universität wien

DISSERTATION

Titel

Optimization of Stochastic-Dynamic Decision Problems with Applications in Energy and Production Systems

Verfasser

Dipl.-Kfm. Nils Löhndorf

angestrebter akademischer Grad

Doctor of Philosophy (PhD)

Wien, Oktober 2011

Studienkennzahl lt. Studienblatt:	A 094 146
Dissertationsgebiet lt. Studienblatt:	PhD Management
Betreuer:	Univ.-Prof. Dr. Stefan Minner

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der mathematischen Optimierung von stochastisch-dynamischen Entscheidungsproblemen. Diese Problemklasse stellt eine besondere Herausforderung für die mathematische Optimierung dar, da bislang kein Lösungsverfahren bekannt ist, das in polynomieller Zeit zu einer exakten Lösung konvergiert. Alle generischen Verfahren der dynamischen Optimierung unterliegen dem sogenannten *Fluch der Dimensionen*, der dazu führt, dass die Problemkomplexität exponentiell in der Anzahl der Zustandsvariablen zunimmt. Da Entscheidungsprobleme von realistischer Größenordnung meist über eine Vielzahl von Zustandsvariablen verfügen, stoßen exakte Lösungsverfahren schnell an ihre Grenzen.

Einen vielversprechenden Ausweg, um dem Fluch der Dimensionen zu entgehen, stellen Verfahren der *approximativ-dynamischen Optimierung* dar (engl.: *approximate dynamic programming*), welche versuchen eine Nährungslösung des stochastisch-dynamischen Problems zu berechnen. Diese Verfahren erzeugen eine künstliche Stichprobe des Entscheidungsprozesses mittels Monte-Carlo-Simulation und konstruieren basierend auf dieser Stichprobe eine Approximation der Wertfunktion des dynamischen Problems. Dabei wird die Stichprobe so gewählt, dass lediglich diejenigen Zustände in die Stichprobe aufgenommen werden, welche für den Entscheidungsprozess von Bedeutung sind, wodurch eine vollständige Enumeration des Zustandsraums vermieden wird. In dieser Arbeit werden Verfahren der approximal-dynamischen Optimierung auf verschiedene Probleme der Produktions- und Energiewirtschaft angewendet und daraufhin überprüft, ob sie in der Lage sind, das zugrundeliegende mathematische Optimierungproblem nährungsweise zu lösen.

Die Arbeit kommt zu dem Ergebnis, dass sich komplexe stochastisch-dynamische Bewirtschaftungsprobleme effizient lösen lassen, sofern das Optimierungsproblem konvex und der Zufallsprozess unabhängig vom Entscheidungsprozess ist. Handelt es sich hingegen um ein diskretes Optimierungsproblem, so stoßen auch Verfahren der approximativ-dynamischen Optimierung an ihre Grenzen. In diesem Fall sind gut kalibrierte, einfache Entscheidungsregeln möglicherweise die bessere Alternative.

Abstract

This thesis studies mathematical optimization methods for stochastic-dynamic decision problems. This problem class is particularly challenging, as there still exists no algorithm that converges to an exact solution in polynomial time. Existing generic solution methods are all subject to the *curse of dimensionality*, which means that problem complexity increases exponentially in the number of state variables. Since problems of realistic size typically come with a large number of state variables, applying exact solution methods is impractical.

A promising methodology to break the curse of dimensionality is *approximate dynamic programming*. To avoid a complete enumeration of the state space, solution techniques based on this methodology use Monte Carlo simulation to sample states that are relevant to the decision process and then approximate the value function of the dynamic program by a function of much lower complexity. This thesis applies approximate dynamic programming techniques to different resource management problems that arise in production and energy settings and studies whether these techniques are capable of solving the underlying optimization problems.

The thesis concludes that stochastic-dynamic resource management problems can be solved efficiently if the underlying optimization problem is convex and randomness independent of the resource states. If the optimization problem is discrete, however, the problem remains hard to solve, even for approximate dynamic programming techniques. In this case, simple but well-adjusted decision policies may be the better choice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many decision-makers face the problem that they have to manage resources over time while environmental conditions are changing randomly. For example, a financial portfolio manager has to find the right timing to buy and sell an asset, not knowing how prices will develop. A retailer has to order on time, not knowing how demand will be in the future. The operator of a hydro reservoir needs to decide how much water to release, not knowing how much water will arrive during the next month. All of these problems are classic examples of stochastic-dynamic decision problems. We make decisions not knowing how things will develop, then observe new information, and again make decisions not knowing how things will develop, and so on. It may be easy to formulate these problems in terms of a mathematical model, but optimizing them is a different matter.

1.1 Motivation and Research Questions

To date, no reasonably fast generic solution method is known which is able to solve stochastic-dynamic decision problems to optimality. The research outlined in this thesis is motivated by recent progress in developing new methods that approximately solve these problems, or, more specifically, a subclass of stochastic-dynamic decision problems known as *Markov decision processes* (MDPs). Like any stochastic-dynamic decision problem, MDPs have the unpleasant property that they are getting increasingly difficult to solve as the number of decision states increases, a phenomenon also known as the *curse of dimensionality* of dynamic programs. A promising methodology to break the curse of dimensionality is *approximate dynamic programming* (ADP), also known as *neuro-dynamic programming*, which combines Monte Carlo simulation with function approximation techniques.

ADP has its roots in computer science in the field of artificial intelligence, where it is referred to as *reinforcement learning*. From an artificial intelligence perspective, a decision-maker or *agent* learns to make the right decisions by interacting with his environment. By repeatedly making decisions and observing the consequences of these decisions, the agent creates a model in his mind which guides his decisions in the future. The aim of reinforcement learning is to artificially reconstruct the process of learning to enable machines to make decisions in a complex and uncertain environment. In ADP, the idea of learning is a key concept and still an active area of research. Here, learning takes place by sampling the decision process and collecting information about the relationship between actions and outcomes. This information is then used to construct a mathematical model of this relationship which is an approximation of the true, but unknown causal mechanism. In contrast to reinforcement learning, ADP takes a more formal perspective on the underlying stochastic-dynamic decision problem and combines the idea of learning with the mathematical rigor of operations research.

The objective of the research outlined in this thesis is to apply ADP to complex resource management problems which cannot be solved by generic solution methods in reasonable time. Since each solution strategy has to be tailored to the problem at hand, different solution strategies are applied to different resource management problems that frequently arise in production and energy settings. All of these problems have in common that the size of the state space is driven by the number of resources that have to be managed over time. The problems differ in the way decisions are made and whether the resource states are discrete or continuous. With these problems in mind, the thesis aims at exploring the boundaries of ADP to answer the following research questions:

1. Which ADP technique is most useful for which problem? ADP is not a generic solution method and there exists various algorithms and paradigms within the field that need to be explored to find the technique that fits the problem. Answering this question involves experimenting with different techniques as well as carefully tuning algorithmic parameters.

2. How does ADP compare to other methods? Solutions based on ADP are only competitive if they represent a significant improvement over alternative solutions. For most resource management problems, there exist simple heuristics or rules of thumb to guide the decision-making process. Also, for small problems, optimal solutions are readily available. Therefore, it is important to benchmark ADP against other methods.

3. What characterizes a problem that ADP cannot solve? Every method has got its boundaries, and ADP is certainly no exception. To answer this question, we need to find a problem that ADP cannot solve and compare it with the class of problems that ADP can solve. Although we have to be careful to infer to the general case, identifying a problem that at present is too difficult to solve is important for future developments in the field.

1.2 Structure of the Thesis

The thesis is organized in four chapters, of which Chapters 3 to 5 present applications of ADP to solve complex resource management problems along with numerical results.

Chapter 2 provides a short introduction to Markov decision processes as well as a brief overview of various ADP techniques. The chapter covers the algorithmic concepts necessary to understand the idea behind ADP and discusses different types of function approximation as well as learning strategies.

Chapter 3 deals with the problem of finding an optimal bidding strategy for a power generating company that operates system of renewables and a single energy storage. The problem is motivated by the large-scale development of renewable power generation combined with the opportunity to trade on wholesale electricity markets, which has created new challenges for resource management in the energy sector. The problem is modeled as a continuous-state Markov decision process and solved using an ADP algorithm that uses least squares to approximate the value function. The results of this chapter have been published in Energy Systems (Loehndorf and Minner, 2010).

Chapter 4 complements the previous chapter. Although the solution method proposed in Chapter 3 shows promising results, the model formulation lacks details of the actual decision process found in practice. Chapter 4 presents a more detailed model formulation which decomposes the decision problem into a short-term bidding problem and a long-term problem of managing the storage content of multiple hydro reservoirs over time. The short-term problem is formulated as a stochastic program and involves day-ahead bidding decisions under price uncertainty. The long-term problem is modeled as a Markov decision process, whereby the continuous state space is approximated by a set of discrete states which serve as explanatory variables of an econometric electricity price model. The problem is solved using an ADP algorithm that constructs a polyhedral approximation of the value function of the dynamic program. The proposed ADP algorithm is provably convergent if the optimization problem is convex. Numerical results for an industry-scale problem indicate that the algorithm is capable of solving the problem efficiently. The chapter is based on a working paper (Loehndorf et al., 2011).

Chapter 5 covers a stochastic-dynamic decision problem that frequently arises in production systems. In production planning, lot-sizing and scheduling decisions are often treated as deterministic optimization problems. Although this assumption is reasonable in some production environments, there are many applications where demand uncertainty requires integrating lot-sizing and scheduling with safety stock planning. The chapter studies simulation optimization methods to optimize the stochastic economic lot-scheduling problem

(SELSP) formulated as a semi-Markov decision process. Based on a large-scale numerical study, ADP is compared with a global search for parameters of simple control policies. Although ADP worked well for small problems, it was clearly outperformed by the global policy search as soon as the complexity of the SELSP increased. The chapter is based on a working paper (Loehndorf and Minner, 2011).

Chapter 2

Methodological Background

Although each chapter is self-contained in terms of terminology and notation, this chapter provides a brief overview about the fundamentals relevant for understanding the models and methods presented in the remainder of this book.

2.1 Markov Decision Processes

In our treatment of stochastic-dynamic decision problems, we focus on a specific class of models referred to as Markov decision processes. Following Puterman (2005), a decision problem can be modeled as a Markov decision process (MDP) if the problem fits the following description. The decision-maker takes *actions* at specific points in time called *decision epochs*, and there exist distinct *states* of the environment which provide the decision-maker with all relevant information to assess the outcome of an action. During a decision epoch, the decision-maker observes the current state and chooses an action which incurs an immediate reward. Then, the environment evolves into a new state according to a probability distribution that only depends on the current state as well as the action chosen. The decision-maker observes the new state during the next decision epoch, where faced with a similar decision problem. The rules that guide the decision-maker's actions are provided by a *policy.* During each decision epoch, the policy tells the decision-maker which actions to take in a given state. As the decision process evolves over time, following a particular policy produces a sequence of rewards. The stochastic-dynamic decision problem is to select a policy such that this sequence of rewards is optimized. Many stochastic-dynamic decision problems that arise in the real world fit this description, and unless the evolution of the stochastic process depends on the complete sequence of previous states and actions, most of these problems can be modeled as MDPs.

2.1.1 Terminology and Notation

Now that we have a basic idea of the type of decision problem covered in this book, it is useful to introduce some terminology and notation.

Let us begin with the definition of decision epochs. In *discrete-time* problems, the timing of a decision epoch t is defined by periods or stages and a decision is made at the beginning of each period. If the set of decision epochs $\{1, \ldots, T\}$ is finite, in which case $T < \infty$, we refer to the decision problem as a *finite horizon* problem. Otherwise, we refer to it as an *infinite horizon* problem. In *continuous-time* problems, on the other side, decision epochs occur at random points in time or after discrete events, in which case the timing of decision epochs is defined on the interval $[0, T]$ for finite horizon problems or $[0, \infty)$ for infinite horizon problems.

At the beginning of each decision epoch, the environment is in a certain state S_t which is an element of a set of states called the *state space* $\mathcal{S}$. The decision-maker observes this state and takes an action x_t from a set of state-dependent, feasible actions $\mathcal{X}(S_t)$, also referred to as the *action space*. If the decision-maker uses a specific rule during each decision epoch, we refer to this rule as the policy π which specifies an action x_t given a state S_t. Note that a general dependence of $\mathcal{S}$ on t is unnecessary if we define $\mathcal{S} = \bigcup_{t=1}^{T} \mathcal{S}_t$.

At the end of each decision epoch, the environment evolves from the current state S_t into a new state S_{t+1} with probability $\mathbb{P}(S_{t+1}|S_t, x_t)$. The function $\mathbb{P}(S_{t+1}|S_t, x_t)$ is referred to as the *probability transition function* or *probability transition matrix* if we assume that state and action space are both discrete and finite. Since the transition probability to the next state only depends on the current state and action, the stochastic process is called a Markov process.

As a result of action $x_t \in \mathcal{X}(S_t)$, the decision-maker receives a finite, real-valued, reward $r_t(S_t, x_t)$. The reward is accumulated during the decision epoch and its (expected) value is known before an action is taken. The reward may reflect the (expected) profit, cost, or any real-valued performance measure. Accordingly, the result of implementing a policy is that the decision-maker receives a sequence of rewards. To evaluate the policy of a finite horizon problem, it is often sufficient to compute the (expected) *total reward*. In infinite horizon problems, however, the expected total reward may be infinite, so that either the long-term *average reward* is used or the *discounted reward* which requires specification of a discount factor γ.

In contrast to discrete-time problems, where decision epochs have unit length, the length of a decision epoch in continuous-time problems is given by the *sojourn time* $\tau_t(S_t, x_t, S_{t+1})$ which depends on the current state and action as well as the subsequent state. In a more general continuous-time model, called *semi-Markov decision process* (SMDP), the state may change multiple times during a decision epoch and the length of an epoch follows a general probability distribution. To streamline the presentation and avoid redundancy, the remainder of this chapter deals with discrete-time problems. The interested reader is referred to Chapter 5 for a stochastic-dynamic decision problem formulated as an SMDP.

```
(1) Input arguments: final value function $V_T$
(2) Do for $t = T-1, T-2, \dots, 1$
    (2.1) Do for all $S_t \in \mathcal{S}$
        (2.1.1) Solve $\pi(S_t) = \arg\max_{x_t \in \mathcal{X}(S_t)} \Big\{ r_t(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t) V_{t+1}(S_{t+1}) \Big\}$
        (2.1.2) Compute $V_t(S_t) = r_t\big(S_t, \pi(S_t)\big) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}\big(S_{t+1}|S_t, \pi(S_t)\big) V_{t+1}(S_{t+1})$
(3) Return value function $V_t,\ t = 1, \dots, T-1$
```

Figure 2.1: Backward dynamic programming for finite horizon problems

2.1.2 Finite Horizon Problems

In general, a Markov decision process can be formulated as a recursive function that relates the value of being in a state at the beginning of a period to the value of the states that are (possibly) encountered during subsequent periods. This recursive function is known as the *value function* V_t of the dynamic program. Assuming that the decision-maker's objective is to maximize the discounted reward over a finite planning horizon and that V_T is given, a policy $\pi = \{x_1^*, \dots, x_{T-1}^*\}$ is optimal if it holds that

$$V_t(S_t) = \max_{x_t \in \mathcal{X}(S_t)} \Big\{ r_t(S_t, x_t) + \gamma \mathbb{E}\big[V_{t+1}(S_{t+1})|S_t, x_t\big] \Big\} \quad \forall\ S_t \in \mathcal{S},\ 1 \le t \le T-1. \tag{2.1}$$

Note that the discount factor γ can be dropped if the optimality criterion is total reward.

Equation (2.1) is the *expectation form* of the optimality equation, also known as *Bellman equation.* If we assume that state and action space are discrete, we can express the expectation explicitly, which gives us the *standard form* of the optimality equation,

$$V_t(S_t) = \max_{x_t \in \mathcal{X}(S_t)} \left\{ r_t(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t) V_{t+1}(S_{t+1}) \right\} \quad \forall\ S_t \in \mathcal{S},\ 1 \le t \le T-1. \tag{2.2}$$

Let us assume for now that $\mathcal{S}$ and $\mathcal{X}$ are discrete, so that we use the standard form of the optimality equation unless otherwise noted.

When we solve a finite horizon problem, we assume that there exists a function V_T to evaluate the final state of the system, also known as the *salvage value.* To find an optimal policy, we start at the last decision epoch $T-1$ using V_T to evaluate S_T, compute the optimal decision for each state S_{T-1} and then use the associated optimal values in $T-2$. An outline of the algorithm known as *backward dynamic programming* is shown in Figure 2.1.

In case V_T is not given, we can simply choose T larger than the planning horizon and then set $V_T \equiv 0$. If T is sufficiently large and covers more periods than necessary, we can assume that the quality of the policy during the relevant decision epochs is sufficiently good. Otherwise, we may be better off by modeling the problem as an infinite horizon problem.

2.1.3 Infinite Horizon Problems

When we solve an infinite horizon problem, we assume that parameters of the reward function, the transition function, and the stochastic process are stationary over time. For this reason, in many formulations of infinite horizon problems, the time index is dropped, and S_{t+1} is replaced by S' to denote the successor state. The *steady state* optimality equations are then given by

$$V(S) = \max_{x \in \mathcal{X}(S)} \left\{ r(S,x) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}(S'|S,x) V(S') \right\} \quad \forall\ S \in \mathcal{S}. \tag{2.3}$$

Note that if time is relevant for the state transition, e.g., to model a weekly planning cycle, it can be included as an additional state variable.

In contrast to finite horizon problems, we cannot simply drop the discount factor if we are interested in maximizing the total expected reward of an infinite horizon problem. Instead, we have to reformulate the problem using the average reward criterion. The corresponding optimality equations are given by

$$V(S) = \max_{x \in \mathcal{X}(S)} \left\{ r(S,x) - g^\pi + \sum_{S' \in \mathcal{S}} \mathbb{P}(S'|S,x) V(S') \right\} \quad \forall\ S \in \mathcal{S}, \tag{2.4}$$

with g^π as the average reward or *gain* under policy π,

$$g^\pi = \lim_{T \to \infty} \mathbb{E}\left[\sum_{t=1}^{T} \frac{r_t(S_t, x_t)}{T} \right]. \tag{2.5}$$

For operational decision problems, the average reward criterion appears to be the most natural choice, since economic discounting rarely plays a role if decision epochs are hours or days. However, algorithms for solving average reward MDPs are unstable for some problems (Gosavi, 2009), so that we focus on the discounted reward criterion as the default unless otherwise noted.

The most widely used algorithm to solve infinite horizon problems is *value iteration* which converges towards the actual value function by iterative improvement. An outline of the algorithm is shown in Figure 2.2. The algorithm begins with an initial estimate of the value function, typically $V^0 \equiv 0$. Then, for each iteration n, the algorithm computes an approximation of the value function V^n by using the approximation of the value function from the previous iteration V^{n-1} to solve the optimality equation. The algorithm stops when an ε-optimal policy has been found, i.e., when the maximum difference between the optimal value $V^*(S)$ and the infinite

```
(1) Input arguments: initial value function $V^0 \equiv 0$
(2) Repeat
    (2.1) Increment $n \leftarrow n+1$
    (2.2) Do for all $S \in \mathcal{S}$
        (2.1.1) Solve $\pi^n(S) = \arg\max_{x \in \mathcal{X}(S)} \Big\{ r(S,x) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}(S'|S,x) V^{n-1}(S') \Big\}$
        (2.1.1) Compute $V^n(S) = r\big(S, \pi^n(S)\big) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}\big(S'|S, \pi^n(S)\big) V^{n-1}(S')$
    Until $\|V^n(S) - V^{n-1}(S)\|_\infty < \varepsilon(1-\gamma)/2\gamma$
(3) Return value function $V^n$
```

Figure 2.2: Value iteration for infinite horizon problems

horizon value from following the current policy $V^\pi(S)$ is less than ε,

$$\|V^\pi(S) - V^*(S)\|_\infty < \varepsilon, \tag{2.6}$$

where $\|\cdot\|_\infty$ is defined as the *maximum norm*. Since we can neither measure the optimal value nor the infinite horizon value from following policy π^n, the algorithm measures the maximum difference between the value of the previous iteration $V^{n-1}(S)$ and the current iteration $V^n(S)$ instead. If we change the stopping criterion to

$$\|V^n(S) - V^{n-1}(S)\|_\infty < \varepsilon(1-\gamma)/2\gamma, \tag{2.7}$$

then it can be shown that the algorithm converges to an ε-optimal policy (Puterman, 2005, Theorem 6.3.1).

An alternative approach to solve infinite horizon problems is by formulating the problem as a linear program. Denote $V(S) \in \mathbb{R}$ as the unconstrained decision variables of the primal and $\lambda(S,x) \in \mathbb{R}_+$ as the decision variables of the dual formulation. Then, we can find the optimal value function by solving the following linear program

$$\min_{V(S) \in \mathbb{R}} \sum_{S \in \mathcal{S}} V(S) \tag{2.8}$$

$$s.t.\ V(S) \geq r(S,x) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}\big(S'|S,x\big) V(S') \quad \forall\ S \in \mathcal{S}, x \in \mathcal{X}. \tag{2.9}$$

An optimal policy is found by choosing action x in state S where constraint (2.9) is binding, so that

$$\pi(S) = \underset{x \in \mathcal{X}}{\arg\max}\{\lambda^*(S,x)\} \quad \forall\, S \in \mathcal{S}. \tag{2.10}$$

Note that in case no constraint is binding, an arbitrary decision is chosen (Puterman, 2005, Theorem 6.9.4).

Another algorithm for solving MDPs is *policy iteration.* As it also requires performing matrix operations, its computational effort is comparable to that of linear programming. However, modern solvers can perform matrix operations much more efficiently than any ordinary implementation of policy iteration, so that this algorithm will not be addressed here.

An advantage of using value iteration over linear programming is that we do not have to store a complete matrix in memory since rewards and transition probabilities can be computed when needed. Moreover, looping over all states as part of the inner loop can be easily distributed over multiple processors. However, if the problem fits into memory, solving the problem as a single linear program may be faster. Modern linear programming solvers are designed to exploit sparse matrices and can handle problems with millions of constraints.

2.1.4 The Curse of Dimensionality

In most MDPs, a state is typically defined as a vector of state variables $S_t = (S_{t1}, \ldots, S_{tI})$, where each state variable S_{ti} is itself defined as an element of a set of discrete values $S_{ti} \in \{S_{ti1}, \ldots, S_{tiJ}\}$. As a result, the size of the state space increases exponentially in the number of state variables, $|\mathcal{S}| = TJ^I$. Consider that each state variable represents a resource state, e.g., inventory for a specific product, as in Chapter 5. Then, the problem size grows exponentially in the number of resources that ought to be managed over time. In the inventory example, the state space for a problem with 10 products and at most 100 items on stock has already 100^{10}=10^{20}=100,000,000,000,000,000,000 states. Although advances in computing technology continuously shift the boundary for solving ever larger and more complex problems, any of the previously discussed algorithms eventually falls prey to this so-called *curse of dimensionality.*

2.2 Approximate Dynamic Programming

As a methodology to break the curse of dimensionality, approximate dynamic programming (ADP) has recently received increasing attention. To avoid a complete enumeration of the state space, an ADP algorithm uses Monte Carlo simulation to sample the Markov decision process and then approximates the original value function by a function of much lower complexity. When developing an ADP algorithm, there are three problem-specific issues that have to be addressed. First, a policy that uses an approximate value function has to be able to mimic the behaviour of the optimal policy. Second, the sampling process has to be designed such that those states are sampled sufficiently often which can be reached by the optimal policy. Third, the mechanism that updates the value function approximation has to account for noise in the sampled observations.

This section only provides a brief overview of approximate dynamic programming and introduces its key concepts. Comprehensive text books on the subject are Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998), and Powell (2007), where most of the ADP techniques presented next are discussed in more detail.

