,back} - P_{(B,2)}C^{d}_{B,back}, \\
L_{T-2,8} &= -C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^{e}_{1,A} + C^{e}_{2,B} + C^{c}_{A\to B} \\
&\quad +P_{(A,1)}C^{d}_{A,back} + P_{(B,2)}C^{d}_{B,inv}.
\end{aligned}$$

Among the eight pseudo slopes, $L_{T-2,1}$ is always the smallest, $L_{T-2,8}$ is always the largest, and the others lie somewhere in between these two values, but their order depends on particular problems. We obtain an optimal policy by partitioning into cases. For example, in the case where

$$P_{(A,1)}C^{d}_{A,back} + P_{(A,1)}C^{d}_{A,inv} \geq P_{(B,2)}C^{d}_{B,back} + P_{(B,2)}C^{d}_{B,inv},$$

and

$$C^{c}_{A\to B} + C^{c}_{B\to A} \geq P_{(A,1)}C^{d}_{A,back} + P_{(A,1)}C^{d}_{A,inv} + P_{(B,2)}C^{d}_{B,back} + P_{(B,2)}C^{d}_{B,inv},$$

the eight pseudo slopes can be sorted in increasing order as

$$\{L_{T-2,1}, L_{T-2,2}, L_{T-2,3}, L_{T-2,4}, L_{T-2,5}, L_{T-2,6}, L_{T-2,7}, L_{T-2,8}\},$$

and using the procedure given later, we obtain the following optimal policy:

$$\mu^*_{T-2}(\overline{X}(T-2)) = \begin{cases} X_{(A,1)}(T-2) & L_{T-2,8} < 0 \\ max(0, S_{A,T-2}(\cdot)), S_{B,T-2}(\cdot)) & L_{T-2,7} < 0 \leq L_{T-2,8} \\ max(0, S_{A,T-2}(\cdot)) & L_{T-2,6} < 0 \leq L_{T-2,7} \\ I_{\{S_{A,T-2}(\cdot) \geq 0\}} & \\ \quad med(0, S_{A,T-2}(\cdot), S_{B,T-2}(\cdot)) & L_{T-2,5} < 0 \leq L_{T-2,6} \\ 0 & L_{T-2,4} < 0 \leq L_{T-2,5} \\ I_{\{S_{A,T-2}(\cdot) \leq 0\}} & \\ \quad med(0, S_{A,T-2}(\cdot), S_{B,T-2}(\cdot)) & L_{T-2,3} < 0 \leq L_{T-2,4} \\ min(0, S_{A,T-2}(\cdot)) & L_{T-2,2} < 0 \leq L_{T-2,3} \\ min(0, S_{A,T-2}(\cdot), S_{B,T-2}(\cdot)) & L_{T-2,1} < 0 \leq L_{T-2,2} \\ -X_{(B,2)}(T-2) & L_{T-2,1} \geq 0, \end{cases} \tag{6.23}$$

where $med(\cdot,\cdot,\cdot)$ represents the median function.

Consider the optimal policy when $L_{T-2,8} < 0$. We assume that $L_{T-1,1} < 0 \leq L_{T-1,2}$; then $-L_{T-2,8} = 2C^e_{1,A} - 2C^e_{2,B} - C^c_{A\to B} - P_{(A,1)}C^d_{A,back} - P_{(B,2)}C^d_{B,inv}$. The optimal policy, $\mu^*_{T-2}(\overline{X}(T-2)) = X_{(A,1)}(T-2)$ if $L_{T-2,8} < 0$, can be explained as: if twice the unit operating cost for A minus that for B is greater than the sum of the unit switching cost from A to B, the unit inventory cost for B and the backlog cost for A, then switch as much as possible from A to B, since the benefit from switching as much as possible due to the operating cost difference would cover the sum of the increased inventory cost for B, the increased backlog cost for A, and the increased switching cost. This situation, along with the situation where $L_{T-2,1} \geq 0$, is the situation when the unit operating cost difference dominates.

Next, if twice the unit operating cost for A minus that for B is less than sum of the unit switching cost from A to B, the unit inventory cost for B, and the backlog cost for A, i.e., $-L_{T-2,8} \leq 0$, the benefit from switching as much as possible would not compensate for the total loss from the increased inventory cost for B, the increased backlog cost for A, and the increased switching cost. So it is better to switch a little less from A to B. How much is the best depends on the unit cost balance. As unit operating cost difference decreases, the amount of switching from A to B goes down, until at some point, switching cost dominates operating cost, inventory and backlog cost, i.e., $L_{T-2,4} < 0 \leq L_{T-2,5}$, the optimal policy is to avoid switching at all.

There are a total of 12 cases, for which we can uniquely sort out pseudo slopes and determine $\mu^*_{T-2}(\overline{X}(T-2))$.

Now we present the procedure for developing the structure of an optimal policy for the one-machine, two-product example with deterministic demands as follows:

Step 0 Initially, we have $J^*_T = 0$, and set $t = T - 1$.

Step 1 Unfold the expression of $Q_t(\overline{X}(t), \overline{U}(t))$, where

$$\begin{aligned} Q_t(\overline{X}(t), \overline{U}(t)) &= g_t(\overline{X}(t), \overline{U}(t)) \\ &+ J^*_{t+1}(\{X_{(A,1)}(t) - V^{(A\to B)}(t), X_{(B,2)}(t) + V^{(A\to B)}(t), \\ &\quad I_A(t) + X_{(A,1)}(t)P_{(A,1)} - d_A(t), I_B(t) + X_{(B,2)}(t)P_{(B,2)} - d_B(t)\}). \end{aligned} \tag{6.24}$$

Step 2 Verify that $Q_t(\overline{X}(t), \overline{U}(t))$ is piecewise linear convex.

Step 3 Construct a set of pseudo partition functions by observing unfolded $Q_t(\overline{X}(t), \overline{U}(t))$.

Step 4 Identify possible partitions and determine the relative order of pseudo partition functions.

Step 5 For the mth partition, minimize $Q_t(\overline{X}_b(t), \overline{U}(t))$ over the pseudo partition points

$$\{q_{0,t}(\overline{X}_b(t)), \ldots, q_{M,t}(\overline{X}_b(t))\}$$

for an arbitraily chosen $\overline{X}_b(t) \in D_{\overline{X}(t)}$. Suppose $Qt(\overline{X}_b(t), q^m_{k^*_m,t}(\overline{X}_b(t))$ give the smallest value, then $q^m_{k^*_m,t}(\cdot)$ is the optimal policy for mth partition. Repeat this step for all partitions.

Step 6 Combine optimal policies for all partitions to obtain $\mu^*_t(\overline{X}(t))$ for the whole state space.

Step 7 Calculate $J^*_t(\overline{X}(t)) = Q_t(\overline{X}(t), \mu^*_t(\overline{X}(t)))$.

Step 8 If $t = 1$, stop; else $t = t - 1$, go to Step 1.

When we use this procedure to derive the structure of an optimal policy for the one-machine, two-product example, step 1 and step 7 can be executed using the MATHEMATICA software. The following three scenarios show optimal policies have a threshold-like structure, given unit costs and production rates. Note that we do not need to know the pseudo slopes when applying the above procedure. The resulting optimal policies using this procedure are consistent with those from linear programming, by using the CPLEX software.

Scenario 1: $C^c_{A\to B} = 428$, $C^c_{B\to A} = 438$,$C^e_{1,A} = 856$, $C^e_{2,B} = 931$, $C^d_{A,inv} = 0.375$, $C^d_{A,back} = 3.75$, $C^d_{B,inv} = 0.488$, $C^d_{B,back} = 3.25$, $P_{(A,1)} = 50$, $P_{(B,2)} = 50$ and $T = 4$.

Using the above procedure, we obtain $\mu^*_4 = 0$, $\mu^*_3 = 0$, $\mu^*_2 = 0$ and

$$\begin{aligned}\mu^*_1 &= I_{\{R_{A,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{A,1}(\cdot)>Q_{B,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}} \\ &\qquad second(R_{A,1}(\cdot), R_{B,1}(\cdot), Q_{A,1}(\cdot), Q_{B,1}(\cdot)) \\ &+ (1 - I_{\{R_{A,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{A,1}(\cdot)>Q_{B,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}}) \\ &\qquad third(R_{A,1}(\cdot), R_{B,1}(\cdot), Q_{A,1}(\cdot), Q_{B,1}(\cdot)),\end{aligned}$$

where $second(\cdots, \ldots, \cdots)$ is the second smallest of all its arguments and $third(\cdots, \ldots, \cdots)$ is defined similarly,

$$R_{A,1}(\overline{X}(1)) = 2X_{(A,1)}(1) + (I_A(1) - d_A(1) - d_A(2))/P_{(A,1)},$$

$$R_{B,1}(\overline{X}(1)) = -2X_{(B,2)}(1) - (I_B(1) - d_B(1) - d_B(2))/P_{(B,2)},,$$
$$Q_{A,1}(\overline{X}(1)) = (3P_{(A,1)}X_{(A,1)}(1) + I_A(1) - d_A(1) - d_A(2) - d_A(3))/(2P_{(A,1)}),$$
$$Q_{B,1}(\overline{X}(1)) = (-3P_{(B,2)}X_{(B,2)}(1) - I_B(1) + d_B(1) + d_B(2) + d_B(3))/(2P_{(B,2)}).$$

Scenario 2: $C^c_{A\to B} = 428$, $C^c_{B\to A} = 438$,$C^e_{1,A} = 856$, $C^e_{2,B} = 931$, $C^d_{A,inv} = 1.25$, $C^d_{A,back} = 12.5$, $C^d_{B,inv} = 0.78$, $C^d_{B,back} = 5.2$, $P_{(A,1)} = 50$, $P_{(B,2)} = 50$ and $T = 4$. Using the above procedure, we obtain $\mu^*_4 = 0$, $\mu^*_3 = 0$, $\mu^*_2 = min(0, S_{A,2}(\cdot))$ and

$$\begin{aligned}\mu^*_1 &= I_{\{Q_{A,1}(\cdot)<min(0,R_{A,1}(\cdot))\}}min(0, Q_{A,1}(\cdot)) \\ &+ I_{\{Q_{A,1}(\cdot)\geq min(0,R_{A,1}(\cdot))\}}second(0, R_{A,1}(\cdot), Q_{A,1}(\cdot), Q_{B,1}(\cdot)).\end{aligned}$$

Scenario 3: $C^c_{A\to B} = 428$, $C^c_{B\to A} = 438$,$C^e_{1,A} = 856$, $C^e_{2,B} = 931$, $C^d_{A,inv} = 5$, $C^d_{A,back} = 50$, $C^d_{B,inv} = 5.85$, $C^d_{B,back} = 39$, $P_{(A,1)} = 50$, $P_{(B,2)} = 50$ and $T = 4$.

Using the above procedure, we obtain $\mu^*_4 = 0$, $\mu^*_3 = 0$,
$\mu^*_2 = min(0, S_{A,2}(\cdot)) + I_{\{S_{A,2}(\cdot)>0}I_{S_{B,2}(\cdot)>0\}}min(S_{A,2}(\cdot), S_{B,2}(\cdot))$ and

$$\begin{aligned}\mu^*_1 &= I_{\{R_{A,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{A,1}(\cdot)>Q_{B,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}} \\ &\quad (I_{\{R_{A,1}(\cdot)>0\}}R_{A,1}(\cdot) + I_{\{R_{A,1}(\cdot)<0\}}second(0, R_{A,1}(\cdot), R_{B,1}(\cdot), Q_{A,1}(\cdot), Q_{B,1}(\cdot))) \\ &+ (1 - I_{\{R_{A,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}}I_{\{R_{A,1}(\cdot)>Q_{B,1}(\cdot)\}}I_{\{R_{B,1}(\cdot)>Q_{A,1}(\cdot)\}}) \\ &\quad\quad second(R_{A,1}(\cdot), R_{B,1}(\cdot), Q_{A,1}(\cdot)).\end{aligned}$$

6.2.5 Optimal policy with random demands

Note that the optimal policy at time $T-1$ with random demands is the same as that with deterministic demands.

At time $t = 1, \ldots, T-2$, if demands are random,

$$\begin{aligned}Q_t(\overline{X}(t), \overline{U}(t)) &= E_{d_A(t),d_A(t+1),d_B(t),d_B(t+1)}\left[g_t(\overline{X}(t), \overline{U}(t)) \mid \overline{X}(t), \overline{U}(t)\right] \\ &+ J^*_{t+1}(X_{(A,1)}(t) - V^{(A\to B)}(t), X_{(B,2)}(t) + V^{(A\to B)}(t), \\ &\quad I_A(t) + X_{(A,1)}(t)P_{(A,1)} - d_A(t), I_B(t) + X_{(B,2)}(t)P_{(B,2)} - d_B(t)).\end{aligned}$$

If ξ is a random variable with c.d.f. $F_\xi(\cdot)$, we know that

$$\begin{aligned}
E_\xi[p \cdot \max(0, y - \xi)] &= p \cdot \int_{-\infty}^{y} F_\xi(\omega)\ d(\omega), \\
E_\xi[r \cdot \max(0, \xi - y)] &= r \cdot \int_{-\infty}^{-y} (1 - F_\xi(-\omega))\ d(-\omega).
\end{aligned}$$

Therefore, $Q_t(\overline{X}(t), \overline{U}(t)) =$

$$\begin{aligned}
& C^c_{A\to B} \max(0, V^{(A\to B)}(t)) + C^c_{B\to A} \max(0, -V^{(A\to B)}(t)) \\
+ \quad & C^e_{1,A}(X_{(A,1)}(t) - V^{(A\to B)}(t)) + C^e_{2,B}(X_{(B,2)}(t) + V^{(A\to B)}(t)) \\
+ \quad & C^d_{A,inv} \int_{-\infty}^{I_A(t)+2X_{(A,1)}(t)P_{(A,1)}-V^{(A\to B)}P_{(A,1)}} F_{d_A(t)+d_A(t+1)}(\omega)\ d\omega, \\
+ \quad & C^d_{A,back} \int_{-\infty}^{-I_A(t)-2X_{(A,1)}(t)P_{(A,1)}+V^{(A\to B)}(t)P_{(A,1)}} (1 - F_{d_A(t)+d_A(t+1)}(-\omega))\ d(-\omega), \\
+ \quad & C^d_{B,inv} \int_{-\infty}^{I_B(t))+2X_{(B,2)}(t)P_{(B,2)}+V^{(A\to B)}(t)P_{(B,1)}} F_{d_B(t)+d_B(t+1)}(\omega)\ d\omega \\
+ \quad & C^d_{B,back} \int_{-\infty}^{-I_B(t))-2X_{(B,2)}(t)P_{(B,2)}-V^{(A\to B)}(t)P_{(B,1)}} (1 - F_{d_B(t)+d_B(t+1)}(-\omega))\ d(-\omega) \\
+ \quad & J^*_{t+1}(X_{(A,1)}(t) - V^{(A\to B)}(t), X_{(B,2)}(t) + V^{(A\to B)}(t), \\
& \qquad I_A(t) + X_{(A,1)}(t)P_{(A,1)} - d_A(t), I_B(t) + X_{(B,2)}(t)P_{(B,2)} - d_B(t)), \\
& t = 1, \ldots, T-2.
\end{aligned}$$

At time $T-2$, if demands are random,

$Q_{T-2}(\overline{X}(T-2), \overline{U}(T-2)) \quad =$

$$\begin{aligned}
& C^c_{A\to B} \max(0, V^{(A\to B)}(t)) + C^c_{B\to A} \max(0, -V^{(A\to B)}(t)) \\
+ \quad & C^e_{1,A}(X_{(A,1)}(t) - V^{(A\to B)}(t)) + C^e_{2,B}(X_{(B,2)}(t) + V^{(A\to B)}(t)) \\
+ \quad & C^d_{A,inv} \int_{-\infty}^{I_A(T-2)+2X_{(A,1)}(T-2)P_{(A,1)}-V^{(A\to B)}(T-2)P_{(A,1)}} F_{d_A(T-2)+d_A(T-1)}(\omega)\ d\omega, \\
+ \quad & C^d_{A,back} \int_{-\infty}^{-I_A(T-2)-2X_{(A,1)}(T-2)P_{(A,1)}+V^{(A\to B)}(T-2)P_{(A,1)}} \\
& \qquad (1 - F_{d_A(T-2)+d_A(T-1)}(-\omega))\ d(-\omega), \\
+ \quad & C^d_{B,inv} \int_{-\infty}^{I_B(T-2))+2X_{(B,2)}(T-2)P_{(B,2)}+V^{(A\to B)}(T-2)P_{(B,1)}} F_{d_B(T-2)+d_B(T-1)}(\omega)\ d\omega
\end{aligned}$$

$$
\begin{aligned}
+ \quad & C^d_{B,back} \int_{-\infty}^{-I_B(T-2))-2X_{(B,2)}(T-2)P_{(B,2)}-V^{(A\to B)}(T-2)P_{(B,1)}} \\
& \qquad (1 - F_{d_B(T-2)+d_B(t+1)}(-\omega))\ d(-\omega) \\
+ \quad & C^{mix}_{A,T-1}(X_{(A,1)}(T-2) - V^{(A\to B)}(T-2)) \\
+ \quad & C^{mix}_{B,T-1}(X_{(B,2)}(T-2) + V^{(A\to B)}(T-2)).
\end{aligned}
$$

Remark 6.2.2 *$Q_{T-2}(\overline{X}(T-2), \overline{U}(T-2))$ has the following properties:*

1. *$Q_{T-2}(\overline{X}(T-2), \overline{U}(T-2))$ is a convex function with respect to its second argument;*

2. *$h_{T-2}(\overline{X}(T-2), \overline{U}(T-2)) \equiv \partial Q_{T-2}(\overline{X}(T-2), \overline{U}(T-2))$ =*

$$
\begin{cases} h^r_{T-2}(\overline{X}(T-2), \overline{U}(T-2)), V^{(A\to B)}(T-2) \geq 0, \\ h^l_{T-2}(\overline{X}(T-2), \overline{U}(T-2)), V^{(A\to B)}(T-2) < 0, \end{cases}
$$

where $h^r_{T-2}(\cdot) :=$

$$
\begin{aligned}
& C^c_{A\to B} - C^e_{1,A} + C^e_{2,B} + C^d_{A,back} - C^d_{B,back} - C^{mix}_{A,T-1} + C^{mix}_{B,T-1} \\
& -(C^d_{A,inv} + C^d_{A,back}) \\
& \quad \cdot F_{d_A(T-1)+d_A(T-2)}(I_A(T-2) + 2X_{(A,1)}(T-2)P_{(A,1)} - V^{(A\to B)}(T-2)P_{(A,1)}) \\
& +(C^d_{B,inv} + C^d_{B,back}) \\
& \quad \cdot F_{d_B(T-1)+d_B(T-2)}(I_B(T-2) + 2X_{(B,2)}(T-2)P_{(B,2)} + V^{(A\to B)}(T-2)P_{(B,2)}),
\end{aligned}
$$

and $h^l_{T-2}(\cdot) :=$

$$
\begin{aligned}
& -C^c_{B\to A} - C^e_{1,A} + C^e_{2,B} + C^d_{A,back} - C^d_{B,back} - C^{mix}_{A,T-1} + C^{mix}_{B,T-1} \\
& -(C^d_{A,inv} + C^d_{A,back}) \\
& \quad \cdot F_{d_A(T-1)+d_A(T-2)}(I_A(T-2) + 2X_{(A,1)}(T-2)P_{(A,1)} - V^{(A\to B)}(T-2)P_{(A,1)}) \\
& +(C^d_{B,inv} + C^d_{B,back}) \\
& \quad \cdot F_{d_B(T-1)+d_B(T-2)}(I_B(T-2) + 2X_{(B,2)}(T-2)P_{(B,2)} + V^{(A\to B)}(T-2)P_{(B,2)}).
\end{aligned}
$$

The convexity property leads to the following result:

- if $h_{T-2}(\cdot) < 0$ for all $V^{(A\to B)}(T-2)$, $\mu^*_{T-2}(\overline{X}(T-2)) = X_{(A,1)}(T-2)$;

- if $h_{T-2}(\cdot) > 0$ for all $V^{(A\to B)}(T-2)$, $\mu^*_{T-2}(\overline{X}(T-2)) = -X_{(B,2)}(T-2)$;
- if $h^l_{T-2}(\cdot) < 0$ for all $V^{(A\to B)}(T-2) < 0$ and $h^r_{T-2}(\cdot) > 0$ for all $V^{(A\to B)}(T-2) > 0$, $\mu^*_{T-2}(\overline{X}(T-2)) = 0$,

which is the counterpart of the following in the deterministic demand case:

- situation 1: if $L_{T-2,8} < 0$, $\mu^*_{T-2}(\overline{X}(T-2)) = X_{(A,1)}(T-2)$;
- situation 2: if $L_{T-2,1} > 0$, $\mu^*_{T-2}(\overline{X}(T-2)) = -X_{(B,2)}(T-2)$;
- situation 3: if $L_{T-2,4} < 0 < L_{T-2,5}$, $\mu^*_{T-2}(\overline{X}(T-2)) = 0$.

If the demands for product A and B can only take a finite number of values, then both the c.d.f. of the demand $F_{\{\cdot\}}(\cdot)$ and $Q_{T-2}(\cdot)$ are piecewise linear convex functions, which means we can use the previous results for piecewise linear convex functions. For example, if it is assumed that $d_A(T-1) + d_A(T-2)$ can only take values $w_{A,1}, \ldots, w_{A,r_A}$ with probabilities $p_{A,1}, \ldots, p_{A,r_A}$ and that $d_B(T-1) + d_B(T-2)$ can only take values $w_{B,1}, \ldots, w_{B,r_B}$ with probabilities $p_{B,1}, \ldots, p_{A,r_A}$, the pseudo partition functions for $Q_{T-2}(\cdot)$ would be $\{0, S^i_{A,T-2}(\overline{X}(T-2)), S^j_{B,T-2}(\overline{X}(T-2))\}$, where

$$S^i_{A,T-2}(\overline{X}(T-2)) = 2X_{(A,1)}(T-2) + (I_A(T-2) - w_{A,i})/P_{(A,1)}, \quad i = 1, \ldots, r_A$$

and

$$S^j_{B,T-2}(\overline{X}(T-2)) = (-2X_{(B,2)}(T-2) - (I_B(T-2) - w_{B,j})/P_{(B,2)}, \quad j = 1, \ldots, r_B,$$

and the pseudo slopes depend on $p_{A,i}$, $p_{B,i}$ and unit cost parameters. By the result for piecewise linear convex functions, we can obtain the structure of an optimal policy.