2.2.1 Finite Horizon Problems

In the previous section, we assumed that we had access to an explicit model of the state transition process which was given by the probability transition matrix. This model allowed us to compute the expected value over all possible successor states for a given state and action. By contrast, in approximate dynamic programming, we do not require an explicit model but merely assume that we have access to a simulation model of the state transition process.

Since we want to solve the problem without using a transition matrix, let us return to the expectation form of the optimality equations

$$V_t(S_t) = \max_{x_t \in \mathcal{X}(S_t)} \big\{ r_t(S_t, x_t) + \gamma \mathbb{E}\big[V_{t+1}(S_{t+1}) | S_t, x_t\big] \big\} \quad \forall\, S_t \in \mathcal{S},\ 1 \leq t \leq T-1. \tag{2.11}$$

Although we cannot compute the expectation explicitly without a model of the state transition process, we can compute a sample average using Monte Carlo simulation. A straightforward solution based on simulation is to replace the value function by a function of an estimate of the state-action value, $Q_t(S_t, x_t)$, also known as *Q-factors*. Again, let us assume that $Q_T(S_T, x_T)$ is given. To compute the Q-factors, we simulate N state transitions for each $Q_t(S_t, x_t)$ and solve the following equations using backward recursion,

$$Q_t(S_t, x_t) = r_t(S_t, x_t) + \frac{\gamma}{N} \sum_{n=1}^{N} \max_{x_{t+1}^n \in \mathcal{X}(S_{t+1}^n)} \big\{ Q_t(S_{t+1}^n, x_{t+1}^n) \big\} \quad \forall\, S_t \in \mathcal{S}, x_t \in \mathcal{X}(S_t), \tag{2.12}$$

from $t = T-1, \ldots, 1$. Of course, this approach only works when state and action space are reasonably small. In high-dimensional state and action spaces, however, this approach is itself subject to the curse of dimensionality.

To avoid the problem of looping over the entire action space, we are going to express the value function in terms of the *post-decision state* (Powell, 2007, p. 101), also known as the *afterstate* value function (Sutton and Barto, 1998, p. 156). In contrast to Q-factors which return the value before an action is taken, the post-decision value function returns the value at the end of a decision epoch. Let us express the expectation of the value right after a decision has been made by $\bar{V}_t(S_t, x_t)$, so that Equation (2.11) becomes

$$V_t(S_t) = \max_{x_t \in \mathcal{X}(S_t)} \left\{ r_t(S_t, x_t) + \gamma \bar{V}_t(S_t, x_t)] \right\} \quad \forall\ S_t \in \mathcal{S}. \tag{2.13}$$

Note that the expectation is hidden inside the function $\bar{V}_t$, which turns the problem into a deterministic optimization problem. The new formulation now allows us to rewrite the optimality equation in terms of the post-decision value function

$$\bar{V}_t(S_{t-1}, x_{t-1}) = \mathbb{E}\left[\max_{x_t \in \mathcal{X}(S_t)} \left\{ r_t(S_t, x_t) + \gamma \bar{V}_t(S_t, x_t) \right\} \middle| S_{t-1}, x_{t-1} \right] \quad \forall\ S_t \in \mathcal{S}, x_t \in \mathcal{X}(S_t). \tag{2.14}$$

Such a formulation is clearly advantageous in decision problems with large action spaces, where only a subset of actions is relevant for the state transition. We make extensive use of the post-decision value function in Chapter 4, where the maximization problem requires solving a large mathematical program.

Up to now, we have assumed that we compute a separate value for each state-action pair. However, this strategy does not yet solve the dimensionality problem. A strength of approximate dynamic programming is to solve the optimization problem by approximating the (post-decision) value function by a function of much lower complexity. This function, called the *approximate value function*, takes advantage of the fact that in most problems knowing something about one state may tell us something about another.

Assume that we have an approximate value function $\bar{V}(\,\cdot\,; w_t)$ dependent on a parameter vector $w_t \in \mathbb{R}^K$. The objective of an approximate dynamic programming algorithm is to find a w_t that minimizes the approximation error,

$$\min_{w_t} \left\{ \|\bar{V}_t(S_t, x_t; w_t) - \bar{V}_t(S_t, x_t)\| \right\}, \quad \forall\ 1 \le t \le T-1. \tag{2.15}$$

The resulting function is then an approximation of the expectation given in (2.14), i.e.,

$$\bar{V}_t(S_t, x_t; w_t) \approx \bar{V}_t(S_t, x_t), \quad \forall\ 1 \le t \le T-1. \tag{2.16}$$

Since both functions are defined recursively, finding the optimal parameter vector requires solving the recursion and constructing an approximation simultaneously. Let us introduce two algorithms that are capable of solving this problem efficiently.

Approximate Value Iteration

Due to its simplicity, the most widely used ADP algorithm is *approximate value iteration.* An outline of the algorithm is shown in Figure 2.3. The algorithm is initialized with an approximate value function $\bar{V}_t(\,\cdot\,; w_t^0)$ and a set of (possibly identical) initial states S_1^n. Then, over N

```
(1) Input arguments: approximate value function $\bar{V}_t(\,\cdot\,; w_t^0),\ t = 1, \ldots, T$; initial states $(S_1^n)_{n=1}^N$
(2) Do for $n = 1, 2, \ldots, N$
    (2.1) Do for $t = 1, 2, \ldots, T-1$
        (2.1.1) Solve $x_t^n = \arg\max_{x \in \mathcal{X}(S_t^n)} \Big\{ r_t(S_t^n, x) + \gamma \bar{V}_t(S_t^n, x; w_t^{n-1}) \Big\}$
        (2.1.2) Simulate $S_{t+1}^n = S^M(S_t^n, x_t^n)$
    (2.2) Solve $v_T^n = \max_{x \in \mathcal{X}(S_T^n)} \Big\{ r_T(S_T^n, x) + \gamma \bar{V}_T(S_T^n, x; w_T) \Big\}$
    (2.3) Do for $t = T-1, T-2, \ldots, 2$
        (2.3.1) Compute $v_t^n = r_t(S_t^n, x_t^n) + \gamma v_{t+1}^n$
        (2.3.2) Update $w_{t-1}^n = U^V(\bar{V}_{t-1}, w_{t-1}^{n-1}, S_{t-1}^n, x_{t-1}^n, v_t^n)$
(3) Return approximate value function $\bar{V}_t(\,\cdot\,; w_t^N),\ t = 1, \ldots, T-1$
```

Figure 2.3: Approximate value iteration for finite horizon problems

iterations, it alternates between a forward pass (Step 2.1) and a backward pass (Steps 2.2-2.3). At each iteration of the forward pass, the algorithm chooses the action that maximizes the sum of the immediate reward and the current estimate of the discounted post-decision value (Step 2.1.1). The algorithm then calls the simulation model S^M which generates the next state given the current state and action (Step 2.2.2). At each iteration of the backward pass, the algorithm first recursively computes the discounted value of the state-action pair sampled during the forward pass (Step 2.2.1),

$$v_t^n = r_t(S_t^n, x_t^n) + \gamma v_{t+1}^n. \tag{2.17}$$

Then, it passes this value to the updating function U^V (see Section 2.2.3) to update the estimate of the approximate value function of the previous state-action pair (Step 2.2.2). In Step 3, the algorithm returns the final approximation of the post-decision value function.

Note that many implementations of approximate value iteration update the value function during the forward pass. While this is inevitable in infinite horizon problems, as we will see shortly, in finite horizon problems, it would take T iterations of the outer loop to pass information from the final stage to the first stage, which slows down convergence of the algorithm.

In Chapter 4, a variant of approximate value iteration is used to solve a multi-stage stochastic programming problem.

Approximate Policy Iteration

Another popular ADP algorithm is *approximate policy iteration.* The label *policy iteration* is somewhat misleading, since we are still updating the value function. In the reinforcement

```
(1) Input arguments: approximate value function $\bar{V}_t( \cdot ; w_t^0)$, $t = 1, \dots, T$; initial states $(S_1^n)_{n=1}^N$
(2) Do for $m = 1, 2, \dots, M$
    (2.1) Do for $n = 1, 2, \dots, N$
        (2.1.1) Do for $t = 1, 2, \dots, T-1$
            (2.1.1.1) Solve $x_t^{n,m} = \arg\max_{x \in \mathcal{X}(S_t^{n,m})} \Big\{ r_t(S_t^{n,m}, x) + \gamma \bar{V}_t(S_t^{n,m}, x; w_t^{m-1}) \Big\}$
            (2.1.1.1) Simulate $S_{t+1}^{n,m} = S^M(S_t^{n,m}, x_t^{n,m})$
        (2.1.2) Solve $v_T^{n,m} = \max_{x \in \mathcal{X}(S_T^{n,m})} \Big\{ r_T(S_T^{n,m}, x) + \gamma \bar{V}_T(S_T^{n,m}, x; w_T) \Big\}$
        (2.1.3) Do for $t = T-2, T-2, \dots, 2$
            (2.1.3.1) Compute $v_t^{n,m} = r_t(S_t^{n,m}, x_t^{n,m}) + \gamma v_{t+1}^{n,m}$
    (2.4) Update $w_t^m = U^P\big(\bar{V}_t, w_t^{m-1}, (S_t^{n,m}, x_t^{n,m}, v_t^{n,m})_{n=1}^N\big)$, $t = 1, \dots, T-1$
(3) Return approximate value function $\bar{V}_t( \cdot ; w_t^M)$, $t = 1, \dots, T-1$
```

Figure 2.4: Approximate policy iteration for finite horizon problems

learning community, this class of algorithms is therefore also referred to as *batch methods.* The idea is to store a batch of observations of (S_t, x_t, v_{t+1}) tuples and update the value function after a complete batch has been collected. An outline of the algorithm is shown in Figure 2.4.

What immediately stands out is that approximate policy iteration has an additional outer loop. During each iterations of this outer loop, the algorithm performs N forward and backward passes as in approximate value iteration. However, instead of updating the approximate value function during the backward pass, the algorithm stores the entire information about state, action, and value collected during all N sample paths. After N iterations of the second loop, the algorithm passes the collected information to the updating function U^P to obtain a new estimate of the approximate value function.

An obvious disadvantage of approximate policy iteration is that it only updates the value function every N iterations. However, if updating the value function is computationally expensive while generating new samples is not, then approximate policy iteration may be the better choice.

2.2.2 Infinite Horizon Problems

In line with our presentation of infinite horizon MDPs, we drop the time index and replace S_{t+1} with S'. The steady state version of the post-decision value function is then given by

$$\bar{V}(S, x) = \mathbb{E}\left[\max_{x' \in \mathcal{X}(S')} \big\{ r(S', x') + \gamma \bar{V}(S', x') \big\} \middle| S, x \right] \quad \forall \quad S \in \mathcal{S}. \tag{2.18}$$

```
(1) Input arguments: approximate value function $\bar{V}(\,\cdot\,; w^0)$, initial state $S^0, x^0$
(2) Do for $n = 1, 2, \ldots, N$
        (2.1)   Simulate $S' \leftarrow S^M(S, x)$
        (2.2)   Solve $x' \leftarrow \arg\max_{x' \in \mathcal{X}(S')} \left\{ r(S', x') + \gamma \bar{V}(S', x'; w) \right\}$
        (2.3)   Compute $v \leftarrow r(S', x') + \gamma \bar{V}(S', x'; w)$
        (2.4)   Update $w \leftarrow U^V(\bar{V}, w, S, x, v),\ S \leftarrow S',\ x \leftarrow x'$
(3) Return approximate value function $\bar{V}(\,\cdot\,; w)$
```

Figure 2.5: Approximate value iteration for infinite horizon problems

The algorithmic strategies used for solving infinite horizon problems are essentially the same as for finite horizon problems, but there are a few subtleties.

Approximate Value Iteration

In contrast to the finite horizon case, approximate value iteration for infinite horizon problems updates the approximate value function during one single forward pass. An outline of the algorithm is shown in Figure 2.5.

After observing a state transition from S to S' (2.1), the algorithm chooses an action x' that maximizes the expected discounted reward based on its current estimate about the post-decision value (2.2). Then, the algorithm computes the value estimate of the new state and action (2.3), updates the parameters of the approximate value function with the new information, and sets $S \leftarrow S'$ and $x \leftarrow x'$ (2.4). When the iteration counter reaches N, the algorithm returns the approximate value function (3). This algorithm has been first proposed in Rummery and Niranjan (1994) and is often referred to as *SARSA*, as it requires the current **S**tate and **A**ction, the **R**eward, and the next **S**tate and **A**ction for an update (Sutton and Barto, 1998, p. 146).

Of course, one could also include a backward pass. This immediately makes sense if we assume that all policies have an *absorbing state*, e.g., the decision-maker sells all of his assets. However, a difficulty with infinite horizon problems is that we may update the same state multiple times during a backward pass which distorts its value estimate. Nevertheless, there exists a variant of approximate value iteration which passes information backwards through time. The idea is to update the value estimates of a number of previous states but to discount the weight of the update depending on the temporal difference between the updated state and the current state. This variant of approximate value iteration was first proposed by Sutton (1988) and is known as *temporal difference* (TD) learning. It largely depends on the problem whether adding a backward pass really makes the difference (Sutton and Barto, 1998, Ch. 7).

```
(1) Input arguments: approximate value function $\bar{V}( \cdot \,; w^0)$, initial state $S^0, x^0$
(2) Do for $m = 1, 2, \ldots, M$
    (2.1) Do for $n = 1, 2, \ldots, N$
        (2.1.1) Simulate $S^n \leftarrow S^M(S^{n-1}, x^{n-1})$
        (2.1.2) Solve $x^n \leftarrow \arg\max_{x \in \mathcal{X}(S^n)} \Big\{ r(S^n, x) + \gamma \bar{V}(S^n, x; w^{m-1}) \Big\}$
        (2.1.3) Compute $v^n \leftarrow r(S^n_t, x^n) + \gamma \bar{V}(S^n, x^n; w^{m-1})$
    (2.2) Update $w^m = U^P\big(\bar{V}, w^{m-1}, (S^n, x^n, v^n)_{n=1}^N\big)$
    (2.3) Set $S^0 \leftarrow S^N,\ x^0 \leftarrow x^N$
(3) Return approximate value function $\bar{V}( \cdot \,; w^M)$
```

Figure 2.6: Approximate policy iteration for infinite horizon problems

An application of approximate value iteration to approximate the value function of an infinite horizon semi-Markov decision process can be found in Chapter 5.

Approximate Policy Iteration

Just like approximate value iteration, the inner loop of approximate policy iteration for infinite horizon problems has only a single forward pass, during which the algorithm generates a sample path of state-action-value tuples. An outline of the algorithm is shown in Figure 2.6. As in the finite horizon case, the algorithm updates the value function after a batch of N tuples has been collected. Since there exists no fix initial state in infinite horizon problems, the last state of the previous pass is used as initial state during the next forward pass.

In Chapter 3, a variant of approximate policy iteration is used to approximate the value function of an infinite horizon Markov decision process which is defined on a continuous state space.

2.2.3 Value Function Approximation

So far, the approximate value function has only been vaguely described by its dependence on a vector of parameters. Let us now assume that we approximate the original value function by a linear combination of K basis functions ϕ_k, where each basis function is multiplied with a real-valued weight w_k, $k \in \{1, 2, ..., K\}$. A basis function ϕ_k can be an arbitrary (possibly non-linear) function of S and x. Denote w and Φ as the corresponding vectors of length K and

$w^\top$ as the transpose of the weight vector. Then, the approximate value function is given by

$$\bar{V}(S,x;w) = w^\top \Phi(S,x) = \sum_{k=1}^{K} w_k \phi_k(S,x). \tag{2.19}$$

To avoid notational clutter, let us drop the possible dependence of $\bar{V}$, S, x, and w on t, since the difference between finite and infinite horizon is irrelevant to function approximation.

Updating Mechanisms

In approximate value iteration, the updating function U^V is typically based on stochastic gradient algorithms (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998; Powell, 2007). A stochastic gradient algorithm adjusts the weight vector w after each observation in the direction that minimizes the *mean squared error*

$$\min_w \frac{1}{2}\big(\bar{V}(S^n, x^n; w) - v^{n+1}\big)^2. \tag{2.20}$$

For a linear function, the *stochastic* sample gradient with respect to w is given by $\Phi(S,x)$. Since the actual gradient is unknown, the stochastic gradient algorithm adjusts the weight vector in the direction of the gradient only by a small amount $\alpha_t \in (0,1]$, referred to as stepsize. The function U^V that updates the weight vector is then given by

$$U^V\big(w^{n-1}, S, x, v\big) = w^{n-1} + \alpha_t\big(v - \bar{V}(S,x;w^{n-1})\big)\Phi(S,x). \tag{2.21}$$

For a stationary policy, the weight vector w is guaranteed to converge to a local optimum as long as we gradually reduce the stepsize to zero (Bertsekas, 2007, p. 333). Practical convergence, however, is largely affected by choosing the right stepsize for each updating step. Although the optimal stepsize schedule is unknown, experimental work has shown that there exist simple stepsize rules that work well in practice. A rule that is simple yet effective is the *generalized harmonic* stepsize, which is given by

$$\alpha^n = \frac{a}{a+n-1}\, b, \tag{2.22}$$

with $a \in \mathbb{R}^+$ and $b \in (0,1]$ as scaling parameters. See (Powell, 2007, Ch. 6), for a review of deterministic as well as stochastic stepsize rules. Ryzhov et al. (2011) propose an *optimal* stepsize rule which chooses the stepsize automatically such that the mean squared error after each new observation is minimized. In this work, we are going to assume that (2.22) is used unless noted otherwise.

An alternative to using stochastic gradients and perform an update after each transition is to collect a batch of observations and then using (non-) linear regression to update the weight vector. We apply this strategy in approximate policy iteration, where the updating function U^P receives an entire batch of observations. Batch-wise updating is particularly useful when we are fitting continuous functions, e.g., polynomials or radial basis functions (see Section 2.2.3).

The most commonly used method to estimate the weights of a linear value function for a given set of observations is least squares. We can use least squares to estimate the weight vector w if we assume that all basis functions in $\Phi(S,x)$ are linearly independent. The weight vector is then computed by solving

$$\begin{aligned} &U^P\left(\bar{V}, w^{m-1}, (S^n, x^n, v^n)_{n=1}^N\right) \\ &\qquad = (1-\alpha^m)w^{m-1} + \alpha^m \left(\sum_{n=1}^{N-1} \Phi(S^n,x^n)\Phi^\top(S^n,x^n)\right)^{-1} \left(\sum_{n=1}^{N-1} \Phi(S^n,x^n)v^{n+1}\right). \end{aligned} \tag{2.23}$$

An advantage of using least squares over stochastic gradients is that it converges with fewer training samples and is not as sensitive to the choice of the stepsize parameter (Boyan, 2002). A major disadvantage, however, is that if we update too frequently efficiency becomes a problem, since (2.23) involves solving a system of linear equations. To combine the updating frequency of approximate value iteration with the efficiency of least squares estimation, we can use *recursive least squares* (RLS) instead. While RLS has only $\mathcal{O}(n^2)$ time complexity, keep in mind that the stochastic gradient algorithm has an even lower $\mathcal{O}(n)$ time complexity and that only coefficients of non-zero basis functions need to be updated. For an analysis of using least squares for approximate policy iteration, see Boyan (2002); Nedic and Bertsekas (2003); Lagoudakis and Parr (2003).

State Aggregation

A special type of a linear function approximation is *state aggregation*, where the state space is partitioned into a manageable number of K disjoint subsets $\mathcal{A}$, i.e.,

$$\mathcal{S} \times \mathcal{X} = \bigcup_{k=1}^{K} \mathcal{A}_k. \tag{2.24}$$

Instead of storing a separate value for each state, we aggregate several states in one set and only store one value for each set. The easiest way to think of state aggregation is as a coarse grid laid over the state space (Sutton and Barto, 1998, p. 204).

To represent state aggregation as a linear combination of basis functions, each basis function can be viewed as an indicator function $\mathbb{1}_{\mathcal{A}_k}$ which is defined as

$$\phi_k(S,x) = \mathbb{1}_{\mathcal{A}_k}(S,x) = \begin{cases} 1 & \text{if } (S,x) \in \mathcal{A}_k, \\ 0 & \text{otherwise} \end{cases} \quad \forall\, k = 1,\ldots,K. \tag{2.25}$$

The indicator function returns 1 when the state-action tuple (S,x) is a member of partition $\mathcal{A}_k$ and 0 otherwise. This concept is related to that of dummy variables often used in regression analysis to represent categorical variables.

An approximate value function that uses K indicator functions as basis functions is given by

$$\bar{V}(S,x;w) = w^\top \Phi(S,x) = \sum_{k=1}^{K} w_k \mathbb{1}_{\mathcal{A}}(S,x). \tag{2.26}$$

If Equation (2.24) holds, then $\bar{V}(S,x;w)$ returns w_k if $(S,x) \in \mathcal{A}_k(S,x)$, in which case the updating step given in (2.21) simplifies to

$$w_k^n = U^V\big(w^{n-1}, S, x, v\big) = w_k^{n-1} + \alpha^n(v - w_k^{n-1}) \tag{2.27}$$

Evidently, updating the weight of a single basis function using stochastic gradients is far more efficient than updating the entire matrix of basis function values as it is done by least squares methods.

Although coarse grids are often used for state aggregation in approximate dynamic programming, they do not solve the dimensionality problem, since the size of the state space still increases exponentially in the number of state variables. A way to deal with this problem is by allowing the subsets to overlap so that each state is mapped to multiple subsets. Instead of updating one subset at a time, several subsets are being updated simultaneously. An efficient data structure that allows subsets to overlap is the CMAC neural network (Glanz et al., 1991), which can be viewed as multiple layers of the same grid shifted against one another. Another way to implement overlapping subsets is by combining multiple levels of aggregation (Powell, 2007, p. 233). An application of state aggregation to approximate the value function of a stochastic production planning problem is given in Chapter 5, where a linear combination of piecewise-constant functions is used to approximate the value function.

Regression Models

A problem of state aggregation is its poor ability to interpolate, so that approximation errors at the boundaries of a subset get large when the value function becomes steeper. Let us briefly introduce two linear architectures which circumvent this problem and are well-suited to approximate continuous value functions.

In empirical research, the influence of a vector of input variables on an output variable is often modeled as a polynomial function. An advantage of using polynomials is that we can approximate high-dimensional continuous value functions through a function which has a relatively low number of terms. For example, with n and m being the dimension of the state and action space, respectively, a second-order polynomial has at most $(n+m-1)(n+m)/2+n+m+1$ terms. Moreover, if we use a second- or third-order polynomial as approximate value function, we can find an optimal action analytically. Using higher-order polynomials, we lose this property and have to resort to numerical techniques.

For example, assume that states and actions are both scalar and that a second-order polynomial is used. Then, the approximate value function is given by

$$\bar{V}(S,x;w) = w_0 + w_1 S + w_2 x + w_3 S^2 + w_4 Sx + w_5 x^2, \tag{2.28}$$

with the optimum either at $x^* = (-w_2 - w_4 S)(2w_5)^{-1}$ if $w_5 \neq 0$ or at the boundaries of $\mathcal{X}$. The derivation for larger numbers of state variables follows along the same lines. See Chapter 3, for an application where second- and third-order polynomials are used for value function approximation.

Finding the global optimum in higher-dimensional action spaces is computationally more complex. Also, overfitting quickly becomes a problem, if the ratio of sample size to number of weights is too small. One way to deal with these problems is by including additional assumptions about the value function. In many resource management problems, for example, the value function is jointly concave in the resource state. We can include this information by solving the least squares problem with the additional constraint that the polynomial is jointly concave in the action variables, which requires solving a semidefinite programming problem (Magnani et al., 2005; Boyd and Vandenberghe, 2004).