In some cases, if $F_{d_A(T-1)+d_A(T-2)}(\cdot)$ and $F_{d_B(T-1)+d_B(T-2)}(\cdot)$ are "far away" from each other for all states, we can obtain similar results as in the deterministic cases. Define

$$\widehat{S}_{A,T-2}(\overline{X}(T-2)) = 2X_{(A,1)}(T-2) + I_A(T-2)/P_{(A,1)} - F^{-1}_{d_A(T-1)+d_A(T-2)}(0)/P_{(A,1)},$$

$\widehat{S}_{B,T-2}(\overline{X}(T-2)) = -2X_{(B,2)}(T-2) - I_B(T-2)/P_{(B,2)} + F^{-1}_{d_B(T-1)+d_B(T-2)}(0)/P_{(B,2)}$.

An optimal policy $\mu^*_{T-2}(\overline{X}(T-2))$ would be a function of $\widehat{S}_{A,T-2}(\overline{X}(T-2))$, $\widehat{S}_{B,T-2}(\overline{X}(T-2))$ for different state partitions as in Eq. (6.23). In the case that both $F_{d_A(T-1)+d_A(T-2)}(\cdot)$ and $F_{d_B(T-1)+d_B(T-2)}(\cdot)$ are normal distributions with means μ_A, μ_B and variances σ^2_A,σ^2_B, respectively, $F_{d_A(T-1)+d_A(T-2)}(\cdot)$ and $F_{d_B(T-1)+d_B(T-2)}(\cdot)$ are "far away" from each other if

$$2UB[X_{(A,1)}(T-2)] + (UB[I_A(T-2)] - d_A(T-1) - d_A(T-2))/P_{(A,1)} + 3\sigma_A <$$
$$-2UB[X_{(B,2)}(T-2)] - (UB[I_B(T-2)] - d_B(T-1) - d_B(T-2)/P_{(B,2)} - 3\sigma_B,$$

or

$$2LB[X_{(A,1)}(T-2)] + (LB[I_A(T-2)] - d_A(T-1) - d_A(T-2))/P_{(A,1)} - 3\sigma_A >$$
$$-2LB[X_{(B,2)}(T-2)] - (LB[I_B(T-2)] - d_B(T-1) - d_B(T-2)/P_{(B,2)} + 3\sigma_B,$$

where $LB[\cdot]$ and $UB[\cdot]$ are lower bound and upper bound functions. For general distributions, the "far away" can be characterized by the metrics, such as Hellinger distance of two random variables X and Y defined as follows: $H(X,Y) = (1 - A(X,Y))^{1/2}$, where $A(X,Y) = \int(f(x)g(x))^{1/2}dx$ with $f(x)$ and $g(x)$ being the p.d.f. of X and Y, respectively.

At time $t = 1, \ldots, T-3$, when demands are stochastic, we have results for some special cases in which optimal policy is to switch as much as possible or not to switch at all. We first denote:

$$\begin{aligned}
L_{t,lb} &:= -C^e_{1,A} + C^e_{2,B} - (T-t-1)[P_{(A,1)}C^d_{A,inv} + P_{(B,2)}C^d_{B,back}],\\
L_{t,ub} &:= -C^e_{1,A} + C^e_{2,B} + (T-t-1)[P_{(A,1)}C^d_{A,back} + P_{(B,2)}C^d_{B,inv}],\\
L_{t,um} &:= C^c_{A\to B} - C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^e_{1,A} + C^e_{2,B}\\
&\quad -(T-t-1)[P_{(A,1)}C^d_{A,inv} + P_{(B,2)}C^d_{B,back}],\\
L_{t,lm} &:= -C^c_{B\to A} - C^{mix}_{A,T-1} + C^{mix}_{B,T-1} - C^e_{1,A} + C^e_{2,B}\\
&\quad +(T-t-1)[P_{(A,1)}C^d_{A,back} + P_{(B,2)}C^d_{B,inv}],
\end{aligned}$$

$t = 1, \ldots, T-3.$

One result of an optimal policy at time $t = 1, \ldots, T-3$ is:

- situation 1: if $L_{t,ub} < 0$ and $L_{T-2,8} < 0$, $\mu_t^*(\overline{X}(t)) = X_{(A,1)}(t)$;
- situation 2: if $L_{t,lb} \geq 0$ and $L_{T-2,1} \geq 0$, $\mu_t^*(\overline{X}(t)) = -X_{(B,2)}(t)$;
- situation 3: if $L_{t,lm} < 0 < L_{t,um}$ and $L_{T-2,4} < 0 \leq L_{T-2,5}$, $\mu_t^*(\overline{X}(t)) = 0$.

For example, the optimal policy in situation 1 can be paraphrased as: "If the unit operating cost for A minus that for B is greater than the cumulative unit backlog cost for A plus the cumulative unit inventory cost for B until the last stage, then the optimal policy for each stage until the last stage is to switch as much capacity from A to B as much as possible".

This optimal policy has the following intuitive desirable properties:

- the larger the unit operating cost for A relative to the unit operating cost for B, the more likely it will be to switch capacity from A to B to avoid larger operating costs;
- the smaller the unit backlog cost for A, the more likely it will be to switch capacity from A to B, since there will be a lower penalty for not meeting demand for product A;
- the smaller the unit inventory cost for B, the more likely it will be to switch capacity from A to B, since a lower storage cost is incurred for producing more product B;
- the effect of the holding and backlog costs on the switching decision diminishes as the end of the planning horizon is approached.

The optimal policy in situation 2 is analogous to that in situation 1.

The optimal policy condition in situation 3 involves $C^{mix}_{A,T-1}$ and $C^{mix}_{B,T-1}$. We assume that $L_{T-1,1} < 0 \leq L_{T-1,2}$; then

$$L_{\tau,um} = C^c_{A\to B} - 2C^e_{1,A} + 2C^e_{2,B} - (T-\tau-1)P_{(A,1)}C^d_{A,inv} - (T-\tau-1)P_{(B,2)}C^d_{B,back}$$

and

$$L_{\tau,lm} = -C^c_{B\to A} - 2C^e_{1,A} + 2C^e_{2,B} + (T-\tau-1)P_{(A,1)}C^d_{A,back} + (T-\tau-1)P_{(B,2)}C^d_{B,inv}.$$

The optimal policy can be paraphrased as: "if the sum of the unit switching cost from A to B and twice the unit operating cost for B minus that for A is greater than the sum of the cumulative unit inventory cost for A and the cumulative unit backlog cost for B until the last stage, and if the sum of the unit switching cost from B to A and twice the unit operating cost for A minus that for B is greater that the sum of the cumulative unit inventory cost for B and the cumulative unit backlog cost for A until the last stage, then do not switch at all."

6.3 Discussion

In this chapter, we first presented a case study of fab-level decision making using simulation-based policy iteration. From the simulation results, it is observed that the simulation-based policy iteration can lead to a near optimal policy with only nine features, linear approximation architecture and batch mode least squares solver. The resulting policies also give us some prior knowledge of optimal policies. However, this algorithm can not be extended to more complicated cases of the fab-level decision-making problem, since the control space expands very fast.

Then, we analyzed the structure of the optimal policy for a special one-machine, two-product case of our fab-level decision-making problem. Specially, we presented a procedure for developing the structure of an optimal policy for the one-machine, two-product case with deterministic demands. The resulting optimal policies for three experimental scenarios using this procedure are consistent with those from linear programming. Note that our resulting policies are policies with structure. For the special one-machine, two-product case with random demands, there are also corresponding results.

However, we have not yet applied simulation-based algorithms directly to the testbed example, next we discuss some general implementation issues of applying simulation-based algorithms to the fab-level decision-making problem, including how to reformulate a finite horizon MDP problem into into a weighted cost MDP problem and how to construct a parameterized heuristic policy based on the structure of an optimal policy for the special case.

Model Reformulation

Note that our fab-level decision-making problem is a finite horizon problem; we will convert this finite horizon problem into a weighted cost problem by viewing time as an additional component of the state. Then, we can use the simulation-based algorithms introduced in Chapter 4 and Chapter 5.

In the reformulation, transitions occur from state-time pairs $[i, t]$ to state-time pairs $[j, t+1]$ according to the transition probabilities $p_{ij}(u)$ of the finite horizon problem; the termination state corresponds to the end of the horizon; it is reached in a single transition from any state-time pair of the form $[j, T]$ at a terminal cost $G(j)$ [4]. The reformulation is as follows:

$$\begin{aligned} &J^*([i,T]) = G(i); \qquad\qquad i = 0, \ldots, n-1 \\ &J^*([i,t]) = \min_{u \in \overline{U}([i,t])} \textstyle\sum_{j=0}^{n-1} p_{ij}(u)(g(i,u,j) + J^*([j,t+1])). \end{aligned} \tag{6.25}$$

Note that, in our fab-level decision-making formulation in Chapter 2, the original state is $\overline{X}(t)$, whereas the new augmented state is $(\overline{X}(t), t)$.

Parameterizing Policy

We intend to use simulation-based algorithms to solve our fab-level decision-making problem. One crucial part to implement simulation-based algorithms is to develop a scheme to parameterize the policy. We propose two heuristics. The first one involves dividing the original problem into several local problems, each of which relates to only one type of machine. According to (2.16), only bottleneck operations, the operations minimizing throughputs for products, affect the inventory level of products. Therefore, if a machine with word w, defined by a sequence of operations, does not involve any bottleneck operation, the expansion and allocation of this machine's capacity can be done locally. We call such a machine a *non-bottleneck machine* (*N-machine*). For example, in the simple two-machine, two-product example, the flexible machine with $w = 013$ is an N-machine if neither operation 1 nor operation 3 is a bottleneck operation. Given a machine w, the local state is $X^{(w)} = (X_{(l,i),w}, T_{(w)})$, where $(l,i) \in \mathcal{P}_t \times w$, and the local objective function is

$$E\left[\sum_{t=0}^{T-1}[g^{(w)}(\overline{X}(t), \overline{U}(t))]\right],$$

where

$$g^{(w)}(\overline{X}(t), \overline{U}(t)) = (C_w^a(B_w(t)) + C_w^b(D_w(t)))$$
$$+ \sum_{\{(l,i),(m,j)\}} C_w^c(V_w^{(l,i),(m,j)}(t)) + \sum_{\{(l,i)\}} C_w^e(X_{(l,i),w}(t)) .$$

We can solve these local MDP problems for all w, N-machine or not analytically, using similar procedures as in Section 6.2. We call the resulting policy a *local non-bottleneck policy* (*LN-policy*).