Another powerful class of models for value function approximation are radial basis functions (RBFs). Like polynomials, RBFs are able to approximate any real-valued function defined on a compact subspace with arbitrary precision (Park and Sandberg, 1993). An RBF returns a weighting of the distance between a fixed *center point* and a query point or a query state, as in our case. Typical choices of RBFs are Gaussian functions, (inverse) multiquadric functions or thin-plate splines.

For example, assume that the state space is continuous and normalized, so that $\mathcal{S} \in [0,1]^n$, and there exists a finite number of actions, $\mathcal{X} = \{\hat{x}_1, \dots, \hat{x}_m\}$. Also assume that there is a set of Gaussian radial basis function with center points at c_k assigned to each action. Then, each basis function is defined as

$$\phi_k(S,x) = \begin{cases} w_k \exp\left(-\|c_k - S\|_2\right) & \text{if } x = \hat{x}_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.29}$$

As opposed to state aggregation, each basis function is defined over the entire state space, but the weight of the function decreases with increasing Euclidean distance between c_k and S.

A question that arises when using radial basis functions is how to define the center points. If the state space is high-dimensional and the samples only cover a small subset, it makes no sense to scatter the center points randomly over the state space. A simple solution is to first run a clustering algorithm to find good center points and then use least squares to find the optimal weights. Other iterative approaches are in Jung and Polani (2007) or Xu et al. (2007).

A somewhat related approach to function approximation using radial basis functions are tree-based regression techniques (Ernst et al., 2005). In tree-based regression, all observations are stored in a spatial data structure which partitions the state space into several regions, e.g., k-d-trees (Cormen et al., 2009, Ch. 10). To approximate the value function, the tree returns the (weighted) average of all observations which are in a neighborhood around the query point. A radial basis function with its center at the query point can then be used to assign each observation a weight that depends on its distance to the query point. Alternatively, one can fit a local model around the query point using locally weighted regression (Atkeson et al., 1997). This is particularly useful in sparse regions where the sample heterogeneity increases the variance of the predictor.

2.2.4 Exploration vs. Exploitation

In our discussion of approximate dynamic programming algorithms, we have assumed until now that all algorithms apply a *greedy policy* during the forward simulation, i.e., they choose the action that maximizes the sum of the immediate reward and the current estimate of the discounted post-decision value. However, since the current policy determines what we know about the value of an action in a particular state, it could happen that the policy never chooses the optimal action simply because its estimate of the post-decision value is incorrect. Therefore, it may be useful to take actions which are considered sub-optimal by the current value estimate to learn more about what they contribute and into which states they can take us. The question is, do we exploit our knowledge about an action or do we explore some new territory? Let us briefly review heuristic learning strategies which trade off exploration against exploitation.

Pure Exploitation

A pure exploitation strategy is identical to the greedy policy, i.e, an action is chosen by solving

$$x^n = \arg\max_{x \in \mathcal{X}(S^n)} \Big\{ r(S^n, x) + \gamma \bar{V}(S^n, x; w^{n-1}) \Big\}. \tag{2.30}$$

The problem with this strategy is that it may lead to a locally optimal policy, unless we can ensure that our estimate of the post-decision value is always optimistic, e.g., it is an upper bound of the true value. In Chapter 4, we use a pure exploitation strategy, where the approximate value function is a polyhedral approximation of the true value function. In that case, we can show that this strategy finds an optimal policy.

Optimistic Initial Values

A simple solution to circumvent the problem of getting stuck in local optima is to start with optimistic initial values. For example, if the objective is to minimize discounted cost, starting with a default initial estimate of zero will encourage a greedy policy to explore actions which it has not tried yet.

Pure Exploration

A pure exploration strategy would be to choose actions randomly. This guarantees that, in the limit, every state-action pair is going to be sampled infinitely often. Using pure exploration, actions are chosen as follows

$$x = x_i, \ i \sim U^d\big(1, |\mathcal{X}|\big), \tag{2.31}$$

with U^d as the discrete uniform distribution. In very large problems, however, pure exploration eventually falls prey to the curse of dimensionality, because we can only cover a small fraction of the complete state and action space during the sampling process. Nevertheless, pure exploration can be used during the initial phase of information collection when there still exists no or only little information about the value of taking an action in a particular state.

Epsilon-Greedy Exploration

The most basic strategy to combine pure exploitation with pure exploration is to use an epsilon-greedy policy which follows the greedy policy with probability $1 - \varepsilon$ and takes a random action otherwise. Using an epsilon-greedy policy π^E, actions are chosen as follows,

$$x^n = \pi^E(S^n, u, i) = \begin{cases} \underset{x \in \mathcal{X}(S^n)}{\arg\max} \Big\{ r(S^n, x) + \gamma \bar{V}(S^n, x; w^{n-1}) \Big\} & \text{if } u < 1 - \varepsilon, \\ x_i & \text{otherwise,} \end{cases} \tag{2.32}$$

where $u \sim U^c(0, 1)$ and $i \sim U^d\big(1, |\mathcal{X}|\big)$ with U^c being the continuous uniform distribution. In the early iterations, it makes sense to choose ε large and decrease its value as the number of iterations increases. Another option is to decrease ε depending on the number of times a state has been visited. However, similar to pure exploration, this strategy only works as long as the number of actions is small.

Boltzmann Exploration

A problem with pure or epsilon-greedy exploration is that all actions have the same probability of being sampled even if these actions yield very poor rewards. A somewhat more elaborate strategy is to rank all actions by their value estimate and adjust the probabilities accordingly. A popular exploration strategy in this spirit is *Boltzmann exploration*, also referred to as *softmax action selection* (Sutton and Barto, 1998, p. 30).

Let $Q(S,x) = r(S,x) + \gamma V(S,x)$ and denote $N(S)$ as the number of times that state S has been sampled. Using a softmax policy π^S, at iteration n, we would choose action x in state S with probability,

$$P(S,x) = \frac{\exp\big(Q(S,x)/\tau\big)}{\sum_{x\in\mathcal{X}(S)} \exp\big(Q(S,x)/\tau\big)}, \quad \forall\, x \in \mathcal{X}, \tag{2.33}$$

where $\tau = \big(\max_{x\in\mathcal{X}(S)} Q(S,x) - \min_{x\in\mathcal{X}(S)} Q(S,x)\big)/N(S)$ serves as scaling factor, with $\tau \to 0$ in the limit (see Powell, 2007, p. 328). The softmax policy then chooses an action by recursively solving,

$$x^n = \pi^S(S^n, 1, u) = \begin{cases} x_i & \text{if } P^n(S^n, x^n) \leq u, \\ \pi^S(S^n, i+1, u) & \text{otherwise,} \end{cases} \tag{2.34}$$

with $u \sim U^c(0,1)$.

Boltzmann exploration focuses on actions with high value estimates, but still explores actions which yield very poor rewards. The strategy thereby makes more informed decisions than epsilon-greedy exploration.

Interval Estimation

One of the central motives of exploration is to reduce uncertainty in the value estimates. However, neither epsilon-greedy nor Boltzmann exploration take the variance in the estimates into account. Instead these strategies explore the action space by choosing actions randomly, which is like searching for a needle in a haystack in large action spaces. An exploration strategy that circumvents this problem is *interval estimation* which is designed to reduce the uncertainty in the estimates. The basic idea behind interval estimation is to add a bonus to the value estimate that is equal to a certain number of standard deviations above the value estimate.

In (2.14), we defined the post-decision value function as the function of the expected value of the value after a decision has been made but before transition to the next state,

$$\bar{V}(S,x) = \mathbb{E}\left[V(S')|S,x\right]. \tag{2.35}$$

Accordingly, let us define a function of the corresponding variance of the value after a decision has been made, but before transition to the next state,

$$\mathrm{Var}\big[V(S,x)\big] = \mathbb{E}\big[V(S')^2\big|S,x\big] - \mathbb{E}\big[V(S')\big|S,x\big]^2. \tag{2.36}$$

Since the second term of the right-hand side is the squared estimate of the post-decision value, all we need is an estimate of the squared post-decision value to obtain the first term. Let us define an approximate squared value function,

$$\bar{V}^2(S,x;w_\sigma) \approx \mathbb{E}\Big[\big(V(S')\big)^2\Big|S,x\Big], \tag{2.37}$$

with w_σ as another parameter vector. An estimate of the standard deviation that is based on $\bar{V}$ and $\bar{V}^2$ is then given by

$$\bar{\sigma}(S^n,x^n;w_\sigma^{n-1}) = \sqrt{\bar{V}^2(S^n,x;w_\sigma^{n-1}) - \big(\bar{V}(S^n,x;w^{n-1})\big)^2}. \tag{2.38}$$

Together with our point estimate, we can now use the standard deviation to compute an optimistic estimate of the post-decision value, e.g.,

$$\bar{V}(S^n,x^n;w^{n-1}) + 2\bar{\sigma}(S^n,x^n;w_\sigma^{n-1})/\sqrt{N(S,x)},$$

which returns a value two standard deviations above the point estimate. The value converges towards the point estimate as $N(S,x)$ increases, i.e., in the number of times that (S,x) has been sampled. A policy based on interval estimation then chooses the next action by solving

$$x^n = \arg\max_{x\in\mathcal{X}(S^n)} \Big\{r(S^n,x) + \gamma\Big(\bar{V}(S^n,x;w^{n-1}) + 2\bar{\sigma}(S^n,x;w_\sigma^{n-1})/\sqrt{N(S,x)}\Big)\Big\}. \tag{2.39}$$

The weight vectors w and w_σ can be updated simultaneously, so that this type of exploration integrates well with the approximate dynamic programming algorithms described above.

A comparison of different learning strategies as well as a more detailed discussion of the underlying information collection problem can be found in Powell (2007), Ch. 10.

Chapter 3

Optimal Bidding and Storage of Renewable Energies

Many European countries today subsidize investments in renewable energies by guaranteeing a fixed rate for each kilowatt hour of electricity fed into the grid. Since this rate decreases over the years, a producer of renewable power is going to begin selling electricity directly at the market as soon as subsidies yield lower profits than direct trading. This combination of renewable power production and trading has created new investment opportunities but also challenges for power generating companies.

One challenge is that prices in electricity markets are uncertain. A wholesale electricity market typically consists of a day-ahead and a real-time market. At the day-ahead market, producers (e.g., generating companies) place supply bids and consumers (e.g., electricity providers) place demand bids which mature at the following day. After the day-ahead market is closed, the system operator announces a uniform clearing price which depends on the cumulated bids of all market participants. Since market participants do not reveal their bidding decisions, bidding eventually takes place under price uncertainty.

Another challenge is that the bidding volume may not match the physical volume generated by the renewable power source. In that case, the difference is cleared at the real-time market, also known as the balancing market. If the total difference of demand and supply is positive, the system operator activates operational reserves, which increases the real-time price. If the total difference is negative, the system operator deactivates operational reserves, which decreases the real-time price. Some markets, such as NordPool in Scandinavia, even create asymmetric real-time prices, so that a negative imbalance always settles above and a positive imbalance always below the day-ahead price. A renewable power producer therefore has to account for the cost that arises when a bid does not match supply. Moreover, in a market with a significant share of solar or wind power, producers are going to use the same weather forecasts, which leads to a correlation of forecast errors and real-time price deviations. Bidding decisions of renewable power producers therefore take place under price and supply uncertainty.

Most literature on optimal bidding strategies in electricity markets deals with supplier interaction and the commitment of thermal units. See Wen and David (2000) for an overview. Renewable power producers, however, are often too small to exercise market power and influence other player's decisions. Only few authors deal with the optimal bidding strategy of renewable power producers. Bathurst et al. (2002) propose a stochastic model to minimize the expected cost of balancing. The authors assume that real-time prices are known in advance and develop a stochastic supply model from historical data. Matevosyan and Söder (2006) propose a two-stage stochastic program to minimize the cost of balancing of a wind power producer. The authors use price scenarios and model wind power supplies as an autoregressive (ARMA) model.

An even greater challenge arises when a renewable power producer has the additional option to store electricity. Storage can then be used as a hedge against the costs of balancing and allows the producer to respond to market prices. In addition to price and supply uncertainty, an optimal bidding strategy has to account for future price and supply realizations as well as future bidding decisions. The ability of energy storage to take advantage of arbitrage opportunities is addressed in Graves et al. (1999), although not in combination with renewable energies. Optimal bidding strategies for hybrid systems of wind power generation and energy storage are addressed in Bathurst and Strbac (2003), Korpås and Holen (2006) and Brunetto and Tina (2007) for deterministic supply and price paths. Uncertainty is considered in Fleten and Kristoffersen (2007) and García-González et al. (2008) who model optimal bidding strategies as two-stage stochastic programs. However, there still exists a research gap for models that consider both, supply and price uncertainty, as well as making bidding decisions over time.

Note that renewable power sources, such as wind or solar, do not have the natural means to control their energy output, and for them storage capacity requires additional investments. The most common storage scheme is pumped-hydro storage, where water is pumped into an elevated reservoir to consume excess electricity and released to generate electricity when it is needed later on. See Schainker (2004) for an overview of different energy storage technologies.

We propose to model the optimal bidding problem of a renewable power producer with storage as a continuous-state Markov decision process (MDP). A feature of the MDP is that an optimal bidding strategy derived from solving the problem not only considers price and supply uncertainty, but also takes future states and decisions into account. The difficulty with this formulation is that generic solution algorithms for MDPs, such as value or policy iteration, are subject to the *curse of dimensionality.* These algorithm are therefore only capable of computing an optimal bidding strategy for a small number of discrete states and decisions with known transition matrix. However, the state and action space of the bidding problem is continuous and a discretization with increasing resolution is computationally intractable.

To circumvent the curse of dimensionality and solve the problem efficiently, we propose a solution based on *approximate dynamic programming* (ADP). The ADP strategy presented in this chapter goes back to a class of methods known as *temporal difference* (TD) learning (Sutton and Barto, 1998). These methods learn the value function of an MDP by controlling the decision process according to some policy and iteratively updating the value function estimate. We will focus on a variant known as *least squares policy evaluation* (LSPE) (Nedic and Bertsekas, 2003) which approximates the value function by a set of linearly independent basis functions, where the function weights are estimated by least squares methods. A key advantage of using LSPE for our problem is that this method can handle a continuous state space. In line with Lagoudakis and Parr (2003), we combine LSPE with policy iteration to approximate the value function and to find a near-optimal policy, i.e., we iteratively find a near-optimal bidding strategy. For policy evaluation and improvement, our ADP algorithm has access to a simulation model of the stochastic decision process.

As a benchmark, we compute the transition matrix for an equivalent discrete state MDP and use linear programming to determine an optimal solution. We then compare the LP policy against two ADP policies with different value function approximations. Additionally, to study the influence of model parameters on discounted rewards, we analyze the response surface with a regression model.

3.1 Model Formulation

3.1.1 Markov Decision Process

We assume that during each day the renewable power producer places a bid at the day-ahead market before observing the realization of price and supply. If the realized supply is below the volume of a bid, the producer first empties the storage and then purchases the remainder at the real-time market. If the realized supply exceeds the bidding volume, the producer first stores excess supply before using the real-time market to clear the imbalance.

The power producer is assumed to be price-taker and the bidding strategy does not affect the bidding behavior of other market participants – think of a small player such as an individual wind farm operator. Since the marginal value of wind power is zero, we assume that the producer places a fixed-volume bid for any realization of the uncertain price. Price and supply follow an autoregressive stochastic process so that both observations contain information on price and supply of the following day. A bid that does not match supply is automatically balanced by the system operator. We assume that the price of positive reserve is always u times higher and the price of negative reserve always o^{-1} times lower than the day-ahead price. We aggregate reservoir, pump and generator capacity as storage capacity. Charges and discharges are subject to losses; the total loss is referred to as *round-trip efficiency*.

Each period the power producer observes the current market price p as well as renewable power supplies y. The producer furthermore observes the amount of energy available in storage g at the end of period previous period. These three variables constitute a state of the process $S = \{y, p, g\} \in \mathcal{S}$, with $\mathcal{S}$ as the state space. The transition function $P(S'|x, S)$ denotes the probability that the next state will be S' given that the previous state was S and decision x was made. Based on this information, the producer makes a bidding decision $x \in \mathcal{X}$. Then, a random transition to the next state occurs, in which the bid matures, and the producer obtains a reward $r(S, x, S')$ that depends on the bid as well as the transition from S to S'.

The objective of the power producer is to select a policy $\pi(S)$ that assigns each state in S a decision in $\mathcal{X}$ such that the expected discounted cash flow of rewards is maximized. Let us state the objective function as the following infinite horizon, discounted Markov decision process,

$$V(S) = \max_{x \in \mathcal{X}} \left\{ \int_{S' \in \mathcal{S}} P(S'|S, x)\big(r(S, x, S') + \gamma V(S')\big)\ dS' \right\}, \tag{3.1}$$

with V being the value function and γ being the discount factor.

Denote C as storage capacity and c^+ (c^-) as the amount of energy charged (discharged) during a period and η^+ (η^-) as the efficiency of the charging (discharging) process with $0 < \eta^\pm \leq 1$, i.e.,

$$c^+ = \max\Big\{\min\Big\{y' - x, \frac{C - g}{\eta^+}\Big\}, 0\Big\}, \tag{3.2}$$

$$c^- = \max\Big\{\min\Big\{x - y', \eta^- g\Big\}, 0\Big\}. \tag{3.3}$$

The storage state transition from g to g' is deterministic for given realizations of price and supply. In case of a positive imbalance, the final storage level in the next period is C, and in case of a negative imbalance the final storage level is zero. The storage balance equation is given by

$$g' = g + \eta^+ c^+ - \frac{c^-}{\eta^-}. \tag{3.4}$$

Rewards $r(S, x, S')$ are uncertain and depend on the capability to match the bid x with available capacity in the next period. If a bid exceeds renewable power supplies, the storage is discharged until $x - c^- - y' = 0$. Then the producer has to pay $u \cdot p'$ with $u > 1$ to the system operator for each unit of negative imbalance. If a bid is lower than renewable power supplies, the storage is charged until $y' - x - c^- = 0$. Then the producer receives $o \cdot p'$ with $0 \leq o < 1$ from the system operator for each unit of positive imbalance. The reward function is given by

$$r(S,x,S') = \begin{cases} (y'+c^-)p' - up'(x-y'-c^-) & \text{if } x > y', \\ xp' + op'(y'-x-c^+) & \text{otherwise.} \end{cases} \tag{3.5}$$

Note that the limitation of first using storage to clear an imbalance and then the real-time market may not be an optimal ex-post decision for all realizations of price and supply.

3.1.2 Stochastic Processes

In line with the literature, we assume that the stochastic processes of price and supply follow a first-order *autoregressive* process, i.e., an AR(1) process with normally distributed error terms. Moreover, as soon as multiple renewable power producers trade in the same market, their supplies will be positively correlated and the market price will move inversely proportional to the overall renewable supply – an effect that has already been observed with wind power supplies and spot market prices in Germany (Neubarth et al., 2006). We therefore additionally assume that the price is dependent on supply to account for the homogeneous trading patterns of renewable power producers.

We define mean, variance, autocorrelation and correlation of price and supply exogenously. Denote Y as the autoregressive process of supply with mean μ_Y, variance σ_Y^2 and autocorrelation $\theta_Y \in [0,1)$. Since the realized supply y of the previous period partially explains the supply realization y of the current period, it has a direct effect on mean and variance of the overall stochastic process. The autoregressive supply process with given mean and variance is modeled as

$$y' = \theta_Y y + \varepsilon^Y \quad \text{with} \quad \varepsilon^Y \sim N(\mu_Y^\varepsilon, \sigma_Y^\varepsilon), \tag{3.6}$$

$$\mu_Y^\varepsilon = (1-\theta_Y)\mu_Y, \tag{3.7}$$

$$\sigma_Y^\varepsilon = \sqrt{(1-\theta_Y^2)\sigma_Y^2}. \tag{3.8}$$

Equations (3.7) and (3.8) adapt the moments of the error term ε_t^Y, such that the supply process has mean μ_Y and variance σ_Y^2.

Denote P as the stochastic process of the price with mean μ_P, variance σ_P^2, autocorrelation $\theta_P \in [0,1)$ and parameter θ_{PY} to control the dependence of P on Y. In this case, the current price p' is partially explained by the realized price p of the previous period as well as the random change in supplies of the current period. The stochastic model is then given by

$$p' = \theta_P p + \theta_{PY}(y' - \theta_Y y) + \varepsilon^P, \quad \text{with } \varepsilon_t^P \sim N(\mu_P^\varepsilon, \sigma_P^\varepsilon), \tag{3.9}$$

$$\mu_P^\varepsilon = (1-\theta_P)\mu_P - \theta_{PY}(1-\theta_Y)\mu_Y, \tag{3.10}$$

$$\sigma_P^\varepsilon = \sqrt{(1-\theta_P^2)\sigma_P^2 - \theta_{PY}^2(1-\theta_Y^2)\sigma_Y^2}. \tag{3.11}$$

As before, equations (3.10) and (3.11) adapt the moments of the error term ε^P such that the price process has the intended mean and variance. Moreover, the parameter θ_{PY} controls the dependency of prices on supplies. Let us define the correlation of price and supply which is used as an exogenous parameter later on,

$$\rho = \frac{\theta_{PY}(1-\theta_Y^2)}{1-\theta_Y\theta_P}\sqrt{\frac{\sigma_Y^2}{\sigma_P^2}} \quad \text{with} \quad |\rho| \leq \frac{\sqrt{(1-\theta_Y^2)(1-\theta_P^2)}}{1-\theta_Y\theta_P}. \tag{3.12}$$

By rearranging terms, we can compute θ_{PY} as a function of autocorrelation and correlation of Y and P, which allows to model both processes with a given correlation. Note that Equation (3.11) is undefined for some θ_P, θ_Y and θ_{PY}, such that the correlation coefficient ρ is bounded from above and below.

3.2 Solution Methods

We approximate the optimal policy of the proposed Markov decision process by using an algorithm which iteratively combines policy evaluation and policy improvement. The algorithm thereby approximates the value function of a given policy and then uses the learned relationship of state and reward to improve the policy.

To assess the solution quality of this approach, it is desirable to know the optimal policy. If we relax the assumption of a continuous state space and use a discrete state space instead, we can compute the transition matrix and have linear programming compute an optimal policy. We use this discrete policy as a benchmark for the ADP policy.

3.2.1 Computing the Transition Matrix

For a discrete solution of the value function (3.1), we need a discrete representation of the state transition function, which we refer to as transition matrix. Such a discrete state transition is characterized by discrete realizations of the random variables P and Y as well as a discrete storage state transition. As a result, we need a formulation of the conditional probabilities of price and supply defined over a discrete set of state values. To determine the conditional probability of supply $P_Y(y'|y)$, we restate (3.6), such that

$$\varepsilon^Y = y' - \theta_Y y, \tag{3.13}$$

With $\Phi_Y \sim N(\mu^\varepsilon_Y, \sigma^\varepsilon_Y)$ as the probability distribution of the random shock ε^Y, the cumulated probability of y' conditional on y then becomes

$$P_Y(y'|y) = \Phi_Y(y' - \theta_Y y). \tag{3.14}$$

If we define the stochastic supply process over the discrete set $Y^d = \{Y_L, ..., Y_U\}$ for the discrete probabilities P^d_Y it needs to hold that $\sum_{y' \in Y^d} P^d_Y(y'|y) = 1 \; \forall \; y$. To map the continuous distribution Φ_Y to the finite set Y^d, we round all real values to the next integer and cumulate the probability mass of the tails at the upper and lower bounds. Then, the truncated discrete conditional probability of supply is

$$P^d_Y(y'|y) = \begin{cases} \Phi_Y(Y_L - \theta_Y y + 0.5) & \text{if } y' = Y_L, \\ 1 - \Phi_Y(Y_U - \theta_Y y - 0.5) & \text{if } y' = Y_U, \\ \Phi_Y(y' - \theta_Y y + 0.5) - \Phi_Y(y' - \theta_Y y - 0.5) & \text{if } Y_L < y' < Y_U. \end{cases} \tag{3.15}$$

Accordingly, we derive the conditional probability of the price $P_P(p'|p, y', y)$ by mapping the distribution $\Phi_P \sim N(\mu^\varepsilon_P, \sigma^\varepsilon_P)$ to the discrete set $P^d = \{P_L, ..., P_U\}$. We restate (3.9) as before and the truncated discrete conditional probability of the price becomes

$$P^d_P(p'|p, y, y') = \begin{cases} \Phi_P(P_L - \theta_P p - \theta_{PY}(y' - \theta_Y y) + 0.5) & \text{if } p' = P_L, \\ 1 - \Phi_P(P_U - \theta_P p - \theta_{PY}(y' - \theta_Y y) - 0.5) & \text{if } p' = P_U, \\ \Phi_P(p' - \theta_P p - \theta_{PY}(y' - \theta_Y y) + 0.5) & \\ \qquad -\Phi_P(p' - \theta_P p - \theta_{PY}(y' - \theta_Y y) - 0.5) & \text{if } P_L < p' < P_U. \end{cases} \tag{3.16}$$

To complete the discrete transition function, we restate the storage balance equation of (3.4). As it is difficult to model a discrete storage state transition with $\eta < 1.0$ we assume perfect round-trip efficiency so that the storage balance simplifies to

$$g' = \max\{\min\{y' + g - x, C\}, 0\}. \tag{3.17}$$

By this, we avoid additional rounding errors when using the discrete policy as benchmark to control the continuous-state process.

With this discretization of the stochastic process, we can formulate the probability transition matrix $\mathbb{P}(S'|S, x)$ which assigns a probability to each transition from a state S to a subsequent state S' after decision x has been made. The matrix is defined as

$$\mathbb{P}(S'|S, x) = \begin{cases} P^d_Y(y'|y) P^d_P(p'|y, y', p) & \text{if } g' = \max\{\min\{y' + g - x, C\}, 0\}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.18}$$

The transition probabilities are computed by multiplying the conditional probabilities of price and supply subject to a constraint on the storage balance. If (3.17) does not hold, the storage transition is infeasible and its probability set to zero.

3.2.2 Linear Programming Formulation

Denote $V(S)$ as the decision variable of the linear program. With $\mathbb{P}(S'|S,x)$ as the probability of a state transition from state S to state S' after decision x is made and $r(S,x,S')$ as the corresponding reward, we can state the discrete state, infinite horizon Markov decision process as

$$\min_{V(S)\in\mathbb{R}} \sum_{s\in\mathcal{S}}\sum_{x\in\mathcal{X}} V(S) \tag{3.19}$$

$$s.t.\ V(S) \geq \sum_{S'\in\mathcal{S}} \mathbb{P}(S'|S,x)\big(r(S,x,S') + \gamma V(S')\big) \quad \forall \quad s\in\mathcal{S}, x\in\mathcal{X} \tag{3.20}$$

The optimal policy is found by selecting $\pi(S) = x$ where (3.20) is binding. In case no constraint is binding, an arbitrary decision is chosen (Puterman, 2005, p.223).

The tractability of this method is limited by the $|\mathcal{S}|$-dimensional decision vector with $|\mathcal{S}|\times|\mathcal{X}|$ inequality constraints and the necessity of a transition matrix. For the benchmark, we therefore compute the optimal policy only for a small discrete state space.

3.2.3 Least Squares Policy Evaluation

A widely used algorithm in approximate dynamic programming is temporal difference (TD) learning (Sutton and Barto, 1998) which combines Monte Carlo simulation with dynamic programming. Denote $\bar{V}$ as the post-decision value function which returns the expected value after a decision has been made,

$$\bar{V}(S,x) = \int_{S'\in\mathcal{S}} P(S'|S,x)\big(r(S,x,S') + \gamma V(S')\big)\, dS'. \tag{3.21}$$

Suppose that we use a simulation model to sample N state transitions, where each transition gives us a realization of the expectation. The TD learning algorithm updates the value estimate of a given state S and decision x by observing the reward and the discounted value of the subsequent state associated with the simulated state transition. The value estimate after n iterations is given by

$$\bar{V}^{n+1}(S^n,x^n) = \bar{V}^n(S^n,x^n) + \alpha^n\big(r(S^n,x^n,S^{n+1}) + \gamma\bar{V}^n(S^{n+1},x^{n+1}) - \bar{V}^n(S^n,x^n)\big), \tag{3.22}$$

where the parameter $\alpha^n \in (0,1]$ is used to smooth the updates of the estimates.

The updating step in (3.22) only works if we assume that the state space is discrete. Since we are dealing with a continuous state and action space, however, let us resort to another popular approximation architecture. Assume that $\bar{V}(S, x; w)$ is a linear combination of basis functions $\phi_i(S, x)$ with weights w_i and $k \in \{1, 2, ..., K\}$ which approximates the true value function. Denote w and $\Phi(S, x)$ as the corresponding vectors of length K with $w^\top$ and $\Phi^\top$ as their transposes. Then, the approximate value function is defined as

$$\bar{V}(S, x; w) = \sum_{k=1}^{K} w_k \phi_k(S, x) = w^\top \Phi(S, x) \approx \bar{V}(S, x). \tag{3.23}$$

A basis function $\phi_k(S, x)$ may be any non-linear function of S and x. If all basis functions in $\Phi(S, x)$ are linearly independent, we can use ordinary least squares to estimate the weight vector w, such that

$$w = \left(\sum_{n=1}^{N-1} \Phi(S^n, x^n) \Phi^\top(S^n, x^n) \right)^{-1} \left(\sum_{N=1}^{N-1} \Phi(S^n, x^n) r(S^n, x^n, S^{n+1}) \right). \tag{3.24}$$

This gives us an approximation of the function of expected immediate rewards, which would be sufficient for $\gamma = 0.0$. However, a basic idea of TD learning is that future rewards are included in the value function. Let w^{m-1} be an estimate of w at iteration m, before a least squares update is made. Then, we can use $\bar{V}(S, x; w^{m-1})$ to obtain a value estimate of the action taken at the successive state, such that

$$\begin{aligned} w^m = &\left(\sum_{n=1}^{N-1} \Phi(S^n, x^n) \Phi^\top(S^n, x^n) \right)^{-1} \\ &\times \left(\sum_{n=1}^{N-1} \Phi(S^n, x^n) \big(r(S^n, x^n, S^{n+1}) + \gamma \bar{V}(S^{n+1}, x^{n+1}; w^{m-1}) \big) \right). \end{aligned} \tag{3.25}$$

By repeating the least squares update over M iterations while collecting new samples, we can approximate the value function associated with the given policy. This algorithm is known as *least squares policy evaluation* (Nedic and Bertsekas, 2003). In the next section, we are going to present an approximate policy iteration algorithm which uses this method for policy improvement.

3.2.4 Approximate Policy Iteration

The idea of combining least squares updates with policy iteration has been first proposed in Lagoudakis and Parr (2003). The authors use the approximate value function of a given policy to compute an improved policy which is then used to construct a new approximate value function.

```
(1) Input arguments: approximate value function $\bar{V}(\,\cdot\,; w^0)$; initial state $S$
(2) Define function $z(m,k) = \big((m-1)N + k \bmod D\big) + 1$
(3) Do for $m = 1, 2, \dots, M$
    (3.1) Do for $n = 0, 1, \dots, N-1$
        (3.1.1) $x \leftarrow \pi^E(S)$
        (3.1.2) $L_{z(m,n)} \leftarrow (S, x)$
        (3.1.3) $S' \leftarrow S^M(S, x)$
    (3.2) $w^m \leftarrow \Big(\sum_{d=z(m,1)}^{z(m,D-1)} \Phi(S_d, x_d)\Phi^\top(S_d, x_d)\Big)^{-1}$
              $\times \Big(\sum_{d=z(m,1)}^{z(m,D-1)} \Phi(S_d, x_d)\big(r(S_d, x_d, S_{d+1}) + \gamma\bar{V}(S^{d+1}, x^{d+1}; w^{m-1})\big)\Big)$
(4) Return approximate value function $\bar{V}(\,\cdot\,; w^M)$
```

Figure 3.1: Least squares approximate policy iteration

The least squares approximate policy iteration (LSAPI) algorithm used in this chapter is shown in Figure 3.1.

The algorithm performs M value function updates, i.e., policy improvement steps. At each iteration m, it generates a sample of N state transitions by following policy π^E and then updates the weight vector. A simple random exploration policy is sufficient, e.g., epsilon-greedy exploration, since the action space is only one-dimensional. The function $S^M(S, x)$ implements the simulation model of the Markov decision process which produces a new state S' for a given state and action. The set $L = \{(S, x, r)_1, \dots, (S, x, r)_D\}$ represents a circular list implementation which stores a set of D state-action tuples that have been sampled sequentially. The least squares update is then made over the entire set. However, to speed up learning and ensure stability of the least squares update, only a fraction of the entire set is being changed at each iteration m, i.e., $D = kN$ with $1 \leq k \leq M$, $k \in \mathbb{N}$.

3.3 Numerical Results

Our research goals are to study the performance of the approximation algorithm for different parameter configurations and to analyze the influence of changes in model parameters on discounted rewards.

#	Parameter	Default Value	Lower Bound	Upper Bound
1	Supply Std Deviation (σ_Y)	2	1	4
2	Supply Autocorrelation (θ_Y)	0.5	0.0	0.75
3	Price Std Deviation (σ_P)	2	1	4
4	Price Autocorrelation (θ_P)	0.5	0.0	0.75
5	Correlation of Y and P (ρ_{PY})	-0.5	-0.75	0.0
6	Storage Capacity (C)	4	0	10
7	Storage Efficiency (η)	1	0.5	1
8	Discount Rate (γ)	0.9	0.0	0.9
9	Negative Imbalance Price Factor (u)	1.0	0.5	2.0
10	Positive Imbalance Price Factor (o)	0.0	0.0	0.5

Table 3.1: Default parameters and parameter ranges for the experimental design

3.3.1 Experimental Design

As experimental design, we adopted a so-called *space filling design* which samples not only at the edges of the hypercube which spans the experimental area but also at its interior. We generated a low-discrepancy *Faure* sequence to select design points which are uniformly scattered over the design space. Faure sequences have the useful property that a longer sequence can be constructed from a shorter one by resuming the sequence while still preserving uniformity, see Chen et al. (2006) for a review on designs for computer experiments.

As the basic setup for our studies we used default values and parameter intervals as shown in Table 3.1. Mean supply and mean price are fixed to $\mu_Y = \mu_P = 5$, because their magnitude only influences the volume but not the structure of the bids as long as the ratios of mean, variance and capacity are held constant. The storage efficiency is defined by its round-trip efficiency, $\eta = \eta^+\eta^-$.

To ensure tractability of the linear program, the stochastic processes of Y and P were bounded to the discrete set $\{0, 1, ..., 10\}$. Accordingly, the continuous counterparts were truncated at zero and 2μ to ensure comparability.

The parameters of LSAPI were pre-optimized so that the algorithm converges. The algorithm was set to collect batches of $T = 500$ samples and performs $M = 200$ value function updates. All samples were stored in a circular list of size $N = 25'000$. As exploration policy the algorithm used a simple epsilon-greedy policy, where a random bid is chosen with probability $\epsilon = 0.01$ and policy π is executed with probability $1 - \epsilon$.

We implemented the proposed algorithms in Java with Matrix computations being done by the Jama package. The linear programming approach was implemented and solved with FICO Xpress 7 and linked to Java via the XPRM model interface. All statistical analyses were done with SPSS 17.

Policy	Algorithm	Mean Reward	Mean Difference	
Optimal Discrete Policy	LP	19.94	LP	LSAPI-2
Second-order Polynomial	LSAPI-2	20.63	0.69 ± 0.31	–
Third-order Polynomial	LSAPI-3	20.74	0.80 ± 0.33	(0.11 ± 0.14)

Table 3.2: Mean rewards and 95% confidence intervals of the difference

3.3.2 Solution Quality

To test the quality of the approximation method, we benchmarked the policies of two value function approximations against policies computed with linear programming (LP). To use the optimal discrete policy during simulation, we projected the continuous states into the discrete state space by rounding to the next integer. Also, the round-trip efficiency was set to $\eta = 1.0$ to ensure comparability. For $\eta < 1.0$, we can expect the discrete policy to perform even worse, due to the additional rounding error.

The Weierstrass approximation theorem states that every continuous function defined on a closed interval can be approximated by a polynomial function to any degree of accuracy. Denote $z = \{x, y, p, g\}$ and $\bar{V}_2(S, x)$ and $\bar{V}_3(S, x)$ as second-order and third-order polynomial value function approximations, such that

$$\bar{V}_2(S, x) = w_0 + \sum_{i=1}^{4} w_i z_i + \sum_{i=1}^{4} \sum_{j=i}^{4} w_{ij} z_i z_j, \tag{3.26}$$

$$\bar{V}_3(S, x) = w_0 + \sum_{i=1}^{4} w_i z_i + \sum_{i=1}^{4} \sum_{j=i}^{4} w_{ij} z_i z_j + \sum_{i=1}^{4} \sum_{j=i}^{4} \sum_{k=j}^{4} w_{ijk} z_i z_j z_k. \tag{3.27}$$

Let LSAPI-2 and LSAPI-3 be the corresponding ADP algorithms. For a large enough sample size, we expect LSAPI-3 to compute a more accurate approximation than LSAPI-2 because $\bar{V}_2(S, x)$ is contained in $\bar{V}_2(S, x)$.

We ran the Faure sequence to generate 90 model configurations uniformly distributed over the parameter space. LSAPI-2, LSAPI-3 and LP then approximated an optimal policy for each configuration. The average reward from following each policy is recorded during simulation. The results are summarized in Table 3.2.

The table shows that LSAPI was able to approximate the optimal bidding strategy and delivers sufficient policies with both value function approximations. The LSAPI-2 policy achieved a 3.5% ($p < .01$) higher reward than the LP policy, and the LSAPI-3 policy achieved a 4.0% ($p < .01$) higher reward than the LP policy. The difference between LSAPI-2 and LSAPI-3, however, is not significant.

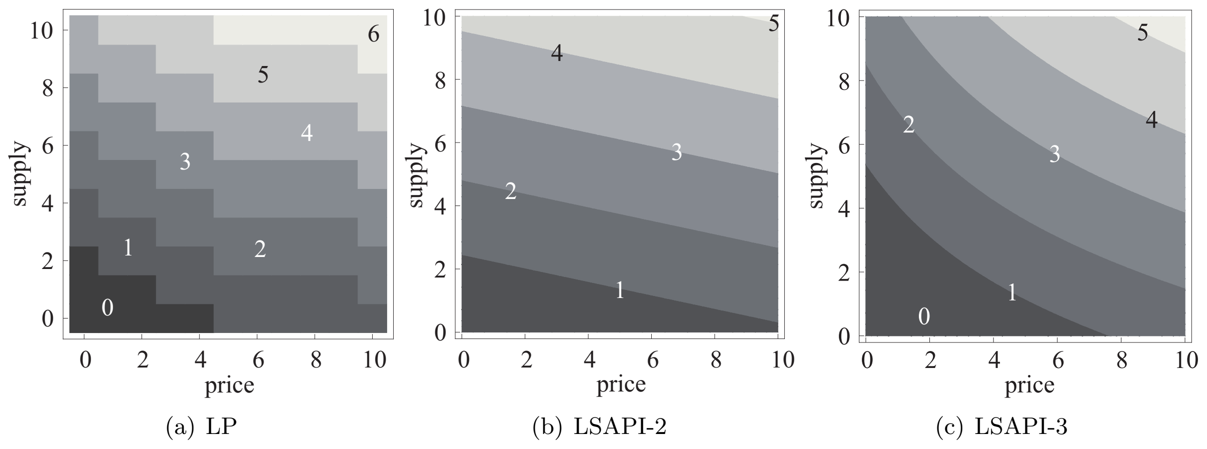


Figure 3.2: Contour plots of the approximate policies at $g = 0$

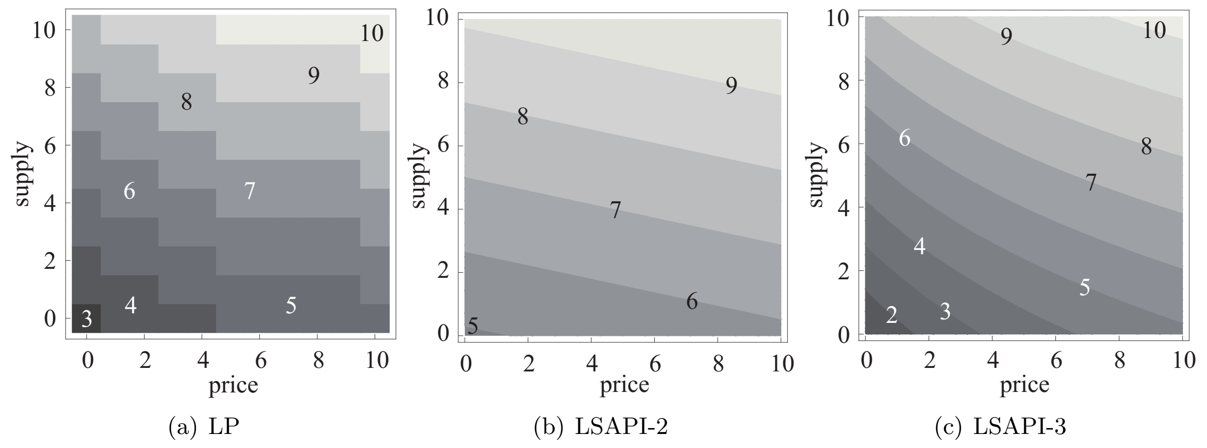


Figure 3.3: Contour plots of the approximate policies at $g = 4$

3.3.3 Policy Analysis

For illustrative purposes, we plot the different policies produced by LSAPI-2, LSAPI-3 and the LP for the default parameter configuration. Since the state-action space is four-dimensional, we fixed the storage state and drew contour plots of the surface which maps inputs of price and supply to bidding decisions, as suggested by the respective policies. Figures 3.2 and 3.3 show the contour plots of the three value functions for storage state $g = 0$ and $g = 4$, respectively.

The contour plot of the LP policy is non-smooth and areas with equal elevation, i.e., bidding decision, are shaded in the same color. For the two LSAPI policies, contour lines are drawn along integer elevations. Although the contour plots do not look alike, they share important similarities. Across all three policies, the elevation around the means of price and supply is nearly identical. The differences among the policies become larger towards the edges of the graph where state transition probabilities are lower and the associated errors have less impact

Model	R^2	R^2_{adj}	Std Error
$\hat{Q}_1$	0.530	0.528	1.900
$\hat{Q}_2$	0.898	0.897	0.889

Table 3.3: Model summary

on policy performance. The slopes of the contour lines exhibit a comparable inclination, so that bids respond to an increase in price and supply.

Note that the contour lines of the LP and LSAPI-3 policies share a comparable curvature. The LSAPI-2 policy, on the other hand, does not capture this detail because the derivative of its value function is linear in price, supply and storage. This leads to an increasing difference between LSAPI-2 and LSAPI-3/LP towards the edges of the graph where the state transition probabilities become lower. LSAPI-2 puts more weight on states which are frequently visited and balances the errors which emerge from decisions in extreme states.

We conclude that using simple polynomials to approximate the value function of the continuous state MDP yields excellent results. The corresponding policies performed even better than the optimal policy of the discrete counterpart.

3.3.4 Metamodel Analysis

To study the sensitivity of the objective value towards changes in model parameters, we analyze the model using regression analysis. We ran the model with 2000 different parameter configurations generated by the space filling design. To approximate the corresponding optimal rewards, we used LSAPI-2 as it produces high-quality policies at low computational cost.

Denote Z as the set of model parameters and Z_i as the value of the i-th parameter, and let us analyze the relationship of model parameters and discounted reward with the following two regression equations:

$$\hat{Q}_1 = \beta_0 + \beta_1 Z_1 + \beta_3 Z_3 + \beta_5 Z_5 + \beta_6 Z_6 + \beta_9 Z_9 + \beta_{10} Z_{10} \tag{3.28}$$

$$\hat{Q}_2 = \hat{Q}_1 + \beta_{12} Z_{12} + \beta_{345} Z_{345} + \sum_{i=1}^{10} \beta_{i6} Z_i Z_6 \tag{3.29}$$

The first equation captures the main effects of model parameters which have a direct influence on the response variable, i.e., the impact of a parameter change on the discounted reward obtained by following the LSAPI-2 policy. The second equation additionally includes interactions of model parameters with storage capacity Z_6 as well as two interaction terms to account for the indirect effect of autocorrelation. For both models, a summary of the regressions analysis is given in Table 3.3.

Table 3.4 shows the results from running a regression on $\tilde{Q}_1$. The explanatory power of

$\hat{Q}_1$	Main Effect	Coefficient	Std Error	t-Statistic	p-Value
Z_0	(Constant)	24.719	0.252	97.917	0.000
Z_1	Supply Std Deviation	-2.008	0.049	-40.924	0.000
Z_3	Price Std Deviation	-0.296	0.049	-6.040	0.000
Z_5	Correlation of Y and P	2.661	0.294	9.038	0.000
Z_6	Storage Capacity	0.274	0.015	18.614	0.000
Z_9	Negative Imbalance Price Factor	-0.805	0.098	-8.206	0.000
Z_{10}	Positive Imbalance Price Factor	1.636	0.294	5.555	0.000

Table 3.4: Coefficients of the main effects

this model is poor ($R^2 = 0.53$) because it only accounts for the individual effects of variability, storage capacity, correlation and imbalance costs on discounted rewards.

We find that supply and price deviation both have a negative effect on rewards ($\beta_1 < 0$, $\beta_3 < 0$), because higher variability increases the costs of imbalances. The difference in magnitude between both effects is substantial. This asymmetry originates in the nature of the stochastic processes with the unilateral dependency of price on supply. Changes in the standard deviation of the supply process therefore affect the price process but not vice versa, so that the overall impact of price variability is lower than the impact of supply variability. However, the correlation of price and supply increases rewards ($\beta_5 > 0$), which may be an argument in favor of investments in solar power which has the maximum yield when consumption is up, i.e., during daytime and summertime in warmer climates. As a matter of course, a higher storage capacity and price factor for positive imbalance have a positive effect on rewards ($\beta_6 > 0$, $\beta_{10} > 0$) while a higher price factor for negative imbalance has a negative effect ($\beta_9 < 0$).

Table 3.5 reports the results from running a regression on $\tilde{Q}_2$, but shows only the interaction terms. A high sample correlation ($R^2 = 0.898$) indicates that the second model already delivers a relatively accurate prediction of the relationship between model parameters and rewards.

Theoretically, supply and price autocorrelation do not have a direct effect on discounted rewards, because they explain the variability of the stochastic process only to some extent. Therefore, we study the interaction of supply autocorrelation with supply standard deviation. In contrast to the supply process, prices moreover depend on their correlation with supplies, so that we need to study the three-way interaction of price variability, autocorrelation and supply-price correlation. In both cases, we find that autocorrelation decreases the negative impact of variability ($\beta_{12} > 0$, $\beta_{345} > 0$), as it stabilizes the time series and thereby decreases the risk of imbalances.