On the other hand, if a machine is utilized by bottleneck operations for all products in each period, the original problem can also be solved locally. We call such a machine a *bottleneck machine* (*B-machine*). In this case, the local state is $X^{(w)} = (X_{(l,i),w}, T_{(w)}, I_{(l)})$, where $(l,i) \in \mathcal{P}_t \times w$ and $l \in \mathcal{P}_t$, and the local objective function is

$$E\left[\sum_{t=0}^{T-1}[g^{(w)}(\overline{X}(t),\overline{U}(t))]\right],$$

where

$$g^{(w)}(\overline{X}(t),\overline{U}(t)) = (C_w^a(B_w(t)) + C_w^b(D_w(t)))$$

$$+ \sum_{\{(l,i),(m,j)\}} C_w^c(V_w^{(l,i),(m,j)}(t)) + \sum_{\{(l,i)\}} C_w^e(X_{(l,i),w}(t)) + \sum_l (C_l^d(I_l(t)) \ .$$

We solve these local MDP problems for all w, B-machine or not, and the resulting policy is called a *local bottleneck policy* (*LB-policy*).

After we solve the above local problems for each machine, we obtain two sets of policies (LN-policy and LB-policy) for each machine. The proposed heuristic policy will choose these resulting LN-policies or LB-policies with some probabilities depending on machine types (N-machine, B-machine, or neither) and operations types (bottleneck operation or not) in each period. The probabilities can be viewed as parameters to represent the policy, and the parameters can be learned by using simulation-based algorithms in Chapter 4 and Chapter 5.

The second way to parameterize the policy involves decomposing the original problem into two problems: an inventory control problem and a capacity expansion and allocation problem, and solve them sequentially. The connections between these two problems are the throughputs in the state equations and the cost model in the objective function. The idea is to first solve the inventory control problem with a new objective function depending on the inventory levels and the desired throughputs, and then construct capacity expansion and allocation policies based on the desired throughputs by only considering the capacity expansion and allocation problem with another new objective function depending on total capacities and capacities allocated to different operations and different products. The obtained policy from the inventory control problem, a sequence of the desired throughouts, can be parameterized using results from inventory control literature. For example, if the objective function of the inventory control problem has no set-up costs for the desired throughputs, a heuristic policy for the lower level can be a generalized base-stock type policy, since the optimal policy for a multi-product inventory control problem under appropriate assumptions is of this type [70]. The policy for the capacity expansion and allocation problem, a sequence of buying, discarding and switching actions, can be parameterized following heuristic policies in the literature [51] [52].

Chapter 7

Summary and Future Research

Summary

This thesis considers developing simulation-based algorithms for sequential decision making problems under uncertainty which formulated as Markov Decision Processes.

Motivated by a capacity expansion and allocation problem in semiconductor manufacturing, we formulated a fab-level decision making problem using a finite-horizon transient MDP model that can integrate life cycle dynamics of the fab and provide a trade-off between immediate and future benefits and costs.

In order to overcome the so-called "curse of dimensionality" and "curse of modeling" associating with the classical MDP solving methodology, we considered a two-step methodology, which first finds the structure of optimal policies for some special cases, in order to construct parameterized heuristic policies for more general cases, and then uses simulation-based algorithms to determine parameters of the heuristic policies. We analyzed the structure of the optimal policy for a special "one-machine, two-product" case for the fab-level decision-making problem, and discussed applicability of simulation-based algorithms.

We developed several simulation-based algorithms for MDPs to overcome the difficulties of "curse of dimensionality" and "curse of modeling", considering both theoretical and practical issues.

For average cost problems, we developed a simulation-based policy iteration (SBPI) algorithm under a unichain assumption, relaxing the common recurrent state assumption. Note that SBPI algorithms can tackle "curse of dimensionality" with respect to state space but not control space, which has been illustrated in Section 6.1.

For weighted cost problems, we developed a new two-timescale simulation-based gradient algorithms based on perturbation analysis, provided a theoretical convergence proof, and compared it with two recently proposed simulation-based gradient algorithms. Simulation-based algorithms based on perturbation analysis need information about transition probabilities, gradient of stage costs, so sometimes they are not applicable, in which cases SPSA algorithms may be better choices. However, when they are applicable, they will lead to better performances than those of SPSA algorithms. Then, we proposed two new Simultaneous Perturbation Stochastic Approximation (SPSA) algorithms for weighted cost problems and verified their effectiveness via simulation. Note that SPSA algorithms only need two sample performance measures for each gradient estimate, so it is generally applicable, and easy to use. However, SPSA algorithms converge slower than algorithms based on perturbation analysis when the latter are applicable.

We also considered a general SPSA algorithm for function minimization and showed its convergence under a weaker assumption: the function does not have to be differentiable.

Future Research

There are many opportunities to expand beyond the research presented here.

In [71], performance sensitivity formulas are derived for semi-Markov decision processes. These results pave the way for developing simulation-based gradient algorithms based on perturbation analysis for semi-Markov decision processes.

Another line of possible future research is the theoretical analysis of general two-timescale simulation-based algorithms, especially global optimization by two-timescale simulation-based algorithms along the lines of [41] [72] [73], and convergence rate analysis of two-timescale algorithms following [74] [28].

As pointed out in Chapter 5, the analysis of SPSA algorithms can be extended to SPSA algorithms with nondifferentiable constraints or correlated noise, as well as to other stochastic approximation algorithms for nondifferentiable function optimization.

For the fab-level decision-making problem, future work includes continuing to construct near-optimal policies with simple structure, to study implementation issues, and to carry out numerical experiments and data analysis for more complicated cases than thoses discussed in Chapter 6.

Appendix

Proof of Proposition 4.2.1

Consider a sequence $\{\phi_0, \phi_1, \ldots\}$ of parameters generated by the following iteration:

$$\phi_{k+1} = \phi_k + \alpha_k R(x_k, \phi_{n(k)}) \tag{8.1}$$

where

$$n(k) = \max\{n_l | k \geq n_l\},$$

$$R(x_k, \phi_{n(k)}) = \nabla g_{i_k}(\phi_{n(k)}) + g_{i_k}(\phi_{n(k)}) z_k + \frac{\nabla p_{i_k i_{k+1}}(\phi_{n(k)})}{p_{i_k i_{k+1}}(\phi_{n(k)})}(L_k + g_{i_k}(\phi_{n(k)})) I_{\{i_{k+1}=i^*\}}, \tag{8.2}$$

and

$$z_{k+1} = \begin{cases} 0, & \text{if} \quad i_{k+1} = i^*; \\ z_k + \frac{\nabla p_{i_k i_{k+1}}(\phi_n(k))}{p_{i_k i_{k+1}}(\phi_{n(k)})}, & \text{otherwise.} \end{cases} \tag{8.3}$$

Then, at time n_l, $\phi_{n_l} = \theta_l$ since

$$\phi_{n_{l+1}} = \phi_{n_l} + \sum_{k=n_l}^{n_{l+1}-1} \alpha_k R(x_k, \phi_{n_l}), \tag{8.4}$$

which implies that $\{\theta_0, \theta_1, \ldots\}$ is a subsequence of $\{\phi_0, \phi_1, \ldots\}$.

Consider the sequence of states $(i_0, i_1, \ldots)$ visited during the execution of the algorithm. Let t_m be the mth time that the state i^* is visited in this sequence. We then have

$$\begin{aligned} \phi_{t_{m+1}} &= \phi_{t_m} + \textstyle\sum_{k=t_m}^{t_{m+1}-1} \alpha_k R(x_k, \phi_{n(k)}) \\ &= \phi_{t_m} + \tilde{\alpha}_m \frac{\nabla \chi(\phi_{t_m})}{E_{\bar{\pi}, \phi_{t_m}}[T]} + \varepsilon_m, \end{aligned} \tag{8.5}$$

where

$$\tilde{\alpha}_m = \sum_{k=t_m}^{t_{m+1}-1} \alpha_k,$$

and

$$\varepsilon_m = \sum_{k=t_m}^{t_{m+1}-1} \alpha_k (R(x_k, \phi_{n(k)}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}).$$

$\nabla\chi(\theta)$ is the gradient of the weighted cost $\chi(\theta)$ with respect to θ and $E_{\bar{\pi},\theta}[T] = E_{\bar{\pi},\theta}[T \mid i_0 = i^*]$ is the mean recurrent time.

We will take the following steps to prove Proposition 4.2.1:

- show that the series $\sum_m \varepsilon_m$ converges;
- show that $\chi(\phi_m)$ converges;
- show that $\chi(\phi_k)$ converges;
- show that $\chi(\theta_l)$ converges with the observation that $\{\theta_0, \theta_1, \ldots\}$ is a subsequence of $\{\phi_0, \phi_1, \ldots\}$.

Preliminary Results

Lemma 8.0.1 *Suppose Assumption 4.1.1 and 4.1.2 hold. Then $\chi(\theta)$ is bounded, twice differentialable, and has bounded first and second derivatives.*

This lemma can be proved following the same argument in the proof of Lemma 2 in [20].

Lemma 8.0.2 *Suppose Assumption 4.1.1, 4.2.2, 4.2.3, and 4.2.1 hold, and let $(i_0, i_1, \ldots)$ be the sequence of states visited during the execution of the two-timescale algorithm, then there exists a constant C and an integer M_0, such* *that for all m and k, $M_0 \le m \le k$,*

$$\begin{aligned}
\| R(x_k, \phi_{n(k)}) \| &\le C(t_{m+1} - t_m), \qquad k \in \{t_m, \dots, t_{m+1}-1\}, \\
\| \phi_{n(k)} - \phi_{t_m} \| &\le \begin{cases} C\beta_{l_m+1}(t_{v_m+1} - t_{v_m}), & t_m \le k < n_{l_m+1}, \\ C\alpha_{t_m}(t_{m+1} - t_m)^2, & n_{l_m+1} \le k < t_{m+1}; \end{cases} \\
\| R(x_k, \phi_{n(k)}) - R(x_k, \phi_{t_m}) \| &\le \begin{cases} C\beta_{l_m+1}(t_{v_m+1} - t_{v_m})^2, & t_m \le k < n_{l_m+1}, \\ C\alpha_{t_m}(t_{m+1} - t_m)^3, & n_{l_m+1} \le k < t_{m+1}; \end{cases}
\end{aligned}$$

where $n_{l_m} = \max\{n_l \mid t_m > n_l\}$ and $v_m = \max\{j \mid t_{j+1} - t_j, j = \underline{m}, \underline{m}+1, \dots, m\}$ with $\underline{m} = \{d \mid n_{l_m} > t_d\}$.