Two-way interactions of model parameters with storage capacity uncover the individual impact of parameter changes on the contribution of storage capacity to rewards. While the value of storage increases in standard deviation ($\beta_{16} > 0$, $\beta_{36} > 0$), supply and price autocorrelation decrease the value of storage ($\beta_{26} < 0$, $\beta_{46} < 0$). This effect can be explained by comparing two

$\hat{Q}_2 \setminus \hat{Q}_1$	Interaction Effect	Coefficient	Std Error	t-Statistic	p-Value
Z_1Z_2	Supply Autocor × Supply Std Dev	1.525	0.057	26.986	0.000
$Z_3Z_4Z_5$	Price Autocor × Price Std Dev × Cor	0.899	0.148	6.080	0.000
Z_1Z_6	Capacity × Supply Std Deviation	0.156	0.008	19.561	0.000
Z_2Z_6	Capacity × Supply Autocorrelation	-0.162	0.026	-6.236	0.000
Z_3Z_6	Capacity × Price Std Deviation	0.019	0.008	2.399	0.017
Z_4Z_6	Capacity × Price Autocorrelation	-0.134	0.020	-6.577	0.000
Z_5Z_6	Capacity × Correlation of Y and P	-0.285	0.048	-5.967	0.000
Z_6Z_6	Capacity × Capacity	-0.063	0.003	-23.432	0.000
Z_7Z_6	Capacity × Storage Efficiency	1.391	0.024	58.343	0.000
Z_8Z_6	Capacity × Discount Rate	0.348	0.013	26.265	0.000
Z_9Z_6	Capacity × Negative Imbalance Price	0.233	0.016	14.692	0.000
$Z_{10}Z_6$	Capacity × Positive Imbalance Price	-0.581	0.048	-12.163	0.000

Table 3.5: Coefficients of the interaction effects

stochastic processes with identical first two moments, one with positive autocorrelation and one without. The process with positive autocorrelation will exhibit a wider average amplitude than the process without. With wider amplitudes we need more storage to buffer the same variability, which decreases the per unit value of storage capacity. The effect of autocorrelation is strong in a renewable power portfolio with only one power source. To decrease its unfavorable effect, investors should diversify, e.g., combine wind and solar power or invest cross-regionally.

Furthermore, we expect the marginal value of storage to decrease due to diseconomies of scale. The negative quadratic effect ($\beta_{66} < 0$) of storage capacity provides evidence for a concave function of storage value on storage capacity. Storage efficiency, on the other hand, increases the value of storage ($\beta_{76} > 0$).

From a technical point of view, the discount factor enables the approximation method to develop a bidding strategy which anticipates future states and decisions. Without a discount factor ($\gamma = 0.0$), there is no need for policy iteration and a one-shot least squares estimation as in (3.24) would be sufficient. This would produce a myopic policy which maximizes the second-stage reward, in line with the two-stage stochastic programs proposed in Fleten and Kristoffersen (2007) and García-González et al. (2008). Figure 3.4 shows the average reward, i.e., equivalent annuity value, for the default model configuration from using a multistage policy ($\gamma = 0.9$) versus using a myopic policy ($\gamma = 0.0$). For the system with storage ($C = 4$), the multistage policy yields a higher average reward than the myopic policy. A power producer with a hybrid system would therefore benefit from using a policy which anticipates future states and decisions. This also explains why the value of capacity increases in the discount factor ($\beta_{86} > 0$). For the system without storage ($C = 0$), however, multistage and myopic policy yield identical rewards, because there is no storage balance which links successive periods. In that case, a two-stage approach similar to the one proposed in Matevosyan and Söder (2006) is optimal.

Figure 3.4 also shows that for the given model configuration rewards of the system without

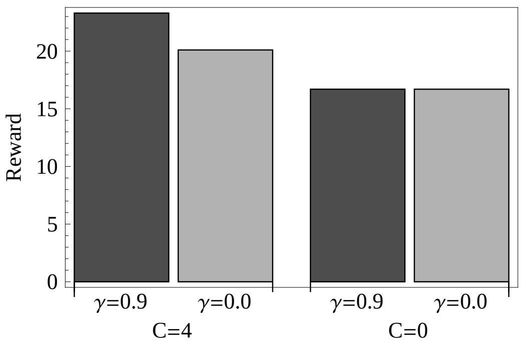


Figure 3.4: Rewards from optimal and myopic policies for systems with and without storage

storage are significantly lower than rewards of the hybrid system. We conclude that the financial benefit of storage is twofold: first, there is the risk associated with imbalance costs. A lower price for positive imbalance as well as a higher price for negative imbalance both increases the value of storage ($\beta_{96} > 0$, $\beta_{10,6} < 0$). Second, storage has the ability to take advantage of price arbitrage and alleviates the necessity to sell power when the price is low. Accordingly, the value of storage increases in negative correlation of price and supply ($\beta_{56} < 0$).

3.4 Discussion

In this chapter, we presented a model of the optimal bidding strategy for renewable power generation with storage option at a day-ahead market. We formulated the model as a continuous-state Markov decision process and presented a solution approach based on approximate dynamic programming. We used least squares policy evaluation within the approximate policy iteration framework to approximate the value function of the optimal policy. For policy evaluation and improvement, the algorithm had access to a simulation model of the process. As a benchmark, we computed the transition matrix of the stochastic decision process for a small discrete approximation of the state space and used linear programming to determine an optimal policy.

We found that approximate value functions based on simple polynomials yield better policies for the continuous state space than the optimal policy of a discretization. This effect will become even more profound when rounding errors occur due to storage efficiency losses.

A numerical analysis of the response surface of rewards on model parameters revealed that supply uncertainty, imbalance costs and a negative correlation of market price and supplies are the main drivers for investments in storage. An interesting result is that the value of storage decreases in autocorrelation, as more capacity is needed to buffer a stochastic process with a wider amplitude.

Chapter 4

Optimizing Trading Decisions for Hydro Storage Systems using Approximate Dual Dynamic Programming

The steady increase of electricity from intermittent renewable energy sources poses challenges for the electrical grid. A key component of a more flexible, smarter grid is the ability to store electricity and thereby to decouple electricity generation from electricity consumption. The most common large-scale storage technology for electricity is hydro storage. A hydro storage power plant either stores the natural flow of water or pumps water into an elevated reservoir to be able to release the water and produce electricity when it is needed. Hydro storage systems thereby offer the ability to buffer the intermittent supply of electricity from renewable power sources such as wind, solar, or run-off river. The European electricity mix, for example, consists of 15 percent hydropower with a total capacity of 260 gigawatts of which 45 gigawatts are pumped-hydro storage (Auer, 2011; Zuber, 2011). The growing share of renewable energies increases Europe's demand for storage, so that generating companies are currently investing about 26 billion Euros into new pumped-hydro storage plants with a total capacity of 27 gigawatts (Zuber, 2011).

Today, generating companies trade their capacities in wholesale electricity markets. Most generating companies in Austria, Germany, and Switzerland, for example, sell electricity at the European Energy Exchange (EEX), which is one of the largest European electricity markets. Since supply and demand have to be synchronized in advance, the EEX provides different types of forward markets, of which the day-ahead market and the intraday market are the most important markets for owners of storage plants. At the day-ahead market, producers (e.g., generating companies) place supply bids and consumers (e.g., electricity retailers) place demand bids for each hour of the following day, one day ahead of delivery. After the day-ahead market is closed, the intraday market allows market participants to clear imbalances that arise during the day up to 45 minutes before delivery. In case actual volumes deviate from day-ahead or intraday bids, all remaining imbalances are automatically cleared at the balancing market with a high risk of additional cost. A generating company with hydro storage capacities typically buys electricity at the market when the price is low and sells electricity when the price is high, while trying to mitigate the risk of positive and negative imbalances.

Evidently, trading with a system of hydro storage plants in a wholesale electricity market involves many decisions as well as a great deal of uncertainty. In particular, there exist two major challenges to solve the problem efficiently. First, not only are day-ahead and intraday prices uncertain, but also the development of electricity prices over time as well as the inflow of water into the reservoirs. Second, a system of hydro storage plants with multiple reservoirs requires a coordinated water release policy, since upstream releases influence downstream reservoir levels. In addition to day-ahead and intraday bidding decisions, a generating company has to decide about water releases from multiple reservoirs over time. Future decisions and states of the system as well as their probabilities therefore have to be considered while making decisions today. This turns storage operation into a complex stochastic-dynamic decision problem.

In the literature, the day-ahead bidding problem is typically modeled as a two-stage stochastic program, with bidding decisions at the first stage and price realizations as well as operational decisions at the second stage (Baíllo et al., 2004; Nowak et al., 2005; Fleten and Kristoffersen, 2007; García-González et al., 2007). A two-stage approach is well-suited for optimizing bidding decisions in the short-term, but does not anticipate future storage states and decisions. To optimize bidding decisions over a longer planning horizon, we have to solve a multi-stage stochastic programming problem. For this class of problems, two basic solution strategies have emerged in the literature.

One strategy is to construct a scenario tree to represent uncertainty and solve the problem as one large mathematical program (Heitsch and Römisch, 2003; Eichhorn et al., 2009; Hochreiter and Wozabal, 2010). This strategy can handle discrete decisions as well as any type of stochastic process, but is limited to problems with a small number of stages. Fleten and Kristoffersen (2008) and Matevosyan et al. (2009) propose mixed-integer programs of hydro storage operation where a scenario tree is used to model uncertainty over a weekly planning horizon. A comparison of solution methods for a tree-based stochastic unit commitment problem is given in Cerisola et al. (2009).

Another strategy is based on formulating the problem as a dynamic program and then applying Benders' decomposition to recursively construct the value function at each stage around a set of sample decisions (Pereira and Pinto, 1991). This strategy, also known as *stochastic dual dynamic programming* (SDDP), can handle problems with a large number of stages as long as the optimization problem at each stage is convex and the stochastic process stagewise independent. Most SDDP formulations of hydro storage operation only consider inflow or demand uncertainty, e.g., Flach et al. (2010), Guigues (2011), and Philpott and de Matos (2011). To the best of our knowledge, the only SDDP formulation that also considers price uncertainty is given in Gjelskvik et al. (2010). However, the authors only model weekly price averages, which keeps the problem tractable but also underrates the short-term value of storage.

We propose a model formulation which decomposes the multi-stage problem into an *intrastage* and an *interstage* problem (Pritchard et al., 2005). Day-ahead bidding decisions as well as hourly reservoir operations are modeled as part of the intrastage problem which is formulated as a stochastic program with random prices. Reservoir management from day to day, on the other hand, is modeled as part of the interstage problem which is formulated as a Markov decision process (MDP). Accordingly, we separate intrastage (hourly) from interstage (daily) randomness by modelling hourly price distributions conditional on an environmental state variable that follows a Markov process with daily transitions. To solve the problem efficiently, we integrate SDDP with ideas from approximate dynamic programming (ADP). ADP algorithms simulate the state transition process of an MDP and use the sampled information to approximate the high-dimensional value function by a function of much lower complexity (Powell, 2007). Loehndorf and Minner (2010) propose an ADP algorithm to optimize bidding and operational decisions of a single storage unit, but only for a simplified decision process.

The solution strategy proposed in this chapter aims at approximating the value function of the interstage problem. First, we fit a discrete-state Markov chain to the continuous-state Markov process by using a clustering method to reduce the state space. The resulting reference states then determine the intrastage price distributions. Second, we propose to use a relaxed version of the intrastage problem to approximate the value function of the interstage problem. As in SDDP, we iteratively solve the interstage problem using forward simulation to sample candidate decisions and backward recursion to construct an approximation of the value function. Third, we construct a polyhedral approximation by removing candidate decisions to accelerate the sampling process. The polyhedral value function approximation of the interstage problem is then used inside the original intrastage problem to find near-optimal bidding and operational decisions. Due to its resemblance to SDDP, we refer to the solution approach as *approximate dual dynamic programming* (ADDP).

4.1 Model Formulation

4.1.1 Assumptions

We consider the stochastic unit commitment problem of a power generating company that operates a network of hydro storage plants and participates in a wholesale electricity market.

The objective of the company is to maximize expected profits from buying and selling electricity while operating its resources efficiently.

We assume that the company is planning storage operation over an entire year, but schedules its resources on a daily basis. Uncertainty enters the planning problem through stochastic natural inflows into the reservoirs as well as through stochastic electricity prices. We assume that the dynamics of the random variables can be separated into an interday and an intraday process. The interday process is characterized by a state transition model of environmental variables, e.g., temperature, wind speed, natural inflows, as well as calendar information, i.e., day of the year, day of the week. This process is assumed to be a Markov chain with a finite state space. The intraday process describes random hourly electricity prices which depend on the realization of interday randomness.

We assume that the electricity market implements a multi-settlement system with a day-ahead, an intraday (hour-ahead) and a balancing (real-time) market. The company makes the majority of its trades in the day-ahead market where it places price-dependent supply and demand bids for each hour of the following day. A bid can have multiple parts, so that producers and consumers are bidding for different volumes at different prices. After the day-ahead market is closed, the system operator announces a uniform clearing price. Day-ahead bidding therefore takes place under price and volume uncertainty. In case produced volumes deviate from day-ahead bids, the company clears all foreseeable imbalances at the intraday market and does not deliberatively use the balancing market. Since the intraday market has a lower market depth than the day-ahead market, we assume that the generating company is price-taker in the day-ahead market, but price-setter in the intraday market. Moreover, we assume that the expected day-ahead price equals the expected intraday price.

The topology of the network of hydro storage plants is convergent, so that each reservoir may have multiple inflows but only a single outflow. Connected reservoirs are close, so that there are no significant delays regarding the flow of water from one reservoir to another. We assume that head effects can be ignored so that the power conversion function only depends on water release per time unit but not on reservoir levels. The natural inflow of water into a reservoir is state-dependent and remains constant throughout the day. Each reservoir is associated with a single turbine and possibly a pump.

4.1.2 Markov Decision Process

We model the interday decision process of storage operation as a finite horizon Markov decision process (MDP) with decision epoch of length one day. Denote t as the time index for a day of the year. Randomness is separated into a process of environmental variables $(S_t)_{t=1}^T$ and a process of intraday electricity prices $(P_t)_{t=1}^T$. Both processes are discrete and defined on a common probability space $(\Omega, \Xi, \mathcal{P})$. We assume that S_t and P_t are adapted to filtrations $\mathcal{F}_1$

and $\mathcal{F}_2$, respectively, and $\mathcal{F}_1 \leq \mathcal{F}_2$, i.e., given a realization of S_t, the distribution of $P_t|\mathcal{F}_1$ is known.

The objective of the generating company is to maximize its discounted expected profits for a given environmental state $S_t \in \mathcal{S}_t$ and initial storage states $R_{t-1} \in \mathcal{R}$ in stage $t \in \{1, \ldots, T\}$, with $\mathcal{S}_t$ being the set of environmental states in t and $\mathcal{R}$ being the set of all possible reservoir states. Denote $\mathbb{P}(S_{t+1}|S_t)$ as the state transition probability of the Markov process. Let $\pi = \{\pi_1, \ldots, \pi_T\}$ be a decision policy that encompasses all operational decision variables subject to the state-dependent feasible set $\Pi_t(S_t, R_{t-1})$ (see Section 4.1.3), and define $C(S_t, R_{t-1}, \pi_t)$ as the random intraday profit (contribution) and γ as discount factor. For R_0 and V_{T+1} fixed, the value of being in state S_t with initial reservoir states R_{t-1} is given by the optimality equations

$$V_t(S_t, R_{t-1}) = \max_{\pi_t \in \Pi_t(S_t, R_{t-1})} \Big\{ \mathbb{E}\Big[C(S_t, R_{t-1}, \pi_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}_{t+1}} \mathbb{P}(S_{t+1}|S_t)\ V_{t+1}\big(S_{t+1}, R_t(\pi_t)\big)\Big]\Big\}, \quad (4.1)$$

for $S_t \in \mathcal{S}_t$, $R_{t-1} \in \mathcal{R}$ and $t = 1, \ldots, T$. Since π_t assigns a decision to every realization of intraday randomness, it results in random reservoir states $R_t(\pi_t)$. An optimal decision policy maximizes the sum of expected intraday profits and expected future profits. Future profits depend on the random state transition from S_t to S_{t+1} as well as the (random) final reservoir state $R_t = R_t(\pi_t)$ in t which is the initial reservoir state in $t+1$. For notational convenience, we suppress the dependence of R_t on π_t and Π_t on S_t, R_{t-1} where no confusion can arise.

In line with Powell (2007), let us reformulate (4.1) using the post-decision state. Denote $\bar{V}_t$ as value function around the post-decision state, which gives us the value of being in state S_t at the end of the day after realization of R_t but before a random transition to the next state. For a fixed function $\bar{V}_T$, the post-decision value function is

$$\begin{aligned}\bar{V}_t(S_t, R_t) &= \sum_{S_{t+1} \in \mathcal{S}_{t+1}} \mathbb{P}(S_{t+1}|S_t)\ V_{t+1}(S_{t+1}, R_t) \\ &= \sum_{S_{t+1} \in \mathcal{S}_{t+1}} \mathbb{P}(S_{t+1}|S_t) \max_{\pi_{t+1} \in \Pi_{t+1}} \Big\{ \mathbb{E}\Big[C(S_{t+1}, R_t, \pi_{t+1}) + \gamma \bar{V}_{t+1}(S_{t+1}, R_{t+1})\Big]\Big\}, \quad (4.2)\end{aligned}$$

for $S_t \in \mathcal{S}_t$, $R_t \in \mathcal{R}$ and $t = 1, \ldots, T-1$. This formulation of the optimality equations is equivalent to (4.1) but provides us with a computational advantage, because the expectation operator associated with the state transition is now outside of the maximization problem. For now, let us assume that $\bar{V}_t(S_t, R_t)$ is known. In Section 4.2, we are going to show how to recursively built an approximation of the post-decision value function.

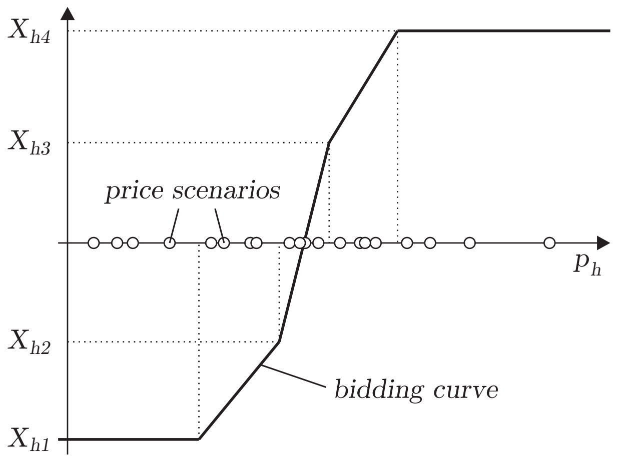


Figure 4.1: Bidding curve with four breakpoints and four price scenarios per segment

4.1.3 Stochastic Programming Formulation

For a given post-decision value function, the intraday problem can be formulated as a stochastic program with recourse, with the objective to maximize the expected profit for a given state S_t and initial reservoir state R_t. Although all variables and random parameters of the intraday problem depend on S_t, we suppress this dependence to streamline the presentation. Furthermore, most parameters and decision variables depend on an hour h and a scenario s. We indicate this by adding the corresponding subscripts.

First-Stage Decision: Day-Ahead Bidding

The first stage of the stochastic program involves fixing price-dependent bidding curves (see Figure 4.1). For each hour $h \in \mathscr{H} = \{1, \dots, 24\}$ of the following day, the generating company submits I price-volume pairs (ρ_{hi}, X_{hi}), with $\rho_{hi} < \rho_{hi+1}$ and $X_{hi} \leq X_{hi+1}$ for $i < I$. A linear interpolation of these pairs yields a monotone increasing, piecewise-linear function that maps price realizations to bidding decisions. However, choosing prices and volumes simultaneously yields a non-convex decision problem. In line with Fleten and Kristoffersen (2007), we therefore fix the price points in advance and only decide about bidding volumes (in megawatt hours).

Denote $p_s \in \mathbb{R}^{24}$, $s \in \mathscr{S} = \{1, \dots, K\}$ as the finite set of realizations of the price process $P_t|S_t$ and p_{sh} as a day-ahead price realization in hour h. We assume $\mathbb{P}(P_t = p_s|S_t) = 1/K$, so that each price scenario has equal probability. The realized day-ahead bid x^d_{sh} in scenario s depends on the bidding curve as well as the realized day-ahead prices,

$$x^d_{sh} = \begin{cases} X_{h1} & \text{if } p_{sh} < \rho_{h1}, & \forall\, s \in \mathscr{S}, h \in \mathscr{H}, \\ X_{hi-1} + \frac{X_{hi} - X_{hi-1}}{\rho_{hi} - \rho_{hi-1}} p_{sh} & \text{if } \rho_{hi-1} \leq p_{sh} < \rho_{hi},\ 1 < i \leq I, & \forall\, s \in \mathscr{S}, h \in \mathscr{H}, \\ X_{hI} & \text{if } p_{sh} \geq \rho_{hI}, & \forall\, s \in \mathscr{S}, h \in \mathscr{H}. \end{cases} \tag{4.3}$$

Depending on the sign of the bid, each bidding decision either represents an offer ($X_{hi} \geq 0$) or a purchase ($X_{hi} < 0$) of electricity (in megawatt hours). To additionally enforce monotonicity, we include the following constraint,

$$X_{hi-1} \leq X_{hi} \quad \forall\, h \in \mathscr{H}, i \in \{2, \ldots, I\}. \tag{4.4}$$

Instead of spreading the price breakpoints ρ_{hi} evenly over the price domain, we assign each line segment approximately the same number of price scenarios. More specifically, we compute the breakpoints by first partitioning the set of sorted prices into $I+1$ subsets and then calculate ρ_{hi} as the mean of the adjacent prices that are not of the same subset, i.e.,

$$\rho_{hi} = \frac{p'_{hg(i)-1} + p'_{hg(i)}}{2}, \quad g(i) = \left\lfloor \frac{iK}{I+1} + 1 \right\rfloor \quad \forall\, h \in \mathscr{H}, i \in \mathscr{I}, \tag{4.5}$$

with p'_{sh} as the day-ahead prices sorted in ascending order. The resulting bidding curve is smoother in areas where the probability mass is high and coarser where the probability mass is low. Note that the number of scenarios K has to satisfy $K \geq 2I + 2$. For example, if there was only one scenario per segment, first-stage bidding decisions could be made with full knowledge of second-stage randomness, which violates non-anticipativity.

Second-Stage Decision: Short-Term Unit Commitment

Short-term operational decisions are modeled at the second stage of the stochastic program. At this stage, day-ahead bids have realized and the generating company uses either storage capacities or the intraday market to close its positions. In line with other authors (Fleten and Kristoffersen, 2007; García-González et al., 2007), we model the unit commitment problem as a mixed-integer program.

Denote $\mathscr{J} = \{1, \ldots, J\}$ as the set of reservoirs, c_{sjh} as charge into reservoir j, and d_{sjh} as discharge from reservoir j (in metric tons). The topology of the reservoir network is defined by matrix $A = (A_{jk}) \in \{-1, 0, 1\}^{J \times J}$, with $A_{jk} = 1$ if water can be released from j into k and $A_{kj} = -1$ if water can be pumped from k into j. The hourly natural inflow of water into reservoir j is given by IN_{tj} on day t. Denote r_{sjh} as the reservoir state with $r_{sj0} = R_{t-1j}$ and o_{sjh} as the overflow (or spill). Then, the storage state at the end of hour h is given by the following balance equation

$$\begin{aligned} r_{sjh} = {} & r_{sjh-1} - d_{sjh} + c_{sjh} - o_{sjh} \\ & + \sum_{k \in \mathscr{J} : A_{kj} = 1} (d_{skh} + o_{skh}) - \sum_{k \in \mathscr{J} : A_{jk} = -1} c_{skh} + \mathrm{IN}_{tj} \quad \forall\, s \in \mathscr{S}, j \in \mathscr{J}, h \in \mathscr{H}. \end{aligned} \tag{4.6}$$

The generating company must balance realized day-ahead and intraday bids with power generation and consumption. Denote x^b_{sh} as the amount of power (in megawatt hours) sold or purchased at the intraday market in hour h and scenario s. All open positions are closed if

$$x^d_{sh} + x^b_{sh} = \sum_{j\in\mathscr{J}} (\eta^+_j d_{sjh} - \eta^-_j c_{sjh}) \quad \forall\ s \in \mathscr{S}, h \in \mathscr{H}, \tag{4.7}$$

with η^+_j and η^-_j as constant power conversion factors (in megawatts per metric ton of water) which relate flow volume to power quantity.

Charge and discharge decisions are constrained by minimum and maximum capacities of pumps and turbines. Denote UB^R_j as the maximum volume of reservoir j (in metric tons), $[\mathrm{LB}^+_j, \mathrm{UB}^+_j]$ as power limits of the j-th turbine, and $[\mathrm{LB}^-_j, \mathrm{UB}^-_j]$ as power limits of the j-th pump (in megawatt). Then,

$$r_{sjh} \leq \mathrm{UB}^R_j \qquad \forall\ s \in \mathscr{S}, j \in \mathscr{J}, h \in \mathscr{H}, \tag{4.8}$$

$$z^+_{sjh}\,\mathrm{LB}^+_j \leq \eta^+_j d_{sjh} \leq z^+_{sjh}\,\mathrm{UB}^+_j \qquad \forall\ s \in \mathscr{S}, j \in \mathscr{J}, h \in \mathscr{H}, \tag{4.9}$$

$$z^-_{sjh}\,\mathrm{LB}^-_j \leq \eta^-_j c_{sjh} \leq z^-_{sjh}\,\mathrm{UB}^-_j \qquad \forall\ s \in \mathscr{S}, j \in \mathscr{J}, h \in \mathscr{H}, \tag{4.10}$$

with binary variables, z^+_{sjh} and z^-_{sjh}, to model the on/off status of turbines and pumps, respectively.

Objective Function

The objective of the generating company is to maximize its expected intraday profits through efficient bidding and storage operation while considering the expected future value of storage as defined by the post-decision value function. Following Pereira and Pinto (1991), we model the post-decision value v_s as a concave, piecewise-linear function of the final reservoir states r_{sj24} at the end of the day. Note that r_{sj24} is a realization of the j-th element of R_t in (4.1) and (4.2). For a given state S_t, the post-decision value function is defined as the minimum of a set of hyperplanes $\mathscr{N} = \{1, \dots, N\}$ with intercepts $a(S_t)_n$ and slopes $b(S_t)_{nj}$ so that the future value of storage is given by

$$v_s = \min_{n\in\mathscr{N}} \left\{ a(S_t)_n + \sum_{j\in\mathscr{J}} b(S_t)_{nj} r_{sj24} \right\} \qquad \forall\ s \in \mathscr{S}. \tag{4.11}$$

If we add v_s to the objective function of a maximization problem, we can reformulate (4.11) by the following set of linear constraints,

$$v_s \leq a(S_t)_n + \sum_{j\in\mathscr{J}} b(S_t)_{nj} r_{sj24} \qquad \forall\ n \in \mathscr{N}, s \in \mathscr{S}. \tag{4.12}$$

Denote $p_{sh} - \beta x^b_{sh}$ as the expected intraday price, with $\beta \geq 0$ as the slope of the price-response function. For a given state S_t and an initial reservoir state R_{t-1}, we formulate the optimization problem as the following stochastic mixed-integer quadratic program

$$
\begin{aligned}
V_t(S_t, R_{t-1}) = \max\ & \frac{1}{K}\sum_{s=1}^{K}\sum_{h=1}^{24}\Big(p_{sh}x^d_{sh} + (p_{sh} - \beta x^b_{sh})x^b_{sh} + \gamma v_s\Big) \qquad (4.13)\\
s.t.\ & (4.3), (4.4), (4.6), (4.7), (4.8), (4.9), (4.10), (4.12);\\
& X_{hi} \in \mathbb{R}\ \forall\ h \in \mathscr{H}, i \in \mathscr{I};\ x^d_{sh}, x^b_{sh} \in \mathbb{R}\ \forall\ s \in \mathscr{S}, h \in \mathscr{H};\\
& r_{sjh}, c_{sjh}, d_{sjh}, o_{sjh} \geq 0\ \forall\ s \in \mathscr{S}, j \in \mathscr{J}, h \in \mathscr{H};\\
& z^+_{sjh}, z^-_{sjh} \in \{0,1\}\ \forall\ s \in \mathscr{S}, j \in \mathscr{J}, h \in \mathscr{H};\\
& v_s \geq 0\ \forall\ s \in \mathscr{S}.
\end{aligned}
$$

4.2 Solution Methods

To obtain the hyperplanes required in (4.11) or (4.12), we integrate stochastic dual dynamic programming (SDDP) with ideas from approximate dynamic programming (ADP). The method referred to as approximate dual dynamic programming (ADDP) constructs a polyhedral approximation of the post-decision value function defined in Section 4.1.2 by sampling the state transitions of the Markov decision process.

4.2.1 Approximate Value Function

To be able to construct a polyhedral approximation of the post-decision value function, we relax certain requirements of the original model formulation.

Definition (Relaxed Problem) Define the relaxed problem as the linear relaxation of the stochastic mixed-integer quadratic program in (4.13) with $\beta = 0$, and denote $V'_t(S_t, R_{t-1})$ as its optimal objective value and $\bar{V}'_{t-1}$ as the corresponding post-decision value function.

By modeling the post-decision value function as the minimum of a set of linear functions (4.12), we tacitly assume that the true function admits a tight concave approximation. While we cannot make this assertion for the post-decision value function associated with the original model formulation, we can show that concavity holds for the post-decision value function of the relaxed problem.

Proposition 4.2.1 *The objective value $V'_t(S_t, R_{t-1})$ as well as the post-decision value $\bar{V}'_t(S_t, R_t)$ are both concave in the reservoir levels.*

Proof With the binary variables relaxed to $z^+_{sjh}, z^-_{sjh} \in [0,1]$, the maximization problem in (4.13) is an ordinary linear program. From the theory of linear programming we know that a problem of this type is jointly concave in the right-hand sides of its constraints, e.g., by Proposition 2.22 in (Rockafellar and Wets, 1998). The vector R_t enters the right-hand side of equation (4.6). Therefore, $V'_t(S_t, R_{t-1})$ is concave in R_{t-1}. Denote $V'_T(S_T, R_{T-1})$ as the objective value of the relaxed problem in the final stage and $\bar{V}'_T$ as an arbitrary piecewise-linear function which is assumed to be concave in R_T. Since $V'_T(S_T, R_{T-1})$ is concave in R_{T-1}, the expected value $\sum_{S_T \in \mathcal{S}} \mathbb{P}(S_T|S_{T-1})\; V'_T(S_T, R_{T-1})$ is concave in R_{T-1}. Hence, $\bar{V}'_{T-1}(S_{T-1}, R_{T-1})$ is concave in R_{T-1}. Concavity of $\bar{V}'_t(S_t, R_t)$ for $t = 1, \dots, T-2$ follows by backward induction. $\square$

With $\beta = 0$ in the relaxed problem, we assume that the generating company is price-taker in both markets, day-ahead and intraday. Without an intraday price response, however, the company has no incentive to trade in the day-ahead market as long as we assume that the mean intraday price is identical to the realized day-ahead price. Instead, the company could move all of its trades to the intraday market. In that case, the relaxed version of the stochastic program can be decomposed into K linear programs, which is supported by the following proposition.

Proposition 4.2.2 *Without a price response, i.e., $\beta = 0$, the relaxed problem is the sample average of K linear programs.*

Proof For each feasible decision $(x^b_{sh}, x^d_{sh}) = (\bar{x}^b, \bar{x}^d)$, the decision $(x^b_{sh}, x^d_{sh}) = (\bar{x}^b + \bar{x}^d, 0)$ is feasible with respect to (4.7), $\forall\; s \in \mathscr{S}, h \in \mathscr{H}$. With $\beta = 0$, the marginal prices of x^d_{sh} and x^b_{sh} in (4.13) are identical and the decisions $(\bar{x}^b, \bar{x}^d)$ and $(\bar{x}^b + \bar{x}^d, 0)$ have the same objective values. Hence, there exists an optimal decision, where $X_{hi} = 0\; \forall\; h \in \mathscr{H}, i \in \mathscr{I}$. The non-anticipativity constraints (4.3) can then be dropped, and the relaxed problem can be decomposed, such that

$$V'_t(S_t, R_{t-1}) = \frac{1}{K} \sum_{s=1}^{K} V'_{ts}(S_t, R_{t-1}),$$

where $V'_{ts}(S_t, R_{t-1})$ is defined as $V'_t(S_t, R_t)$ for a single scenario $s \in \mathscr{S}$. $\square$

The objective value associated with the relaxed problem is an upper bound of the optimal objective value, i.e., $V'_t(S_t, R_{t-1}) \geq V_t(S_t, R_{t-1})$. An operational policy, where the generating company does not bid in the day-ahead market, however, is of little practical use. Nevertheless, as long as the difference between upper bound and optimum is reasonably small, we can use the optimal solution of the relaxed problem to construct an approximation of the post-decision value function. We then use this function inside the original problem formulation to compute near-optimal intraday decisions. As we will see in Section 4.4.1, the difference is small for the actual problems considered in this chapter.

Let us briefly outline how a polyhedral approximation of the post-decision value function can be constructed. Since $V'_t(S_t, \cdot)$ is the optimal objective value of a linear program, the post-decision value function of the relaxed problem can be described by a concave, piecewise-linear function, i.e., by a polyhedral function. We can construct an approximation $\hat{\hat{V}}'_{t-1}(S_{t-1}, R)$ of the post-decision value function by first defining a set of sample reservoir states, $\{\hat{R}_1, \ldots, \hat{R}_N\}$, with $\hat{R}_n \in \mathcal{R}$, and then deriving the corresponding hyperplanes going through

$$(\hat{R}_{11}, \ldots, \hat{R}_{1J}, V'(S_t, \hat{R}_1)), \ldots, (\hat{R}_{N1}, \ldots, \hat{R}_{NJ}, V'_t(S_t, \hat{R_N})) \ \forall \ S_t \in \mathcal{S}_t.$$

To obtain the hyperplanes, let $\partial_R V_t(S_t, R_t)$ be the set of super-gradients of the function $R_t \mapsto V'_t(S_t, R_t)$. From this set, we select a super-gradient, $b(S_t) \in \partial_R V'_t(S_t, \hat{R})$, which is the slope of the supporting hyperplane of $V'_t(S_t, \cdot)$ going through $(\hat{R}_1, \ldots, \hat{R}_J, V'_t(S_t, \hat{R}))$. The hyperplane is given by the linear function

$$H(S_t, R; \hat{R}_i) = a(S_t) + b(S_t)^\top R, \quad a(S_t) = V'(S_t, \hat{R}) - \sum_{j \in \mathscr{J}} b(S_t)_j \hat{R}_{ij}, \tag{4.14}$$

with $a(S_t) \in \mathbb{R}$ as the intercept and $b(S_t) \in \mathbb{R}^J$ as the vector of slopes. Since we are dealing with linear programs, the slopes can be obtained from the dual variables λ associated with constraints (4.6) for $h = 1$,

$$b(S_t)_j = \sum_{s \in \mathscr{S}} \lambda_{sj1}. \tag{4.15}$$

The resulting approximate post-decision value function is then given by

$$\hat{\hat{V}}'_{t-1}(S_{t-1}, R) = \min \left\{ \sum_{S_t \in \mathcal{S}_t} \mathbb{P}(S_t | S_{t-1}) \left(a(S_t)_n + b(S_t)_n^\top (R - \hat{R}_n) \right), \ n = 1, ..., N \right\}, \tag{4.16}$$

where the hyperplane going through $\hat{R}_n$ is the weighted sum of all hyperplanes $H(S_t, R; \hat{R}_n)$ over all successor states. For a given set of sample reservoir states at each stage, a polyhedral approximation of the post-decision value function can be easily constructed by solving the dynamic program using backward recursion.

4.2.2 Approximate Dual Dynamic Programming

Although the number of supporting hyperplanes of $V'_t(S_t, \cdot)$ is finite, computing all hyperplanes is prohibitive for larger problems. Like SDDP, the ADDP algorithm therefore uses Monte Carlo simulation to define a set of sample reservoir states, thereby finding those hyperplanes that are necessary to obtain an optimal decision policy.

The ADDP algorithm is outlined in Figure 4.2. The algorithm is initialized with an environmental state S_1, a reservoir state R_0, and a function of the salvage value of the final reservoir

```
Input arguments: initial states $S_1$ and $R_0$, salvage value function $\hat{\bar{V}}'_T$
Do for $n = 1, 2, \dots, N$
  Forward Pass
  (1) Do for $t = 1, 2, \dots, T-1$
    (1.1) Sample $p_s$ from $P_t|S_t$
    (1.2) Solve $\hat{R}_{nt} \leftarrow \arg\ \max_{\pi_t} \left\{ C(S_t, R_{t-1}, \pi_t) + \hat{\bar{V}}'_t(S_t, R_t(\pi_t)) \right\}$ for the single scenario $p_s$
    (1.3) Sample $S_{t+1} \leftarrow S^M(S_t)$
  Backward Pass
  (2) Do for $t = T, T-1, \dots, 2$
    (2.1) Do for all $S_t \in \mathcal{S}_t$
      (2.1.1) Do for $m \in \mathcal{M}_t \cup \{n\}$
        (2.1.1.1) Get hyperplane $(a(S_t)_m, b(S_t)_m) \leftarrow H_{mt}(S_t, R; \hat{R}_{mt-1}) \in \partial_R V'_t(S_t, \hat{R}_{mt-1})$
    (2.2) If $\exists\ S_t \in \mathcal{S}_t : |\hat{V}'_t(S_t, \hat{R}_{nt-1}) - V'_t(S_t, \hat{R}_{nt-1})| > \varepsilon$ then $\mathcal{M}_t \leftarrow \mathcal{M}_t \cup \{n\}$
    (2.3) Do for all $S_{t-1} \in \mathcal{S}_{t-1}$
      (2.3.1) $\hat{\bar{V}}'_{t-1}(S_{t-1}, R) \leftarrow \min \left\{ \sum_{S_t \in \mathcal{S}_t} P(S_t|S_{t-1}) \left( a(S_t)_m + b(S_t)_m^\top (R - \hat{R}_{mt}) \right), m \in \mathcal{M}_t \right\}$
Return post-decision value functions $\hat{\bar{V}}'_t$ $(t = 1, \dots, T-1)$
```

Figure 4.2: Approximate dual dynamic programming for Markov decision processes

state $\bar{V}'_T$. Over N iterations, ADDP alternates between a forward and a backward pass. During the forward pass, the algorithm generates new states by sampling the state transition function, S^M. For each state, the algorithm solves the relaxed problem for a single (random) price scenario using the current approximation of the value function and stores the final reservoir state (Step 1.2). During the backward pass, in each stage, the algorithm loops over all environmental states and previously stored reservoir states and computes the supporting hyperplanes (Step 2.1). For each predecessor state, we compute the weighted sum of all hyperplanes over all successor states and then update the approximation of the post-decision function (Step 2.3).

In conventional SDDP, the size of the set of sample reservoir states increases by one during each iteration of the outer loop. Some hyperplanes around the set of reservoir states, however, may be redundant or at least *similar* to existing hyperplanes. For ADDP, we therefore propose that hyperplanes which do not improve the approximation quality by more than ε are being omitted (Step 2.2). Denote $\hat{V}'$ as the approximate (pre-decision) value function constructed from a set of hyperplanes H_{mt}, with $m \in \mathcal{M}_t$, such that

$$\hat{V}_t'(S_t, R) = \min \left\{ \sum_{S_t \in \mathcal{S}_t} \left(a(S_t)_m + b(S_t)_m^\top (R - \hat{R}_{mt}) \right), m \in \mathcal{M}_t \right\}. \tag{4.17}$$

A new hyperplane H_{nt} is added to $\mathcal{M}_t$ if

$$\exists\ S_t \in \mathcal{S}_t : \hat{V}_t'(S_t, \hat{R}_{nt-1}) - V_t'(S_t, \hat{R}_{nt-1}) > \varepsilon. \tag{4.18}$$

In this way, ADDP converges to an upper bound of the solution to the relaxed problem, since the approximate value function in general remains an approximation and never converges to the true value function. Note that we also obtain an upper bound if we stop ADDP before an optimal solution is found, as this is often done in the literature, e.g., Flach et al. (2010); Philpott and de Matos (2011). A practical advantage of using $\varepsilon > 0$ instead of $\varepsilon = 0$ is that, omitting new hyperplanes accelerates computation of the outer loop, which allows a larger number of states to be sampled in the same amount of time.

Existing convergence results for SDDP algorithms require that randomness is stagewise independent and enters only the right-hand sides of the constraints of the linear program at each stage (Philpott and Guan, 2008; Shapiro, 2011). Both assumptions are necessary if the linear program is only being solved for a subset of scenarios during the backward pass. Right-hand side randomness guarantees that the optimal dual solutions for scenarios in the subset are also dual feasible for all other scenarios, which significantly accelerates the generation of new hyperplanes. Stagewise independence, in turn, enables sharing hyperplanes among different scenarios at the previous stage, since the post-decision value function is identical for all scenarios. Algorithms that exploit these properties are in Higle and Sen (1991) and Chen and Powell (1999). Although in our model these assumptions are not fulfilled, the algorithm still converges almost surely. First of all, dual solutions are always feasible because the linear program is being solved for the entire set of scenarios during the backward pass. Second of all, hyperplanes are not shared among scenarios, since we can construct a separate post-decision value function for each scenario by using the transition matrix of the Markov process.

Proposition 4.2.3 *Denote π^ε as the policy obtained by ADDP for $\varepsilon > 0$ and π^* as the optimal policy of the relaxed problem. For a given initial reservoir level R_0, the policies obtained by ADDP for $\varepsilon = 0$ converge to the optimal policy in a finite number of steps. The values obtained from following π^ϵ are at most $\varepsilon(T-1)$ worse than the optimal values.*

Proof We first consider the case $\varepsilon = 0$. It follows from Lemma 1 in Philpott and Guan (2008), that for fixed S_t the functions

$$R \mapsto V_t(S_t, R) \tag{4.19}$$

are the pointwise maxima of finitely many linear functions, i.e., are piecewise linear for all $1 \leq t \leq T$. Note that each possible sequence of states $(S_1, \dots, S_T)$ has positive probability and therefore by the Borel-Cantelli Lemma occurs infinitely often in the forward pass. Since we add a hyperplane in each iteration, the finiteness of the set of hyperplanes implies that there exists an $\bar{n} \in \mathbb{N}$ such that no further hyperplanes are added after iteration $\bar{n}$. We denote the approximate value function after that state by $\hat{V}'_t$ for $1 \leq t \leq T$.

Suppose that the policy $\hat{\pi}$ found by using $(\hat{V}'_t)_{1 \leq t \leq T}$ is suboptimal in period $T-1$ and some $(S_1, \dots, S_T)$, i.e., $\max_\pi C(S_T, R_{T-1}, \pi) + \bar{V}_T(S_T, R_T) < \hat{V}'_T(S_T, R_{T-1})$. Since $(S_1, \dots, S_T)$ is sampled in iterations $n > \bar{n}$, the value function approximation would be updated in these iterations – a contradiction to the choice of $\bar{n}$. Hence, $\hat{V}'_T(S_T, R_{T-1})$ coincides with V'_T for all R that can be reached by $\hat{\pi}$ and for all $S_T \in \mathcal{S}_T$. The same holds for the post-decision value function $\hat{\hat{V}}'_{T-1}$. Having established the accuracy of $\bar{V}'_{T-1}$, we can inductively show the accuracy of all $\hat{V}'_t$ and $\hat{\hat{V}}'_t$ for all S_t and t. Hence, the solutions obtained with $\hat{V}'_t$ coincide with the optimal solutions of the relaxed problem.

The finite convergence property carries over to the case $\varepsilon > 0$. To proof the second part of the proposition, we begin with the last period T and note that by definition

$$\hat{V}'_T(S_T, R_{T-1}) - V'_T(S_T, R_{T-1}) \leq \varepsilon, \quad \forall\ S_T \in \mathcal{S}_T, \tag{4.20}$$

for all states of the system R_T that can be reached from R_0 by following π^ε. This inequality also holds for the respective post-decision value functions.

Let $\Delta \equiv C(S_{T-1}, R_{T-2}, \pi^*) - C(S_{T-1}, R_{T-2}, \pi^\varepsilon)$. Since π^ε is optimal for $\hat{V}'_T$ it follows that

$$\hat{\hat{V}}'_{T-1}(S_{T-1}, R_{T-1}, \pi^\varepsilon) \geq \hat{\hat{V}}'_{T-1}(S_{T-1}, R_{T-1}, \pi^*) + \Delta. \tag{4.21}$$

Therefore it holds for all $S_{T-1} \in \mathcal{S}_{T-1}$,

$$\begin{aligned}
0 \ &\leq\ C(S_{T-1}, R_{T-2}, \pi^*) + \bar{V}'_{T-1}(S_{T-1}, R_{T-1}(\pi^*)) \\
&-\ C(S_{T-1}, R_{T-2}, \pi^\varepsilon) - \bar{V}'_{T-1}(S_{T-1}, R_{T-1}(\pi^\varepsilon)) &\quad (4.22)\\
&=\ \Delta + \bar{V}'_{T-1}(S_{T-1}, R_{T-1}(\pi^*)) - \bar{V}'_{T-1}(S_{T-1}, R_{T-1}(\pi^\varepsilon)) &\quad (4.23)\\
&\leq\ \Delta + \bar{V}'_{T-1}(S_{T-1}, R_{T-1}(\pi^*)) - \hat{\hat{V}}'_{T-1}(S_{T-1}, R_{T-1}(\pi^\varepsilon)) + \varepsilon &\quad (4.24)\\
&\leq\ \Delta + \hat{\hat{V}}'_{T-1}(S_{T-1}, R_{T-1}(\pi^*)) - \hat{\hat{V}}'_{T-1}(S_{T-1}, R_{T-1}(\pi^\varepsilon)) + \varepsilon &\quad (4.25)\\
&\leq\ \varepsilon, &\quad (4.26)
\end{aligned}$$

where (4.24) follows from (4.21) and (4.25) from $\bar{V}'_{T-1} \leq \hat{\hat{V}}'_{T-1}$. Since $\hat{V}'_{T-1}$ is an ε-approximation of the function

$$R \mapsto \max_{\pi^\varepsilon} C(S_{T-1}, R, \pi^\varepsilon) + \hat{\hat{V}}'_{T-1}(S_{T-1}, R_{T-1}(\pi^\varepsilon)), \tag{4.27}$$

we have $\hat{V}'_{T-1}(S_{T-1}, R_{T-2}) \leq 2\varepsilon + V'_{T-1}(S_{T-1}, R_{T-2})$. Since the above holds for all $S_{T-1} \in \mathcal{S}_{T-1}$, the property carries over to the post-decision value function $\bar{V}'_{T-2}$ and the error bound follows by induction. $\square$

4.3 Econometric Model

Consistent with our model formulation, we propose an econometric model that separates randomness into a process of environmental variables with daily time increments and a process of electricity prices with hourly time increments. The objective of the econometric model is to accurately describe the dynamics of electricity prices and natural inflows by a small number of explanatory variables that fit into this modeling framework.

As with every commodity, the price of electricity is determined by supply and demand. In the short term, the supply is primarily driven by seasonal variations of intermittent power sources, such as wind, solar, and run-of-river, or by power plant outages. In particular, wind power generation drives down the electricity price, since wind power replaces other, more expensive, technologies. In the long term, it is mainly the price of oil that influences the price of primary energy and thereby the price for electricity. Electricity demand, on the other side, can be largely explained by temperatures and deterministic seasonal factors. The temperature affects electricity prices due to higher demand for heating and cooling. To verify this relationship, we ran a linear regression of the mean demand for electricity per day in Austria and Germany on the mean day temperature, the day length (i.e., the time from sunrise to sunset), as well as dummy variables for national holidays in Germany and Austria. Based on a sample of 1461 observations from 2007 to 2010, the linear model explains 89% of the variance in electricity demand. Accordingly, we model electricity prices dependent on those variables that influence supply and demand. In addition to seasonal variables, such as day length, weekday, and hour, explanatory variables include oil futures prices, the mean day temperature, and the total wind power generation per day.