Proof: For $k \in \{t_m, \dots, t_{m+1}-1\}$, by Assumption 4.1.1, $\frac{\nabla p_{i_k i_{k+1}}(\phi_{n(k)})}{p_{i_k i_{k+1}}(\phi_{n(k)})}$ is bounded, say by B_p. Therefore, there exists a constant C_z, such that

$$\| z_k \| \le C_z(t_{m+1} - t_m).$$

In addition, by Assumption 4.1.1, $g_i(\theta)$ and $\nabla g_i(\theta)$ are bounded, say by B_g and B_n. Since $(t_{m+1} - t_m) \ge 1$, there exists a constant C_R, such that
$\| R(x_k, \phi_{n(k)}) \|$

$$\begin{aligned}
&= \left\| \nabla g_{i_k}(\phi_{n(k)}) + g_{i_k}(\phi_{n(k)}) z_k + \tfrac{\nabla p_{i_k i_{k+1}}(\phi_{n(k)})}{p_{i_k i_{k+1}}(\phi_{n(k)})}(L_k + g_{i_k}(\phi_{n(k)})) I_{\{i_{k+1} = i^*\}} \right\| \\
&\le \| \nabla g_{i_k}(\phi_{n(k)}) \| + |g_{i_k}(\phi_{n(k)})| \, \| z_k \| \\
&+ \left\| \tfrac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{n(t_{m+1})})}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{n(t_{m+1})})}(L_{t_{m+1}-1} + g_{i_{t_{m+1}-1}}(\phi_{n(t_{m+1}-1)})) \right\| \\
&\le \| \nabla g_{i_k}(\phi_{n(k)}) \| + |g_{i_k}(\phi_{n(k)})| C_z(t_{m+1} - t_m) \\
&+ \left| \tfrac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{n(t_{m+1}-1)})}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{n(t_{m+1}-1)})} \right| \| L_{t_{m+1}-1} + g_{i_{t_{m+1}-1}}(\phi_{n(t_{m+1}-1)}) \| \\
&\le \| \nabla g_{i_k}(\phi_{n(k)}) \| + |g_{i_k}(\phi_{n(k)})| C_z(t_{m+1} - t_m) \\
&+ B_p \| \textstyle\sum_{k=t_m}^{t_{m+1}-1} g_{i_k}(\phi_{n(t_{m+1}-1)}) \| \\
&\le B_n + B_g C_z(t_{m+1} - t_m) + B_p B_g(t_{m+1} - t_m) \\
&\le C_R(t_{m+1} - t_m).
\end{aligned}$$

Using this result, $\| R(x_j, \phi_{n(j)}) \| \le C_R(t_{v_m+1} - t_{v_m})$ for $j \in \{n_{l_m}, \dots, t_m - 1\}$.

Furthermore, for $k \in \{t_m, \dots, t_{m+1} - 1\}$,

- if $t_m \leq k < n_{l_m+1}$,

$$\begin{aligned}
\| \phi_{n(k)} - \phi_{t_m} \| &= \| \phi_{n_{l_m}} - \phi_{t_m} \| \\
&= \| \textstyle\sum_{j=n_{l_m}}^{t_m-1} \alpha_j R(x_j, \phi_{n_{l_m}}) \| \\
&\leq \textstyle\sum_{j=n_{l_m}}^{t_m-1} \| \alpha_j R(x_j, \phi_{n_{l_m}}) \| \\
&\leq C_R(t_{v_m+1} - t_{v_m}) \textstyle\sum_{j=n_{l_m}}^{t_m-1} \alpha_j \\
&\leq \beta_{l_m} C_R(t_{v_m+1} - t_{v_m}),
\end{aligned}$$

  Note that the difference between $\phi_{n(k)}$ and ϕ_{t_m} depends on cumulative $\alpha R(\cdot)$ from n_{l_m} to $t_m - 1$. For all $j \in \{n_{l_m}, \ldots, t_m - 1\}$, there are N_u recurrence cycles with $j \in \{t_{u_i}, \ldots, t_{u_i+1} - 1\}$ and $i = \{1, \ldots, N_u\}$.

- if $n_{l_m+1} \leq k < t_{m+1}$,

$$\begin{aligned}
\| \phi_{n(k)} - \phi_{t_m} \| &= \| \textstyle\sum_{j=t_m}^{n(k)-1} \alpha_j R(x_j, \phi_{n_j}) \| \\
&\leq \textstyle\sum_{j=t_m}^{n(k)-1} \| \alpha_j R(x_j, \phi_{n_j}) \| \\
&\leq C_R(t_{m+1} - t_m) \textstyle\sum_{j=t_m}^{n(k)-1} \alpha_j \\
&\leq C_R \alpha_{t_m} (t_{m+1} - t_m)^2.
\end{aligned}$$

Therefore,

$$\| \phi_{n(k)} - \phi_{t_m} \| \leq \begin{cases} C\beta_{l_{m+1}}(t_{v_m+1} - t_{v_m}), & t_m \leq k < n_{l_m+1}, \\ C\alpha_{t_m}(t_{m+1} - t_m)^2, & n_{l_m+1} \leq k < t_{m+1}. \end{cases}$$

Finally, by Assumption 4.1.1, there exists a constant L, such that for all θ, θ',

$$\| \nabla g_i(\theta) - \nabla g_i(\theta') \| \leq L \| \theta - \theta' \|, \qquad \| g_i(\theta) - g_i(\theta') \| \leq L \| \theta - \theta' \| .$$

Therefore, $\parallel R(x_k, \phi_{n(k)}) - R(x_k, \phi_{t_m}) \parallel$

$$\begin{aligned}
&\le && \parallel \nabla g_i(\phi_{n(k)}) - \nabla g_i(\phi_{t_m}) \parallel \\
&+ && \parallel g_i(\phi_{n(k)}) - g_i(\phi_{t_m}) \parallel\parallel z_k \parallel \\
&+ && \left\| \frac{\nabla p_{i_k i_{k+1}}(\phi_{n(k)})}{p_{i_k i_{k+1}}(\phi_{n(k)})} (L_k + g_{i_k}(\phi_{n(k)})) I_{\{i_{k+1}=i^*\}} \right\| \\
&+ && \left\| \frac{\nabla p_{i_k i_{k+1}}(\phi_{t_m})}{p_{i_k i_{k+1}}(\phi_{t_m})} (L_k + g_{i_k}(\phi_{t_m})) I_{\{i_{k+1}=i^*\}} \right\| \\
&\le && L \parallel \phi_{n(k)} - \phi_{t_m} \parallel + L \parallel \phi_{n(k)} - \phi_{t_m} \parallel C_z(t_{m+1} - t_m) \\
&+ && \left| \frac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{n(t_{m+1}-1)})}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{n(t_{m+1}-1)})} \right| \parallel \textstyle\sum_{k=t_m}^{t_{m+1}+1} g_{i_k}(\phi_{n(t_{m+1}-1)}) \parallel \\
&+ && \left| \frac{\nabla p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{t_m})}{p_{i_{t_{m+1}-1} i_{t_{m+1}}}(\phi_{t_m})} \right| \parallel \textstyle\sum_{k=t_m}^{t_{m+1}+1} g_{i_k}(\phi_{t_m}) \parallel \\
&\le && L \parallel \phi_{n(k)} - \phi_{t_m} \parallel (1 + C_z(t_{m+1} - t_m)) \\
&+ && 2B_p B_g (t_{m+1} - t_m) \\
&\le && \begin{cases} 2LC_z C_R \beta_{l_{m+1}} (t_{v_m+1} - t_{v_m})^2, & t_m \le k < n_{l_m+1}, \\ 2LC_z C_R \alpha_{t_m} (t_{m+1} - t_m)^3, & n_{l_m+1} \le k < t_{m+1}, \end{cases} \\
&+ && 2B_p B_g (t_{m+1} - t_m) \\
&\le && \begin{cases} C_x \beta_{l_{m+1}} (t_{v_m+1} - t_{v_m})^2, & t_m \le k < n_{l_m+1}, \\ C_y \alpha_{t_m} (t_{m+1} - t_m)^3, & n_{l_m+1} \le k < t_{m+1}. \end{cases} \qquad \square
\end{aligned}$$

Lemma 8.0.3 *Suppose Assumption 4.1.1 and 4.2.3 hold. Let $(i_0, i_1, \ldots)$ be the sequence of states visited during the execution of the two-timescale algorithm, and let t_m be the mth time that the state i^* is visited in this sequence, then for every integer $s > 0$ there exists a constant D_s such that*

$$E[(t_{m+1} - t_m)^2] \le D_s.$$

The proof of this lemma is given in [20].

Lemma 8.0.4 *Suppose Assumption 4.1.1 and 4.1.2 hold. Let $(i_0, i_1, \ldots)$ be the sequence of states visited during the execution of the two-timescale algorithm, and let t_m be the mth time that the state i^* is visited in this sequence, then,*
(a) for any positive integer s,

$$E\left[\sum_{m=1}^{\infty} \alpha_{t_m}^2 (t_{m+1} - t_m)^s\right] < \infty,$$

and

$$E\left[\sum_{m=1}^{\infty} \alpha_{t_m}\beta_{n_{l_m}}(t_{v_m+1} - t_{v_m})^s\right] < \infty;$$

(b) we have

$$\sum_{m=1}^{\infty} \tilde{\alpha}_{t_m} = \infty, \qquad \sum_{m=1}^{\infty} \tilde{\alpha}_{t_m}^2 < \infty,$$

with probability 1.

Proof: (We use a similar argument as the one in the proof of Lemma 14 in [20]). (a) Because α_{t_m} is $\mathcal{F}_m$ measurable, where $\mathcal{F}_m = \{\phi_0, i_0, i_1, \ldots, i_{t_m}\}$ and stands for the history of the algorithm up to and including time t_m, and by Lemma 8.0.3, we have

$$E\left[\alpha_{t_m}^2 (t_{m+1} - t_m)^s\right] = E\left[\alpha_{t_m}^2 E[(t_{m+1} - t_m)^s \mid \mathcal{F}_m]\right] \le E\left[\alpha_{t_m}^2\right] D_s.$$

Therefore,

$$E\left[\sum_{m=1}^{\infty} \alpha_{t_m}^2 (t_{m+1} - t_m)^s\right] \le D_s \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

Using a similar argument,

$$E\left[\sum_{m=1}^{\infty} \alpha_{t_m}\beta_{n_{l_m}}(t_{v_m+1} - t_{v_m})^s\right] \le D_s \sum_{k=1}^{\infty} \alpha_k \beta_k \le D_s (\sum_{k=1}^{\infty} \alpha_k^2 \sum_{k=1}^{\infty} \beta_k^2)^{1/2} < \infty$$

(b) By Assumption 4.2.1, we have

$$\sum_{m=1}^{\infty} \tilde{\alpha}_m = \sum_{k=1}^{\infty} \alpha_k = \infty.$$

Moreover, by Assumption 4.2.2, stepsize α_k is non-increasing, so we have

$$\tilde{\alpha}_m^2 \le \alpha_{t_m}^2 (t_{m+1} - t_m)^2.$$

Using (a), $\sum_{m=1}^{\infty} \tilde{\alpha}_m^2$ has finite expectation and is therefore finite with probability 1. □

Summability of ε_m

In this section, we would like to show that ε_m is summable where

$$\varepsilon_m = \sum_{k=t_m}^{t_{m+1}-1} \alpha_k (R(x_k, \phi_{n(k)}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}).$$

Given the time t_m at which i^* is visited for the mth time, define a "frozen" augmented state $x_F = \{i_k^F, z_k^F\}$ which evolves the same way as x_k except that $\phi_{n(k)}$ is held fixed at ϕ_{t_m} until the next visit at i^*. In addition, we let $t_{m+1}^F = \min\{k > t_m \mid i_k^F = i^*\}$, and define z_{k+1}^F as

$$z_k^F = \begin{cases} 0, & \text{if} \quad i_{k+1}^F = i^*; \\ z_{k+1}^F + \frac{\nabla p_{i_k^F i_{k+1}^F}(\phi_{t_m})}{p_{i_k^F i_{k+1}^F}(\phi_{t_m})} & \text{otherwise.} \end{cases}$$

and we can use similar argument in Lemma 8.0.2 to prove the following lemma:

Lemma 8.0.5 *Let Assumption 4.1.1 and 4.1.2 hold, and let $(i_0, i_1, \ldots)$ be the sequence of states visited during the execution of the two-timescale algorithm. Then there exists a constant C and*

$$\| R(x_k^F, \phi_{t_m}) \| \leq C(t_{m+1}^F - t_m), \qquad k \in \{t_m, \ldots, t_{m+1} - 1\}.$$

Using the definition of x_{m+1}^F and t_{m+1}^F, we rewrite the noise term ε_m as

$$\varepsilon_m = \varepsilon_m^{(1)} + \varepsilon_m^{(2)} + \varepsilon_m^{(3)} + \varepsilon_m^{(4)} + \varepsilon_m^{(5)},$$

where

$$\begin{aligned} \varepsilon_m^{(1)} &= \textstyle\sum_{k=t_m}^{t_{m+1}-1} (\alpha_k - \alpha_{t_m}) \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}, \\ \varepsilon_m^{(2)} &= \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}^F-1} (R(x_k^F, \phi_{t_m}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}), \\ \varepsilon_m^{(3)} &= \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} (R(x_k, \phi_{t_m}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}) \\ &\quad -\alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}^F-1} (R(x_k^F, \phi_{n(k)}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}), \\ \varepsilon_m^{(4)} &= \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} (R(x_k, \phi_{n(k)}) - R(x_k, \phi_{t_m})), \\ \varepsilon_m^{(5)} &= \textstyle\sum_{k=t_m}^{t_{m+1}-1} (\alpha_k - \alpha_{t_m})(R(x_k, \phi_{n(k)})). \end{aligned}$$

We will show that each one of the series $\sum \varepsilon_m^{(n)}$, $n = 1, \ldots, 5$, converges with probability 1.

Lemma 8.0.6 *The series $\sum \varepsilon_m^{(1)}$ converges with probability 1.*

Proof: (This lemma can be proved following the same argument as the one in the proof of Lemma 16 in [20]).

Let B be the bound of $\nabla\chi(\phi_{t_m})$, which exists according to Lemma 2 in [20], and $E_{\bar{\pi},\phi_{t_m}}[T] \geq 1$. Then use Assumption 4.2.2, we have

$$\begin{aligned}
\varepsilon_m^{(1)} &= \textstyle\sum_{k=t_m}^{t_{m+1}-1}(\alpha_k - \alpha_{t_m})\frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]} \\
&\leq \textstyle\sum_{k=t_m}^{t_{m+1}-1}(\alpha_k - \alpha_{t_m}) \parallel \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]} \parallel \\
&\leq B\textstyle\sum_{k=t_m}^{t_{m+1}-1}(\alpha_k - \alpha_{t_m}) \\
&\leq BA(t_{m+1} - t_m)^p \alpha_{t_m}^2.
\end{aligned}$$

Then by Lemma 8.0.4(a), the infinite sum $\sum \varepsilon_m^{(1)}$ has finite expectation, and therefore converges with probability 1. □

Lemma 8.0.7 *The series $\sum \varepsilon_m^{(2)}$ converges with probability 1.*

Proof: (This lemma can be proved using a similar argument as the one in the proof of Lemma 17 in [20]). The basic idea is to prove $\sum \varepsilon_m^{(2)}$ is a martingale with bounded variance. □

Lemma 8.0.8 *The series $\sum \varepsilon_m^{(3)}$ converges with probability 1.*

Proof: (we use a similar argument as the one in the proof of Lemma 18 in [20]). By Assumption 4.1.1, there exists a constant L, such that for $\theta, \theta' \in \mathcal{R}^K$, and all state $i, j \in S$

$$|P_{ij}(\theta) - P_{ij}(\theta')| \leq \parallel \theta - \theta' \parallel .$$

By Lemma 8.0.2, there exists a constant C, such that for all $m \geq 0$, and $k = t_m, \ldots, t_{m+1}$, we have

$$\parallel \phi_{n(k)} - \phi_{t_m} \parallel \;\leq\; \begin{cases} C\beta_{l_{m+1}}(t_{v_m+1} - t_{v_m}), & t_m \leq k < n_{l_m+1}, \\ C\alpha_{t_m}(t_{m+1} - t_m)^2, & n_{l_m+1} \leq k < t_{m+1}. \end{cases}$$

It follows that

$$\begin{aligned}
P(i_{k+1}^F \neq i_{k+1} \mid i_k^F = i_k) &\leq \sum_{i,j\in S} |P_{ij}(\phi_k) - P_{ij}(\phi_{t_m})| \\
&\leq B \parallel \phi_{n(k)} - \phi_{t_m} \parallel \\
&\leq \begin{cases} D\beta_{l_{m+1}}(t_{v_m+1} - t_{v_m}), & t_m \leq k < n_{l_m+1}, \\ D\alpha_{t_m}(t_{m+1} - t_m)^2, & n_{l_m+1} \leq k < t_{m+1}; \end{cases}
\end{aligned}$$

for some constant B and D.

We define $\mathcal{E}_m$ to be:

$$\mathcal{E}_m = \{x_k^F \neq x_k \text{for some } k = t_m, \ldots, t_{m+1}\}.$$

Using previous result, we have

$$\begin{aligned}
P(\mathcal{E}_m \mid t_m, t_{m+1}) &\leq \begin{cases} D\sum_{k=t_m}^{t_{m+1}-1} \beta_{l_{m+1}}(t_{v_m+1} - t_{v_m}), & t_m \leq k < n_{l_m+1}, \\ D\sum_{k=t_m}^{t_{m+1}-1} \alpha_{t_m}(t_{m+1} - t_m)^2, & n_{l_m+1} \leq k < t_{m+1}; \end{cases} \\
&\leq \begin{cases} D\beta_{l_{m+1}}(t_{v_m+1} - t_{v_m})^2, & t_m \leq k < n_{l_m+1}, \\ D\alpha_{t_m}(t_{m+1} - t_m)^3, & n_{l_m+1} \leq k < t_{m+1}. \end{cases}
\end{aligned}$$

In addition, if the event $\mathcal{E}_m$ does not occur, then $\varepsilon_m^{(3)} = 0$. Therefore,

$$E[\parallel \varepsilon_m^{(3)} \parallel \mid t_m, t_{m+1}] = P(\mathcal{E}_m \mid t_m, t_{m+1}) E[\parallel \varepsilon_m^{(3)} \parallel \mid t_m, t_{m+1}, \mathcal{E}_m]$$

Let B be the bound of $\nabla\chi(\phi_{t_m})$, which exists according to Lemma 2 in [20], and $E_{\bar{\pi},\phi_{t_m}}[T] \geq 1$. Then use Assumption 4.2.2, we have

$$\begin{aligned}
\varepsilon_m^{(3)} &= \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} \left(R(x_k, \phi_{t_m}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}\right) \\
&\quad -\alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}^F-1} \left(R(x_k^F, \phi_{n(k)}) - \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]}\right), \\
&\leq \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} \left[\parallel R(x_k, \phi_{t_m} \parallel - \parallel \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]} \parallel\right] \\
&\quad +\alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}^F-1} \left[\parallel R(x_k^F, \phi_{n(k)} \parallel - \parallel \frac{\nabla\chi(\phi_{t_m})}{E_{\bar{\pi},\phi_{t_m}}[T]} \parallel\right], \\
&\leq \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} [C'(t_{m+1} - t_m) + B] + \alpha_{t_m} \sum_{k=t_m}^{t_{m+1}^F-1} [C'(t_{m+1}^F - t_m) + B] \\
&\leq \alpha_{t_m} C((t_{m+1} - t_m)^2 + (t_{m+1}^F - t_m)^2),
\end{aligned}$$

for some constant C and C'. Thus,

$$E[\| \varepsilon_m^{(3)} \| \mid t_m, t_{m+1}, \mathcal{E}_m] \leq \gamma_{t_m} C \left((t_{m+1} - t_m)^2 + E[(t_{m+1}^F - t_m)^2 \mid t_m, t_{m+1}, \mathcal{E}_m] \right).$$

By the same argument used to prove Lemma 13 in [20], there exists a constant D_2

$$\begin{aligned} E[(t_{m+1}^F - t_m)^2 \mid t_m, t_{m+1}, \mathcal{E}_m] &\leq 2E[(t_{m+1}^F - t_m)^2 \mid t_{m+1}, \mathcal{E}_m] + 2(t_{m+1} - t_m)^2 \\ &\leq D_2 + 2(t_{m+1} - t_m)^2. \end{aligned}$$

Since $(t_{m+1} - t_m) \geq 1$,

$$E[(t_{m+1}^F - t_m)^2 \mid t_m, t_{m+1}, \mathcal{E}_m] \leq C(t_{m+1} - t_m)^2,$$

for some constant C.

$$E[\| \varepsilon_m^{(3)} \| \mid t_m, t_{m+1}, \mathcal{E}_m] \leq C\gamma_{t_m}(t_{m+1} - t_m)^2,$$

and

$$E[\| \varepsilon_m^{(3)} \| \mid t_m, t_{m+1}] \leq \begin{cases} DC\alpha_{t_m}\beta_{l_{m+1}}(t_{v_m+1} - t_{v_m})^4, & t_m \leq k < n_{l_m+1}, \\ DC\alpha_{t_m}^2(t_{m+1} - t_m)^5, & n_{l_m+1} \leq k < t_{m+1}; \end{cases}$$

Then by Lemma 8.0.4(a), the infinite sum $\sum \varepsilon_m^{(3)}$ has finite expectation, and therefore converges with probability 1. $\square$

Lemma 8.0.9 *The series $\sum \varepsilon_m^{(4)}$ converges with probability 1.*

Proof: (we use a similar argument as the one in the proof of Lemma 19 in [20]).