To meet the requirements of a finite horizon Markov decision process, we decompose the dynamics of the environmental variables into a time-dependent trend and a state transition process which has the Markov property. Since oil prices are varying slowly and can be predicted well by their futures prices, at least in the short-term, we assume that the oil prices are known. Together with the seasonal variables, oil prices can then be viewed as deterministic and dependent on the day of the year. The state of the Markov process on day t is defined by the weekday (DAY), the mean day temperature (TEMP), the total wind power generation per day (WP), and the natural inflow (IN),

$$S_t = (\mathrm{DAY}_t, \mathrm{TEMP}_t, \mathrm{WP}_t, \mathrm{IN}_t). \tag{4.28}$$

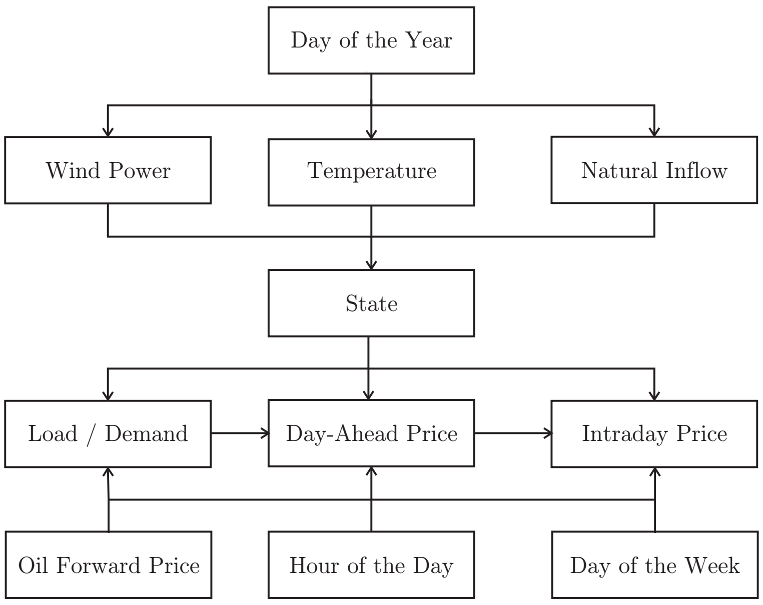


Figure 4.3: Econometric modeling framework

For a given realization of the state and a given day of the year, we model the hourly conditional expectations of the electricity prices, $\mathbb{E}(p_1, \ldots, p_{24}|S_t)$. An illustration of the econometric modeling framework is given in Figure 4.3.

For model estimation, we used hourly day-ahead and intraday spot prices from 2007 to 2010 as published by the European Energy Exchange (EEX). Analogous data on wind power forecasts is published by E.ON, EnBW, RWE, and Vattenfall for the four major German transmission zones. We used day-ahead wind forecasts instead of realized generation because spot prices are fixed one day in advance so that forecasts have a greater explanatory power than actual generation data. Since the price effect of temperature is a function of population density and local temperatures, we define the mean day temperature as a population weighted index over all Austrian and German cities with a population of more than 10,000. The index has been calculated using Mathematica 7 *CityData* and *WeatherData*. Data on natural inflows has been provided by an Austrian generating company for two glacier rivers that feed a hydro-storage system in the Alps (see Section 4.4 for further details). As both inflow patterns exhibit a correlation of $\rho = 0.8$, we aggregated these inflows into a single state variable.

4.3.1 State Transition Model

We decompose the state $(S_t)_{1 \le t \le T}$ into a deterministic trend component $(D_t)_{1 \le t \le T}$ and a random error $(E_t)_{1 \le t \le T}$ which follows a time-homogeneous Markov chain,

$$S_t = D_t + E_t, \quad t = 1, \ldots, T. \tag{4.29}$$

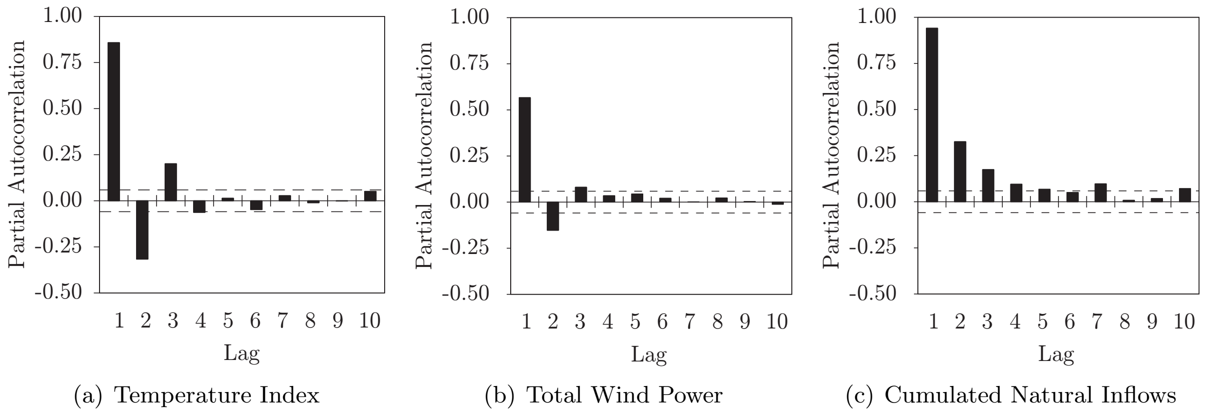


Figure 4.4: Partial Autocorrelation Functions of State Variables

To represent temperatures, we used the temperature index for the years 2007 to 2010 and estimated the parameters δ, α and u of the trigonometric regression model,

$$\text{TEMP}_t = \delta + \alpha \sin\left(\frac{365-t}{2\pi} - u\right) + \epsilon_{t1}. \tag{4.30}$$

The total wind power generation per day exhibits annual seasonality because of higher wind speeds during winter as well as an upward trend over time due to the ongoing installation of new wind power units. To capture these two trends, we include a quadratic term as well as an interaction term. We estimated the parameters δ_i, $i \in \{1,2,3\}$, α_1, α_2, u_1 and u_2 of the trigonometric regression model,

$$\text{WP}_t = \delta_1 + \delta_2 t + \delta_3 t^2 + \alpha_1 \sin\left(\frac{365-t}{2\pi} - u_1\right) + \alpha_2 \sin\left(\frac{365-t}{2\pi} - u_2\right) t + \epsilon_{t2}. \tag{4.31}$$

Based on 18 years of inflow data, we estimated the trend in natural inflows by simply computing the sample average over all 18 observations for each day. We model

$$\text{IN}_t = w_t \epsilon_{t3}, \tag{4.32}$$

where w_t is the mean inflow during day t. We obtain the residuals by dividing the inflow realizations by their respective means.

To estimate a model of E_t, we used the detrended state variables $\hat{e}_t = (\hat{\epsilon}_{t1}, \hat{\epsilon}_{t2}, \hat{\epsilon}_{t3})$, i.e., the residuals from (4.30) to (4.32) for 2007 to 2010. Figure 4.4 shows the partial autocorrelation functions of $\hat{e}_t$. The strong autocorrelation inherent in all state variables supports the hypothesis of stagewise dependence. By modeling the transition from one state to another as a Markov process, we capture autocorrelation up to the first lag. To ensure parsimony of the model, we do not include higher order lags.

Data	01-01-2007 to 31-12-2010
Observations for weekend models	415
Observations for working day models	1041
Average number of regressors	9.92
In-sample fit (R^2) for continuous state variables	70.33%
In-sample fit (R^2) for clustered state variables	69.73%

Table 4.1: Model summary of the day-ahead price model

We estimated the transition probabilities of the Markov chain for E_t by first fixing a number of states M and applying k-means clustering to organize the observations $(\hat{e}_t)_{t=1}^T$ into M clusters, $C_1, \ldots, C_M$. In a second step, we estimated the transition probabilities by counting the number of transitions between clusters as they occur in the sample. Accordingly, the transition probability matrix is given by

$$\mathbb{P}(E_{t+1} = C_j | E_t = C_i) = \frac{|\{t : \hat{e}_{t+1} \in C_j,\ \hat{e}_t \in C_i\}|}{|\hat{e}_t \in C_i|}, \quad \forall\, i, j. \tag{4.33}$$

For our implementation, we chose $M = 20$ to obtain the cluster centers $C_1, \ldots, C_{20}$.

4.3.2 State-Dependent Price Models

Day-ahead prices were modeled using linear models. We estimated one model for every hour and distinguish between working days and weekends, i.e., a total of 48 models. The regressors consist of all state variables, daily demand for electricity, day length, monthly oil future prices as well as three and six month lagged oil future prices. We included all interactions of the regressors up to the second order.

In order to ensure parsimony of the model, we performed stepwise combined forward-backward elimination as described in Draper and Smith (1998), Section 15.2. We used the Bayesian information criterion (BIC) for model selection. For the day-ahead price models, the selection routine chooses 9.92 regressors on average, but at most 19 out of 36 regressors. The number of regressors is reasonable, considering that we used around 415 observations for the weekend models and 1041 observations for the working day models.

The overall in-sample fit of the linear models for the day-ahead prices is R^2=70.33%, which is satisfactory, considering the fact that the time horizon includes the boom of 2007 and 2008 with high prices as well as the recession of 2009 and 2010 where exceptionally low prices were seen. When replacing the actual realizations of the state variables by their respective nearest cluster centers, the overall in-sample fit becomes R^2=69.73%. The model fit decreases slightly due to the loss of information induced by clustering. See Table 4.1.

To estimate the price response on the intraday market, we regressed the difference of intraday and day-ahead price on the hourly demand for electricity. The linear regression yielded an intercept of -20.3 (Euro) and a slope coefficient of 0.0011 (Euro per megawatt), with an in-sample fit of R^2=45.07%. The negative intercept reflects the fact that the true price response function is non-linear. Based on this data, we set the slope of the price-response function in the objective function of the stochastic program to $\beta = 0.0011$.

4.3.3 Simulation

To simulate price trajectories over one year, we began by sampling a state from the steady-state distribution of the Markov chain. The state transition process is simulated using the probabilities in (4.33). Based on the realization of the state variable, the oil futures price, and the day of the year, we generated hourly day-ahead prices using the linear models discussed above. Random noise was added by sampling the error term of location scale t-distributions fitted to the residuals of the linear models. The approach is supported by the Kolomogorov-Smirnov goodness of fit test, which does not reject the null hypothesis of a t-distribution in any of the linear models ($\alpha = 0.05$). Using the t-distribution yields heavy-tailed prices, as they are often observed in electricity markets.

To generate day-ahead price scenarios for the stochastic program, we resorted to *Latin Hypercube* sampling (LHS) as variance reduction technique (Shapiro, 2003). LHS generates scenarios which are more evenly distributed than pseudo-random scenarios. Denote $F_h^{-1}(S_t, U_l)$ as the inverse CDF (t-distribution) of the day-ahead price during hour h for a given S_t, and denote U_l as a uniform random variable. Then, we can generate K day-ahead prices using

$$p_{hs} = p'_{h\Theta(s)}, \quad p'_{hl} = F_h^{-1}(S_t, U_l) \quad \text{with} \quad U_l \sim U[(l-1)/K, l/K] \quad \forall \quad l \in \{1, \dots, K\}, \quad (4.34)$$

where Θ is defined as a mapping from s to l such that p_h is a random permutation of p'_h, i.e., we shuffled until the correlation between all p_h was below a threshold.

4.4 Numerical Results

To validate the efficiency of the proposed algorithm, we conducted a numerical analysis based on data from a generating company in Austria. The company operates a large hydro storage system in the Austrian Alps, which consists of an upper (1) and a lower reservoir (2). Both reservoirs are fed by natural inflows of two glacier rivers, and water can be pumped from the lower into the upper reservoir. In 2011, the system received a capacity upgrade which increased the pumping and generating capacities at the upper reservoir by factor five. System specifications before and after the upgrade for both reservoirs are given in Table 4.2.

For our numerical analyses, we sampled the Markov process as well as electricity price sce-

			Before Upgrade		After Upgrade	
Reservoir		j	1	2	1	2
Max reservoir content	1000m^3	UB_j^R	84 941	83 000		
Initial reservoir content	1000m^3	R_{0j}	34 333	37 000		
Average hourly inflow	1000m^3	$\bar{w}_j$	2.90	0.76		
Max pumping capacity	MW	UB_j^-	120	0	600	
Min pumping capacity	MW	LB_j^-	20	0		
Max generating capacity	MW	UB_j^+	112	220	592	220
Min generating capacity	MW	LB_j^+	15	10		
Pump efficiency	MW/1000m^3	η_j^-	4.23	0		
Turbine efficiency	MW/1000m^3	η_j^+	3.17	7.51		

Table 4.2: System specifications

narios from the econometric model described in Section 4.3. The full problem has $T = 365$ stages and $\mathcal{S}_t = \{S_1, \dots, S_{20}\}$ states at each stage. For each state, we generated $K = 20$ scenario paths, each containing $H = 24$ price realizations (see Section 4.3.3). The discount factor was set to $\gamma = 1.0$ and all bidding curves had four segments with $I = 3$ breakpoints.

The algorithm and the electricity price model were implemented in Java, using FICO Xpress BCL to model the mathematical programs. Computing times were measured on an Intel i7-2600 with 16 GB memory using the 64 bit versions of Java 6, Windows 7, and Xpress Optimizer 20.

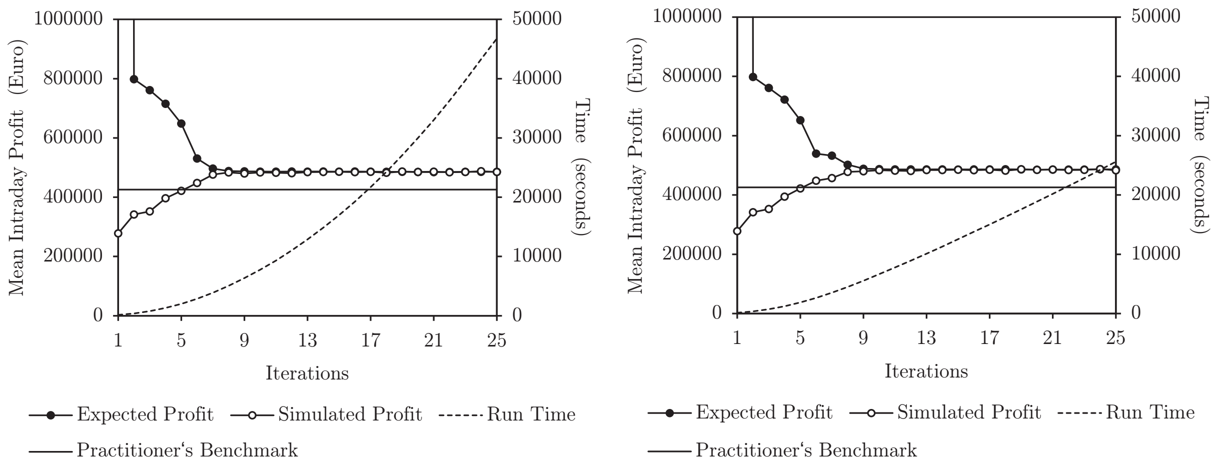
4.4.1 Computational Results

To verify convergence of the algorithm, we compared expected first-stage profits with realized profits. We ran ADDP for 25 iterations with both system configurations (before and after the upgrade) setting $\bar{V}_T \equiv 0$. To obtain the expected profits, after each iteration, we computed the objective values of the stochastic program at $T = 1$. Then, for 100 sample paths, we simulated the realized profits that result from executing bidding and operational decisions obtained by solving the stochastic program.

To obtain a *practitioner's benchmark*, we solved the deterministic counterpart of the problem as a single linear program using the weighted hourly price averages of the entire year. We then took the dual solutions associated with the reservoir balance equations at hours $h \in \{25, 49, 73, ...\}$ as salvage values in $t \in \{1, 2, 3, ...\}$ and again simulated the realized profits as above.

The convergence of expected and simulated profits for the upgraded system along with the computing times are shown in Figure 4.5 for $\varepsilon = 0$ and $\varepsilon = 10000$. The latter equals about 2.5% of the mean intraday profit. For the smaller system, the algorithm converged faster, but plots look alike and offer no additional insights.

We find that, despite using the relaxed problem to approximate the value function, the gap



(a) $\varepsilon = 0$ (b) $\varepsilon = 10000$

Figure 4.5: Convergence of ADDP for the system after the upgrade

between expected and simulated profits after 10 iterations was below 1%. Moreover, not adding hyperplanes that do not improve the approximation by more than $\varepsilon = 10000$ did not significantly increase the gap, which implies that the polyhedral approximation has no detrimental effect. Upon convergence, simulated profits from ADDP are 14% above the practitioner's benchmark, which reflects the value of having a stochastic solution to the problem. We can expect comparable results for problems with similar parameters, which includes a wide range of existing hydro storage systems.

Also, choosing $\varepsilon = 10000$ reduced the computational time by 15 percent after 10 iterations and by 45 percent after 25 iterations. The linear increase in computational time after 10 iterations for $\varepsilon = 10000$ indicates that the algorithm has converged and that no further hyperplanes are being added. This observation corresponds to the convergence of expected and simulated profit. Since it is easy to track how many hyperplanes have been added during a backward pass, one could use this information to stop the algorithm automatically, which would be an alternative to the classic stopping criterion proposed in Pereira and Pinto (1991).

4.4.2 Structural Insights

In addition to studying the behaviour of the algorithm, we were interested in how the capacity upgrade influences intraday profits as well as the decision policy. To avoid that reservoirs are being emptied at the end of the year, the value function of the final stage $\bar{V}_T$ is such that it sufficiently penalizes any reservoir content below the initial reservoir levels.

The profitability of investments in storage in the current market environment is high. Our results indicate that the system upgrade increases profits from €335,000 to €485,000 per day. Assuming 8.0% cost of capital and a stationary price path, the investment will have earned its

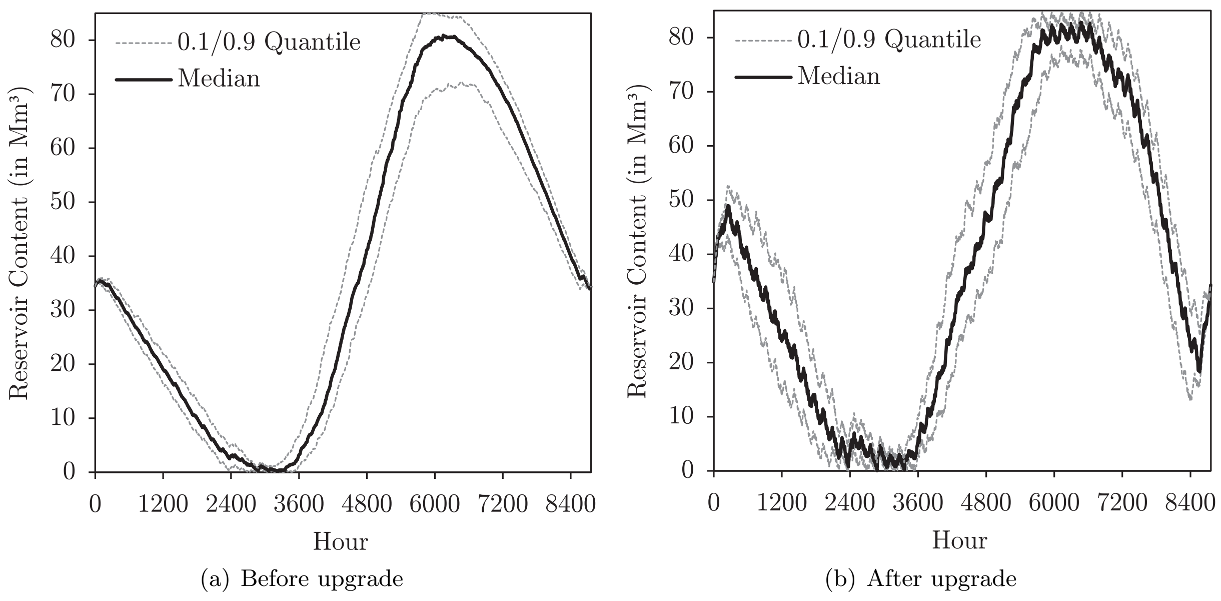


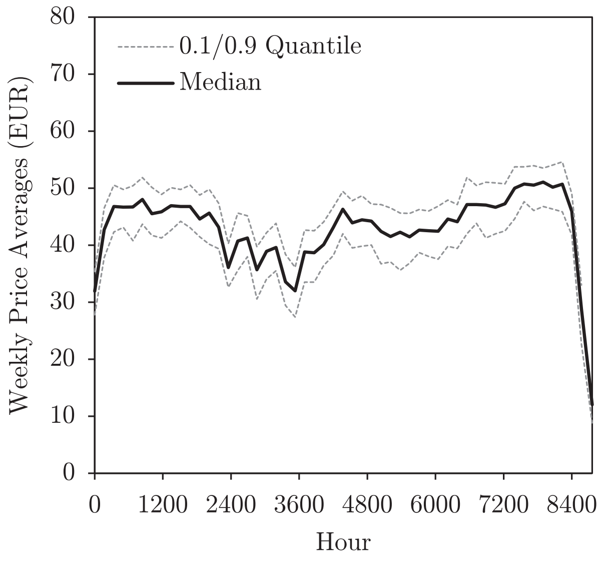
Figure 4.6: Reservoir content curves

investment cost of €405M after 11 years. However, the marginal value of power capacity appears to be decreasing, since an increase of turbine and pump capacities by factor five increases average profits by merely 45%.

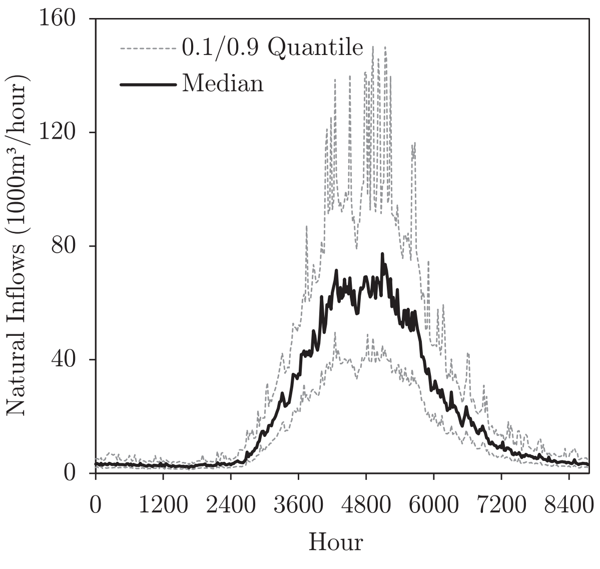
Figure 4.6 shows the reservoir contents of the upper reservoir over the course of the year for the system before (a) and after the upgrade (b). Although we used 100 different sample paths, both reservoir content curves exhibit little variation in between the sample paths. Nevertheless, the capacity upgrade increases the variability of the reservoir content, since less time is needed for large changes in reservoir contents.

Figure 4.7 shows the distribution of electricity prices (a) and natural inflows into the upper reservoir (b) over one year. By taking weekly price averages, we level the price effects of the individual hours as well as the effect of working days versus weekends. Before the upgrade, the optimal decision policy has been largely governed by long-term effects. In winter, when inflows were low but average prices high, the upper reservoir was being emptied, while in summer when average prices were low, but inflows high, the reservoir was being filled up again. By contrast, after the upgrade, the optimal decision policy now additionally takes advantage of low electricity prices around the winter holidays. Since there are virtually no natural inflows during this time of year, water is being pumped into the upper reservoir, so that more electricity can be sold during the more profitable weeks in late January and early December.