By Lemma 8.0.2, we have

$$\begin{aligned} \varepsilon_m^{(4)} &= \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} (R(x_k, \phi_{n(k)}) - R(x_k, \phi_{t_m})) \\ &\leq \alpha_{t_m} \textstyle\sum_{k=t_m}^{t_{m+1}-1} \| R(x_k, \phi_{n(k)}) - R(x_k, \phi_{t_m}) \| \\ &\leq \begin{cases} C\alpha_{t_m}\beta_{l_{m+1}}(t_{v_m+1} - t_{v_m})^2, & t_m \leq k < n_{l_m+1}, \\ C\alpha_{t_m}^2(t_{m+1} - t_m)^3, & n_{l_m+1} \leq k < t_{m+1}. \end{cases} \end{aligned}$$

By Lemma 8.0.4(a), the infinite sum $\sum \varepsilon_m^{(4)}$ has finite expectation. Therefore, it converges with probability 1. $\square$

Lemma 8.0.10 *The series $\sum \varepsilon_m^{(5)}$ converges with probability 1.*

Proof: (Use a similar argument as the one in the proof of Lemma 20 in [20]).

By Lemma 8.0.2 and Assumption 4.2.2, we have

$$\begin{aligned}
\varepsilon_m^{(5)} &= \textstyle\sum_{k=t_m}^{t_{m+1}-1}(\alpha_k - \alpha_{t_m})(R(x_k, \phi_{n(k)})) \\
&\leq \parallel (R(x_k, \phi_{n(k)})) \parallel \textstyle\sum_{k=t_m}^{t_{m+1}-1}(\alpha_k - \alpha_{t_m}) \\
&\leq AC\alpha_{t_m}^2 (t_{m+1} - t_m)^{p+1},
\end{aligned}$$

for some constants A and C, and for some positive integer p.

By Lemma 8.0.4(a), the infinite sum $\sum \varepsilon_m^{(5)}$ has finite expectation. Therefore, it converges with probability 1. □

Converges of $\chi(\phi_k)$ and $\nabla(\phi_k)$

Lemma 8.0.11 *Suppose Assumption 4.1.1, 4.1.2, and 4.2.1 hold, we can show that*

$$\sum_{m=0}^{\infty} \frac{\tilde{\alpha}_m}{E_{\bar{\pi},\phi_{t_m}}[T]} = \infty,$$

$$\sum_{m=0}^{\infty} (\frac{\tilde{\alpha}_m}{E_{\bar{\pi},\phi_{t_m}}[T]})^2 < \infty.$$

The proof of this lemma is given in [20] (Lemma 21).

This lemma says that the iteration of Eq.(8.5) is a gradient algorithm with step sizes α_m are deterministic, nonnegative, and satisfying $\sum_{m=1}^{\infty} \alpha_m = \infty$ and $\sum_{m=1}^{\infty} \alpha_m^2 < \infty$. By the same argument used to prove the Proposition 1 [20], we have

$$\lim_{m\to\infty} \parallel \nabla\chi(\phi_{t_m}) \parallel = 0$$

with probability 1.

By Lemma 8.0.2 and Lemma 8.0.4, for every $\varepsilon > 0$, there exists an integer m_0, such that for all $m \geq m_0$, and all integers $k \in \{t_m, \ldots, t_{m+1}\}$, we have

$$\parallel \phi_k - \phi_{t_m} \parallel \leq \varepsilon.$$

By Lemma 8.0.1, there exists a constant L such that for all $\theta, \theta' \in \mathcal{R}^K$, we have

$$|\chi(\phi_k) - \chi(\phi_{t_m})| \leq L \parallel \phi_k - \phi_{t_m} \parallel, \qquad \parallel \nabla\chi(\phi_k) - \nabla\chi(\phi_{t_m}) \parallel \leq \parallel \phi_k) - \phi_{t_m} \parallel .$$

Therefore, for every $\varepsilon > 0$, there exists an integer m_0, such that for all $m \geq m_0$, and all integers $k \in \{t_m, \ldots, t_{m+1}\}$, we have

$$|\chi(\theta) - \chi(\theta')| \leq \varepsilon, \qquad \parallel \nabla\chi(\theta) - \nabla\chi(\theta') \parallel \leq \varepsilon.$$

It follows that

$$\lim_{k\to\infty} \parallel \nabla\chi(\phi_k) \parallel = 0$$

with probability 1.

Finally, $\chi(\theta_l)$ converges because $\{\theta_0, \theta_1, \ldots\}$ is a subsequence of $\{\phi_0, \phi_1, \ldots\}$.

Proof of Proposition 5.2.1

Define $G^k(t) = \int_0^t \widetilde{sf}(\theta^k(s))ds$, and let B be the bound of $\widetilde{sf}(\theta^k(s))$, which exists due to the boundedness of $\partial f(\theta)$. Then, for each T and $\varepsilon > 0$, there is a δ which satisfies $0 < \delta < \varepsilon/B$, such that for all k, $\sup_{0<t-s<\delta,|t|<T} |G^k(t) - G^k(s)| \leq \sup_{0<t-s<\delta,|t|<T} \int_t^s \widetilde{sf}(\theta^k(u))du < \varepsilon$, which means $G^k(\cdot)$ is equicontinuous.

By (A.3) and (A.4), there is a null set O such that for sample points $\omega \notin O$, $M^k(\omega, \cdot)$ and $B^k(\omega, \cdot)$ go to zero uniformly on each bounded interval in $(-\infty, \infty)$ as $k \to \infty$. Hence, $M^k(\cdot)$ and $B^k(\cdot)$ are equicontinuous and their limits are zero.

By the same argument as in the proof of Theorem 5.2.1 in [64, pp. 96-97], $Z^k(\cdot)$ is equicontinuous.

$\theta^k(\cdot)$ is equicontinuous since $M^k(\cdot)$, $B^k(\cdot)$,$G^k(\cdot)$ and $Z^k(\cdot)$ are equicontinuous.

Let $\omega \notin O$ and let k_j denote a subsequence such that

$$\{\theta^{k_j}(\omega, \cdot), G^{k_j}(\omega, \cdot)\}$$

converges, and denote the limit by $(\theta(\omega, \cdot), G(\omega, \cdot))$. The existence of such subsequences is guaranteed by Arzela-Ascoli theorem.

Since f is convex, according to Corollary 24.5.1 [65, p. 234], $\forall \varepsilon > 0$, $\exists\ \delta > 0$, if $|\theta^{k_j}(\omega, s) - \theta(\omega, s)| < \delta$, then $-\partial f(\theta^{k_j}(\omega, s)) \subset N_\varepsilon(-\partial f(\theta(\omega, s)))$, where $N_\varepsilon(\cdot)$ means ε-neighborhood.

Furthermore, since $\lim_{j\to\infty} \theta^{k_j}(\omega, s) = \theta(\omega, s)$ for fixed ω and s, $\forall \varepsilon > 0$, $\exists$ finite J, if $j \geq J$, $-\partial f(\theta^{k_j}(\omega, s)) \subset N_\varepsilon(-\partial f(\theta(\omega, s)))$, i.e. for each $\widetilde{sf}(\theta^{k_j}(\omega, s)) \in -\partial f(\theta^{k_j}(\omega, s))$ and $\varepsilon > 0$, there is finite J and $\tilde{g}(\omega, s) \in -\partial f(\theta(\omega, s))$ such that if $j \geq J$, $|\widetilde{sf}(\theta^{k_j}(\omega, s)) - \tilde{g}(\omega, s)| < \varepsilon$.

Since $\widetilde{sf}(\cdot)$ and $\tilde{g}(\omega, \cdot)$ are bounded functions on $[0, t]$, by the Lebesgue dominated convergence theorem,

$$\lim_{j\to\infty} \int_0^t \widetilde{sf}(\theta^{k_j}(\omega, s))ds = \int_0^t \tilde{g}(\omega, s)ds,$$

which means $G(\omega, t) = \int_0^t \tilde{g}(\omega, s)ds$.

Thus, we can write $\theta(\omega, t) = \theta(\omega, 0) + \int_0^t \tilde{g}(\omega, s)ds + Z(\omega, t)$, where $\tilde{g}(\omega, s) \in -\partial f(\theta(\omega, s))$.

Using a similar argument as in [64, p. 97], $Z(\omega, t) = \int z(\omega, s)ds$, where $z(\omega, s) \in -C(\theta(\omega, s))$ for almost all s.

Hence, the limit $\theta(\omega, \cdot)$ of any convergent subsequence satisfies the differential inclusion (5.20).

Note that S_H is the set of stationary points of (5.20) in H. Following a similar argument as in the proof of Theorem 5.2.1 [64, pp. 95-99], we can show that $\{\theta_k(\omega)\}$ visits S_H infinite often, S_H is asymptotically stable in the sense of Lyapunov due to the convexity of f, and thus $\{\theta_k(\omega)\}$ converges to S_H w. p. 1. Since f is a convex function and H is a non-empty convex set, by Theorem 27.4 in [65, pp. 270-271], any point in S_H attains the minimum of f relative to H. $\square$

Bibliography

[1] S. Bermon, G. Feigin, and S. Hood, “Capacity analysis of complex manufacturing facilities,” in *Proc. of the 34th Conference on Decision and Control*, New Orleans, LA, 1995, pp. 1935–1940.

[2] M. Zweben and M. S. Fox, *Intelligent Scheduling*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1994.

[3] M. L. Puterman, *Markov Decision Processes*, John Wiley & Sons, Inc., New York, 1994.

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

[5] P. Marbach, O. Mihatsch, and J. N. Tsitsiklis, “Call admission control and routing in integrated services networks using neuro-dynamic programming,” *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 2, pp. 197–208, February 2000.

[6] V. R. Konda and V. S. Borkar, “Actor-critic-type learning algorithms for Markov decision processes,” *SIAM Journal on Control and Optimization*, vol. 38, pp. 94–123, November/December 1999.

[7] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Muller, Eds., Denver, Colorado, 1999, pp. 1008–1014.

[8] R. S. Sutton, D. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Muller, Eds., Denver, Colorado, 1999, pp. 1057–1063.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, The MIT Press, Cambridge, 1998.

[10] Y. He, M. C. Fu, and S. I. Marcus, "A simulation-based policy iteration algorithm for average cost unichain Markov decision processes," in *Computing Tools for Modeling, Optimization and Simulation - Interfaces in Computer Science and Operations Research*, M. Laguna and J. L. Gonzalez-Velarde, Eds., Boston/Dordrecht/London, 2000, pp. 161–182, Kluwer Academic Publishers.

[11] Y. He, M. C. Fu, and S. I. Marcus, "Simulation-based algorithms for average cost Markov decision processes," Tech. Rep., University of Maryland at College Park, 1999, ISR Technical Report, TR-99-56.

[12] X.-R. Cao, "Single sample path based optimization of Markov chains," *Journal of Optimization Theory and Applications*, vol. 100, no. 3, pp. 527–548, March 1999.

[13] J. N. Tsitsiklis and B. Van Roy, "Average cost temporal-difference learning," *Automatica*, vol. 35, no. 11, pp. 1799–1808, November 1999.

[14] D. P. Bertsekas, *Dynamic Programming and Optimal Control Vol 1 & 2*, Athena Scientific, Belmont, 1995.

[15] C. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, May 1992.

[16] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Machine learning*, vol. 16, pp. 185–202, September 1994.

[17] J. Abounadi, D. P. Bertsekas, and V. S. Borkar, "Learning algorithms for Markov decision processes with average cost," *SIAM Journal on Control and Optimization*, vol. 40, no. 3, pp. 681–698, 2001.

[18] B. Van Roy, D. P. Bertsekas, P. Lee, and J. N. Tsitsiklis, "A neuro-dynamic programming approach to retailer inventory management," Tech. Rep., Unica Technologies, 1997, Lincoln, MA.

[19] D. P. Bertsekas, M. L. Homer, D. A. Logan, S. D. Patek, and N. R. Sandell, "Missile defense and interceptor allocation by neuro-dynamic programming," *IEEE Transactions on Systems, Man and Cybernetics - A*, vol. 30, no. 1, pp. 42–51, January 2000.

[20] P. Marbach, *Simulation-Based Optimization of Markov Decision Processes*, Ph.D. thesis, Massachusetts Institute of Technology, 1998.

[21] E. K. P. Chong and P. J. Ramadge, "Stochastic optimization of regenerative systems using infinitesimal perturbation analysis," *IEEE Transactions on Automatic Control*, vol. 39, no. 7, pp. 1400–1410, July 1994.

[22] M. C. Fu, "Convergence of a stochastic approximation algorithm for the GI/G/1 queue using infinitesimal perturbation analysis," *Journal of Optimization Theory and Applications*, vol. 65, no. 1, pp. 149–160, April 1990.

[23] P. L'Ecuyer and P. W. Glynn, "Stochastic optimization by simulation: convergence proof for the GI/G/1 queue in steady-state," *Management Science*, vol. 40, no. 11, pp. 1562–1578, November 1994.

[24] E. K. P. Chong and P. J. Ramadge, "Optimization of queues using a infinitesimal perturbation analysis-based stochastic algorithm with general

update times," *SIAM Journal on Control and Optimization*, vol. 31, no. 3, pp. 698–732, May 1993.

[25] Y.-C. Ho and X.-R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[26] P. Glasserman, *Gradient Estimation Via Perturbation Analysis*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[27] M. C. Fu and J.-Q. Hu, *Conditional Monte Carlo: Gradient Estimation and Optimization Applications*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1997.

[28] V. S. Borkar, "Stochastic approximation with two time scales," *Systems and Control Letters*, vol. 29, no. 5, pp. 291–294, February 1997.

[29] A. Ruszczyński and W. Syski, "Stochastic approximation method with gradient averaging for unconstrained problems," *IEEE Transactions on Automatic Control*, vol. 28, no. 12, pp. 1097–1105, December 1983.

[30] S. Bhatnagar and V. S. Borkar, "A two timescale stochastic approximation scheme for simulation-based parametric optimization," *Probability in the Engineering and Information Sciences*, vol. 12, pp. 519–531, 1998.

[31] S. Bhatnagar and V. S. Borkar, "Multiscale stochastic approximation for parametric optimization of hidden Markov models," *Probability in the Engineering and Information Sciences*, vol. 11, pp. 509–522, 1997.

[32] S. Bhatnagar, M. C. Fu, S. I. Marcus, and P. J. Fard, "Optimal structured feedback policies for abr flow control using two timescale SPSA," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 479–491, August 2001.

[33] S. Bhatnagar, M. C. Fu, S. I. Marcus, and S. Bhatnagar, "Two timescale algorithms for simulation optimization of hidden markov models," *IIE Transactions*, vol. 33, no. 3, pp. 245–258, March 2001.

[34] V. S. Borkar and V. R. Konda, "The actor critic algorithm as multi-time scale stochastic approximation," *Sadhana*, vol. 22, pp. 525–43, August 1997.

[35] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, March 1992.

[36] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins APL Technical Digest*, vol. 19, no. 4, pp. 482–492, 1998.

[37] H. J. Kushner and D. C. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained systems*, Springer-Verlag, New York, 1978.

[38] J. Dippon and J. Renz, "Weighted means in stochastic approximation of minima," *SIAM Journal on Control and Optimization*, vol. 35, no. 5, pp. 1811–1827, September 1997.

[39] L. Gerencsér, "Convergence rate of moments in stochastic approximation with simultaneous perturbation gradient approximation and resetting," *IEEE Transactions on Automatic Control*, vol. 44, no. 5, pp. 894–905, May 1999.

[40] N. L. Kleinman, J. C. Spall, and D. Q. Naiman, "Simulation-based optimization with stochastic approximation using common random numbers," *Management Science*, vol. 45, no. 11, pp. 1570–1578, November 1999.

[41] J. L. Maryak and D. C. Chin, "Stochastic approximation for global random optimization," in *Proceedings of the American Control Conference*, Chicago, IL, 2000, pp. 3294–3298.

[42] J. C. Spall, "A one-measurement form of simultaneous perturbation stochastic approximation," *Automatica*, vol. 33, no. 1, pp. 109–112, January 1997.

[43] J. C. Spall, "Adaptive stochastic approximation by the simultaneous perturbation method," *IEEE Transactions on Automatic Control*, vol. 45, no. 10, pp. 1839–1853, October 2000.

[44] H. F. Chen, T. E. Duncan, and B. Pasik-Duncan, "A stochastic approximation algorithm with random differences," in *Proceedings of the 13th IFAC World Congress*, San Francisco, California, 1996, vol. H, pp. 493–496.

[45] M. C. Fu and S. D. Hill, "Optimization of discrete event systems via simultaneous perturbation stochastic approximation," *IIE Transactions*, vol. 29, no. 3, pp. 233–243, March 1997.

[46] P. Sadegh, "Constrained optimization via stochastic approximation with a simultaneous perturbation gradient approximation," *Automatica*, vol. 33, no. 5, pp. 889–892, May 1997.

[47] J. C. Spall and J. A. Cristion, "Model-free control of nonlinear stochastic systems with discrete-time measurements," *IEEE Transactions on Automatic Control*, vol. 43, no. 9, pp. 1198–1210, September 1998.

[48] I. J. Wang and E. K. P. Chong, "A deterministic analysis of stochastic approximation with randomized directions," *IEEE Transactions on Automatic Control*, vol. 43, no. 12, pp. 1745–1749, December 1998.

[49] R. Uzsoy, C.-Y. Lee, and L.A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry, Part I: System

characterization, performance evaluation and production planning," *IIE Transactions*, vol. 24, no. 4, pp. 47–60, September 1992.

[50] R. Uzsoy, C.-Y. Lee, and L.A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry, Part II: Shop-floor control," *IIE Transactions*, vol. 26, no. 5, pp. 44–55, September 1994.

[51] S. Li and D. Tirupati, "Dynamic capacity expansion problem with multiple products: technology selection and timing of capacity additions," *Operations Research*, vol. 42, no. 5, pp. 958–976, September/October 1994.

[52] S. Li and D. Tirupati, "Technology choice with stochastic demands and dynamic capacity allocation: A two-product analysis," *Journal of Operations Management*, vol. 12, pp. 239–258, June 1995.

[53] S. Li and D. Tirupati, "Impact of product mix flexibility and allocation policies on technology," *Computers Ops. Research*, vol. 24, no. 7, pp. 611–626, July 1997.

[54] M. Swaminathan, "Tool capacity planning for semiconductor fabrication facilities under demand uncertainty," *European Journal of Operational Research*, vol. 120, no. 3, pp. 545–558, February 2000.

[55] A. Angelus, E. L. Porteus, and S. C. Wood, "Optimal sizing and timing expansions with implications for modular semiconductor wafer fabs," preprint, Strategic Decision Group, Inc., 1997.

[56] G. E. Feigin, K. Katircioglu, and D. D. Yao, "Capacity allocation in semiconductor fabrication," in *Proc. of the 38th Conference on Decision and Control*, Pheonix, Arizona, 1999, pp. 1364–1369.

[57] D. P. Connors, G. E. Feigin, and D. D. Yao, "A queueing network model for semiconductor manufacturing," *IEEE Transactions on Semiconductor Manufacturing*, vol. 9, no. 3, pp. 412–427, August 1996.

[58] S. Bhatnagar, M. C. Fu, S. I. Marcus, and Y. He, "Markov decision processes for semiconductor fab-level decision making," in *Proc. of the IFAC 14th Triennial World Congress*, Beijing, P. R. China, 1999, pp. 145–150.

[59] E. Fernadez-Gaucherand, J. Choi, and D. Gerhart, "SYSCODE: Stochastic Systems Control and Decision Algorithms Software Laboratory, FORTRAN and MATLAB versions," The University of Arizona, (cimarron.sie.arizona.edu /modeling /modeling.html), 1998.

[60] M. C. Fu, "Sample path derivatives for (s,S) inventory systems," *Operations Research*, vol. 42, no. 2, pp. 351–364, 1994.

[61] L. Gerencsér, G. Kozmann, and Z. V*ágó*, "SPSA for non-smooth optimization with application in ecg analysis," in *Proceedings of the IEEE Conference on Decision and Control*, Tampa, Florida, 1998, pp. 3907–3908.

[62] P. L'Ecuyer, N. Giroux, and P. W. Glynn, "Stochastic optimization by simulation: numerical experiments with a simple queue in steady-state," *Management Science*, vol. 40, no. 10, pp. 1245–1261, October 1994.

[63] J. C. Spall, "Implementation of the simultaneous perturbation algorithm for stochastic optimization," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 817–823, July 1998.

[64] H. J. Kushner and G. George Yin, *Stochastic Approximation Algorithms and Applications*, Springer-Verlag, New York, 1997.

[65] R. T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, New Jersey, 1970.

[66] S. Bhatnagar, E. Fernandez-Gaucherand, M. C. Fu, Y. He, and S. I. Marcus, "A Markov decision process capacity expansion and allocation," in *Proc. of the 38th Conference on Decision and Control*, Phoenix, Arizona, 1999, pp. 1156–1161.

[67] A. M. Law and W. D. Kelton, *Simulation Modeling & Analysis*, McGraw - Hill, Inc., New York, 1991.

[68] P. M. Fitzpatrick, *Advanced Calculus - A course in mathematical analysis*, PWS Publishing Company, 1996.

[69] A. W. Roberts and D. E. Varberg, *Convex Functions*, Academic Press, New York and London, 1973.

[70] D. Beyer, S. P. Sethi, and R. Sridhar, "Stochastic multi-product inventory models with limited storage," *Journal of Optimization Theory and Applications*, vol. 111, no. 3, pp. 553–588, December 2001.

[71] X.-R. Cao, "Semi-Markov decision problems and performance sensitivity analysis," preprint, 2002.

[72] H. J. Kushner, "Asymptotic global behavior for stochastic approximation and diffusions with slowly decreasing noise effects: Global minimization via monto carlo," *SIAM Journal on Applied Mathematics*, vol. 47, pp. 169–185, February 1987.

[73] S. B. Gelfand and S. K. Mitter, "Recursive stochastic algorithms for global optimization in R^d," *SIAM Journal on Control and Optimization*, vol. 29, no. 5, pp. 999–1018, September 1991.

[74] G. Yin, "Rates of convergence for a class of global stochastic optimization algorithms," *SIAM Journal on Optimization*, vol. 10, no. 1, pp. 99–120, 1999.