Figure 4.8 shows two plots of the post-decision value function at stage $T = 1$ for the system before and after the upgrade. Both functions remain linear over a wide range of reservoir states. We conjecture that this observation is a general feature of the problem, which also explains the



(a) Weekly price averages



(b) Daily inflows into the upper reservoir

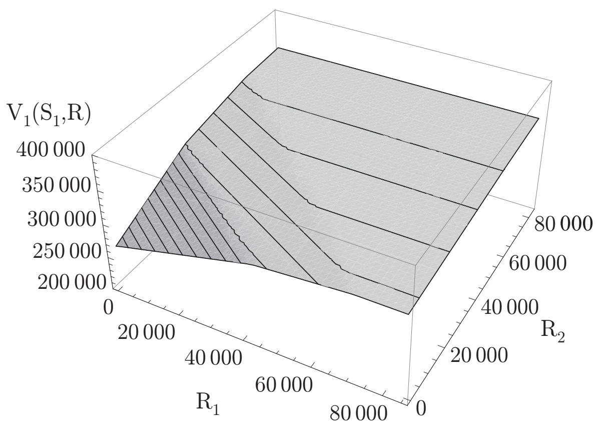
Figure 4.7: Stochastic processes

fast convergence of the algorithm.

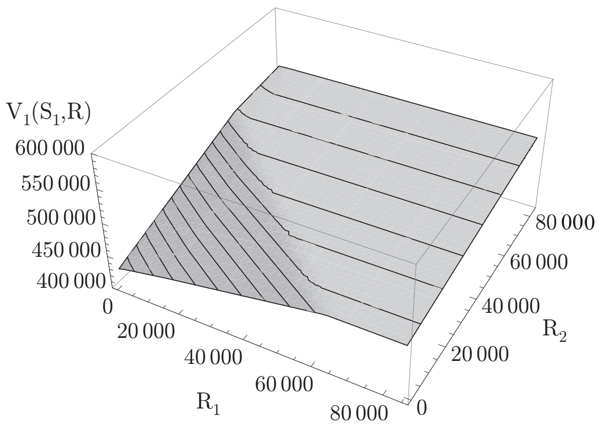
4.5 Discussion

We modeled the bidding problem of a generating company that operates a network of hydro storage plants as a multi-stage stochastic program and proposed a solution strategy that integrates stochastic dual dynamic programming with ideas from approximate dynamic programming. We divided the annual planning horizon into daily stages with hourly bidding decisions as part of the intrastage bidding problem. Accordingly, we separated intrastage from interstage randomness, which enabled us to model price uncertainty at each stage dependent on a state variable that evolves over time following a Markov process. To solve the multi-stage decision problem, we proposed a solution strategy that computes an approximation of the value function of the interstage process. The algorithm, referred to as approximate dual dynamic programming (ADDP), relaxes the original problem formulation and constructs a polyhedral approximation of the value function. This approximation can be used inside the original, more complicated intraday bidding problem to derive near-optimal bidding decisions.

We showed that the algorithm converges and derived an error bound of the polyhedral approximation. Tailored to the modeling framework, we developed an econometric model of electricity prices and stochastic inflows fitted to data from the EEX wholesale electricity market as well as actual inflow data. We then carried out an extensive case study based on a hydro storage system in Austria. We find that approximating the continuous state transition process



(a) Before upgrade



(b) After upgrade

Figure 4.8: Post-decision value functions of the first stage

by a discrete-state Markov chain provides a good model fit. Numerical results indicate that the algorithm converges to a near-optimal solution, despite using a relaxed version of the original problem to approximate the value function of the interstage problem. Moreover, using a polyhedral approximation to accelerate the algorithm had no noticeable detrimental effect on solution quality.

Future work should focus on models that additionally consider the market for reserve electricity. It would also be interesting to find out whether the concave value function could be approximated well by a linear function.

Chapter 5

Simulation Optimization for the Stochastic Economic Lot Scheduling Problem

Lot-sizing and scheduling are classical problems of production planning, with particularly many applications in the process industry. Most researchers treat this problem as a deterministic optimization problem, since this task is usually seen as short term and operational. Although this assumption is reasonable in some production environments, there are many applications where demand uncertainty requires integrating lot-sizing and scheduling with safety stock planning. Besides finding the optimal production sequence and respective lot sizes, production planning needs to provide the right amount of flexibility in response to uncertainty.

We address the problem of scheduling production of multiple products on a single machine with significant setup times under uncertain demand in continuous time. In the literature, this problem is known as the *stochastic economic lot scheduling problem* (SELSP). The SELSP is a computationally complex problem, where the deterministic counterpart, the economic lot scheduling problem (ELSP), is already NP hard (Hsu, 1983). For literature reviews on the ELSP, we refer to Elmaghraby (1978) and Davis (1990). A comprehensive literature review on stochastic lot scheduling problems with a focus on modeling and solution methods is provided by Sox et al. (1999). Winands et al. (2011) review and classify the literature on the SELSP according to sequencing and lot-sizing decisions and include several practical applications.

In general, the SELSP can be formulated as a semi-Markov decision process (SMDP) (Graves, 1980; Qiu and Loulou, 1995), but since this formulation suffers from the curse of dimensionality only small problem instances can be solved to optimality. Most research is therefore dedicated towards simpler policies. Gallego (1990) and Bourland and Yano (1994) both propose procedures where a production plan is set in advance and then a policy is used that restores the plan in response to uncertain demand. For Poisson demands, Federgruen and Katalan (1996) propose analytical solutions to find optimal base-stock levels and idle times for a given sequence. In Federgruen and Katalan (1998), the authors derive the optimal relative frequencies for each product, from which a fixed sequence can be constructed. For more general renewal processes, Anupindi and Tayur (1998) use infinitesimal perturbation analysis to find optimal base-stock levels for a given production sequence, and Markowitz et al. (2000) propose optimal control policies for pure rotation cycles using heavy-traffic approximation. Other approaches are in Krieg and Kuhn (2002), Wagner and Smits (2004), and Brander and Forsberg (2006). Although a fixed sequence is often a good choice, the optimal sequence is likely to be dynamic and has to take the entire vector of inventory states into account. For products with identical parameters, Vaughan (2007) finds that a dynamic sequence resulting from order-point methods outperforms a fixed cyclical schedule in systems with a large number of products and low utilization. Graves (1980) and Gascon et al. (1994) compare several heuristic scheduling rules where the sequencing decision is determined by a product's number of periods of supply. Altiok and Shiue (1994, 1995) derive optimal (s,S) policies for dynamic sequences and Poisson demands by analyzing the underlying Markov chain. Paternina-Arboleda and Das (2005) use a two-level approach of first searching for optimal base-stock levels and then using reinforcement learning to optimize the sequencing decisions.

Although much progress has been seen in simulation optimization, only few authors discuss approximations to optimize the SMDP (Paternina-Arboleda and Das, 2005) or propose black box algorithms to optimize control policies (Anupindi and Tayur, 1998). Our contribution is to close this gap by proposing two different simulation optimization approaches. First, as a methodology to address the curse of dimensionality, approximate dynamic programming (ADP) has received considerable attention (Powell, 2007). ADP uses Monte Carlo simulation to approximate the state-dependent value function of a dynamic program, avoiding a complete enumeration of the state space. We propose two approximate value functions based on linear combinations of piecewise-constant functions and then use an ADP algorithm to find the weights of these functions. Second, even for simple control policies closed-form solutions are complex, so that finding the right parameters is computationally challenging. Global optimizers therefore present a promising alternative. We propose representations of simple base-stock policies amenable to unconstrained global optimization for cyclic as well as dynamic production sequences. To search for the optimal parameters of these policies, we resort to the *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) algorithm (Hansen and Ostermeier, 2001).

To study the SELSP as well as the various solution methods, we performed a large-scale simulation study to answer the following questions. Which simulation optimization method produces the best policy on average and how often? How much worse is a policy if it is not best? Under which circumstances is a particular policy better than another? How far from

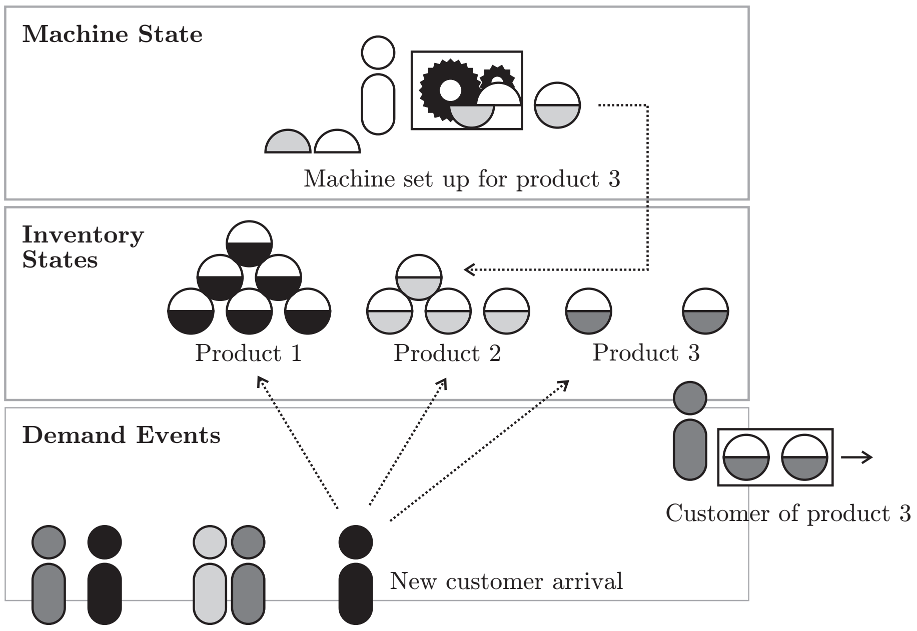


Figure 5.1: Illustrative example of the stochastic economic lot scheduling problem

optimal is the best policy found by simulation optimization? What is the influence of model parameters on average cost?

5.1 Model Formulation

5.1.1 Assumptions

We consider the continuous-time stochastic economic lot scheduling problem with $n \in \{1, \dots, N\}$ products. There is a single machine that can only manufacture one product at a time. If the machine state changes from idle to busy or from one product to another, the machine has to be set up. This requires a deterministic, sequence-independent setup time W_n and incurs a setup cost of A_n. We further assume that the setup status cannot be preserved over an idle period. The production for one unit of product n requires a deterministic production time p_n. During a setup or production of a single item, interruption is not permitted. Inventories are subject to holding cost h_n per item and unit of time and cannot exceed a maximum inventory level $\bar{y}_n$. Demand for each product n follows a compound renewal process with interarrival distribution F_n^A and demand size distribution F_n^D. Interarrival times and demand size are independent for each product and across products. Unsatisfied customer demand is lost at cost v_n per item, but we allow for partial fulfillment of an order. More specifically, we model the compound renewal process as *stuttering* Poisson process, where arrivals follow a Poisson process and the demand size per arrival follows a geometric distribution. Denote λ_n as the arrival rate of demand for product n and $C_n(t)$ as the number of arrivals of product n during a time interval of length t. Then, the probability for the number of arrivals being equal to k is Poisson distributed and given by

$$P_n^A(C_n(t) = k) = \frac{(\lambda_n t)^k}{k!} \exp(-\lambda_n t),\ k \in \mathbb{N}_+, \tag{5.1}$$

with mean number of arrivals $\lambda_n t$. Denote $0 < q_n \leq 1$ as the probability of the demand size D_n per arrival being equal to 1. The probability of the demand size being equal to d is given by

$$P_n^D(D_n = d) = q_n(1 - q_n)^{d-1},\ d \in \mathbb{N}_+, \tag{5.2}$$

with mean demand size $1/q_n$. In general, compound Poisson demand over a time interval of length t is defined as

$$P(D_n(t) = d) = \sum_{i=0}^{d} \frac{(\lambda t)^i}{i!} \exp(-\lambda t) f^d(i), \tag{5.3}$$

where $f^d(i)$ denotes the probability that d demands occur on i demand occasions. Recursive computation of stuttering Poisson demands (Feeney and Sherbrooke, 1966), where $C_n(t) = k$ customers with total demand $D_n(t) = d$ arrive during time period t, is given by

$$P(D_n(t) = d) \quad = \quad \sum_{k=0}^{d} f_n(d, k), \tag{5.4}$$

with $f_n(d, k) = (1 - q_n) f_n(d - 1, k) + q_n \frac{\lambda_n t}{k} f_n(d - 1, k - 1)$, $f_n(0, 0) = \exp(-\lambda_n t)$, $f_n(d, k) = 0$ if $d < k$, and $f_n(d, 0) = 0$ if $d > 0$.

For a given period demand with mean μ_n and standard deviation σ_n, the required stuttering Poisson parameters to obtain the same first two moments are $q_n = 2\mu_n(\mu_n + \sigma_n^2)^{-1}$ and $\lambda_n = \mu_n q_n$. Note that P_n^D is only defined if $q_n \leq 1$ so that feasible mean-variance combinations are limited to $\mu_n \leq \sigma_n^2$.

5.1.2 Semi-Markov Decision Process

We model the SELSP under compound Poisson demand as an infinite horizon, average cost semi-Markov decision process (SMDP). In an SMDP, decisions are only made after a change of the system state which is relevant for the decision-making process. During such a decision epoch, the system state may change several times due to multiple demand arrivals, but no decision can be made.

We describe the state of the production system by the vector $S = (m, y) \in \mathcal{S}$, where y denotes the inventory state $y = (y_1, ..., y_N)$, with $0 \leq y_n \leq \bar{y}_n$ and m the machine state, with $m \in \{0, ..., N\}$. Machine status $m = 0$ indicates that the machine is idle and $m = n > 0$ that the machine is currently set up for product n. We assume that sojourn times are finite and a finite number of transitions takes place during a (finite) decision epoch. Denote $x \in \mathcal{X} = \{0, ..., N\}$ as a production decision. A new decision epoch begins after production or setup of an item has been completed, or, in case the machine has been idle, upon arrival of new customer demand.

To solve the infinite horizon SMDP, we have to find an optimal policy π^* such that the average cost per unit of time g is minimized. Let g^* denote the minimal expected average cost and $V^*(S)$ the minimal average cost (or value) when being in state S. Denote $\mathbb{P}(S'|S,x)$ as the probability of a transition from state S to successor state S' when decision x is taken; and denote $\bar{\tau}(S,x)$ as the average time and $c(S,x)$ as the total cost associated with state S and decision x prior to transition to S'. Then, the optimization problem is to find g^* and $V^*(S)$, such that

$$V^*(S) = \min_{x \in \mathcal{U}} \left\{ c(S,x) - g^* \bar{\tau}(S,x) + \sum_{S' \in \mathcal{S}} \mathbb{P}(S'|S,x) V^*(S') \right\} \quad \forall\ S \in \mathcal{S}. \tag{5.5}$$

For the problem described, the Markov chain of every stationary policy has a unichain transition matrix, so that the expected average cost do not vary with the initial state.

The probability of a transition from state S to S' after decision x is given by

$$\mathbb{P}(S'|S,x) = \prod_{n=1}^{N} P_n(S'|S,x). \tag{5.6}$$

The product-specific transition probabilities $P_n(S'|S,x)$ are defined as

$$P_n(S'|S,x) = \begin{cases} \lambda_n P\big(D_n = y_n - y_n'\big)\big(\sum_{m=1}^{N} \lambda_m\big)^{-1} & \text{if } x = 0 \text{ and } y_n' > 0, \\ \lambda_n P\big(D_n \geq y_n\big)\big(\sum_{m=1}^{N} \lambda_m\big)^{-1} & \text{if } x = 0 \text{ and } y_n' = 0, \\ P(D_n(p_n) = y_x - y_x' + 1) & \text{if } n = x,\ x = m \text{ and } y_n' > 1, \\ P(D_n(p_n) \geq y_x) & \text{if } n = x,\ x = m \text{ and } y_n' = 1, \\ P(D_n(p_x) = y_n - y_n') & \text{if } n \neq x,\ x = m \text{ and } y_n' > 0, \\ P(D_n(p_x) \geq y_n) & \text{if } n \neq x,\ x = m \text{ and } y_n' = 0, \\ P(D_n(W_x) = y_n - y_n') & \text{if } x \neq m \text{ and } y_n' > 0, \\ P(D_n(W_x) \geq y_n) & \text{if } x \neq m \text{ and } y_n' = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

If the machine goes idle ($x = 0$), we multiply the probability of the next event being demand for product n with the probability of demand being either of size equal to $y_n - y_n'$ for $y_n' > 0$ or of size greater than y_n for $y_n' = 0$. If the machine is set up to produce an item other than n ($x = m = n$), demand over p_n periods either depletes inventory from y_n to $y_n' > 1$ or to $y_n' = 1$, so that $y'_n \geq 1$ always includes the previously produced item. If the machine is already set up but produces another product ($x = m \neq n$), demand over p_x periods either depletes inventory from y_n to $y'_n > 0$ or to zero. If the machine has to be set up ($x \neq m$), demand over W_x periods either depletes inventory from y_n to $y'_n > 0$ or to zero.

The average sojourn time $\bar{\tau}(S, x)$ of a decision epoch is given by

$$\bar{\tau}(m, y, x) = \begin{cases} p_x & \text{if } x = m, \\ W_x & \text{if } x \neq m, \\ \left(\sum_{n=1}^{N} \lambda_n\right)^{-1} & \text{if } x = 0. \end{cases} \tag{5.8}$$

Note that in case the machine goes idle, the average sojourn time until the next demand event is a Poisson process with rate $\sum_{n=1}^{N} \lambda_n$.

The cost $c(S, x)$ of being in state S and taking decision x consists of setup or variable manufacturing costs, inventory holding costs, and lost sales penalty costs.

$$c(S, x) = \begin{cases} c_x + \sum_{n=1}^{N} \left(H_n(y_n, p_x) + v_n \sum_{d=y_n+1}^{\infty} (d - y_n) P(D_n(p_x) = d)\right) & \text{if } x = m \\ A_x + \sum_{n=1}^{N} \left(H_n(y_n, W_x) + v_n \sum_{d=y_n+1}^{\infty} ((d - y_n) P(D_n(W_x) = d)\right) & \text{if } x \neq m, \\ \sum_{n=1}^{N} \left(h_n y_n + v_n \lambda_n \sum_{d=y_n+1}^{\infty} (d - y_n) P_n^D(d)\right) \left(\sum_{i=1}^{N} \lambda_i\right)^{-1} & \text{if } x = 0. \end{cases} \tag{5.9}$$

The determination of product-specific and time-dependent expected holding cost $H_n(y_n, t)$ requires to track each of the y_n items. Demand for item k occurs on the l-th demand event if $D_n^{(l-1)} < k$ and $D_n^{(l)} \geq k$ where $D_n^{(l)}$ is the cumulative demand for product n over l transactions and $F_n^{(l)}$ denotes the l-fold convolution of the demand size distribution

$$\begin{aligned} \theta_n(k, l) &= P(D_n^{(l-1)} < k \wedge D_n^{(l)} \geq k) = \sum_{i=1}^{k-1} P(D_n^{(l-1)} = i) P(D_n \geq k - i) \\ &= \sum_{i=1}^{k-1} f_n^{(l-1)}(i)(1 - F_n^D(k - i - 1)) = F_n^{(l-1)}(k - 1) - F_n^{(l)}(k - 1), \end{aligned}$$

with $F_n^{(l)}(1) = 0$, $F_n^{(0)} = 1$. Note that $k \geq 1$ and for $l = 1$, $P(D_n \geq k) = 1 - F_n^{(l)}(k - 1)$. The convolution is given by

$$F_n^{(l)}(k) = \sum_{h=1}^{k-1} F_n^{(l-1)}(h) f_n^D(k - h). \tag{5.10}$$

The time until the l-th demand is Erlang distributed with parameters λ_n and l. Then, for a given inventory level y_n, the expected holding costs over a time interval of length t are given by

$$\begin{aligned} H_n(y_n,t) &= h_n \sum_{k=1}^{y_n}\sum_{l=1}^{k} P(D_n(l-1) < k \wedge D_n(l) \geq k) \left(\int_0^t uE_{n,l}(x)du + t\int_t^{\infty} E_{n,l}(u)du\right) \\ &= h_n \sum_{k=1}^{y_n}\sum_{l=1}^{k} (F_n^{(l-1)}(k-1) - F_n^{(l)}(k-1)) \times \\ &\quad \left(\frac{l}{\lambda_n}(1 - P_n^A(C_n(t) \leq l)) + t \cdot P_n^A(C_n(t) \leq l-1)\right). \end{aligned} \tag{5.11}$$

5.2 Solution Methods

5.2.1 Relative Value Iteration

The solution to the system of equations given in (5.5) is unique w.r.t. g^* and unique up to an additive constant for all $V(S)$. Therefore, we can set the value of an arbitrary state S_0 to $V(S_0) = 0$ and express the other values relative to this value. The optimal gain and the values of each state can be successively approximated by the following recursive scheme where g^j and $V^j(S)$ denote the respective values obtained in iteration number j.

$$g^j = \min_{x \in \mathcal{X}} \left\{ \frac{c(S_0,x) + \sum_{S \in \mathcal{S} \setminus S_0} P(S|S_0,x) \cdot V^j(S)}{\bar{\tau}(S_0,x)} \right\} \tag{5.12}$$

$$V^{j+1}(S) = V^j(S) - g^j + \beta \cdot \min_{x \in \mathcal{X}} \left\{ \frac{c(S,x) + \sum_{S' \in \mathcal{S} \setminus S_0} P(S'|S,x) \cdot V^j(S') - V^j(S)}{\bar{\tau}(S,x)} \right\} \tag{5.13}$$

for $S \in \mathcal{S} \setminus S_0$ and with $\beta < \bar{\tau}(S,x)\ \forall\ S \in \mathcal{S},\ x \in \mathcal{X}$ to ensure numerical stability. In addition, this successive improvement scheme provides a lower and an upper bound on the optimal gain in each iteration (Schweitzer, 1971).

5.2.2 Approximate Value Iteration

Since relative value iteration is subject to the curse of dimensionality for larger problems, we propose to use approximate value iteration instead (see Figure 5.2). For a given approximate value function $\bar{V}(\,\cdot\,; w^0)$ with initial parameters w_0 and a starting state S, the algorithm simulates a sample path of T decision epochs while updating the control policy *online*. The main loop consists of three steps: (Step 2.1) a (greedy) control policy selects the best decision based on the value estimate of the previous iteration for the given state of the system; (Step 2.2) a simulation model S^M samples the state transition function and returns a realization of the immediate cost

```
(1) Input arguments: value function $\bar{V}(\,\cdot\,; w^0)$, starting state $S$
(2) Do for $t = 1, 2, \ldots, T$
    (2.1) Solve $x^* \leftarrow \arg\min_{x \in \mathcal{X}} \bar{V}\big(S, x; w^{t-1}\big)$
    (2.2) Compute $\big(c_t, \tau_t, S'\big) \leftarrow S^M\big(S, x^*\big)$
          $r \leftarrow c + \exp(-\gamma\tau) \min_{x \in \mathcal{X}} \bar{V}\big(S', x; w^{t-1}\big)$
    (2.3) Update $w_t = U^V\big(w^{t-1}, S, x^*, r\big),\ S \leftarrow S'$
(3) Return value function $\bar{V}(\,\cdot\,; w^T)$
```

Figure 5.2: Approximate value iteration for semi-Markov decision processes

c, the sojourn time τ and the successor state S', which gives the discounted reward,

$$r = c + \exp(-\gamma\tau) \min_{x \in \mathcal{X}} \bar{V}\big(S', x; w_{t-1}\big); \tag{5.14}$$

(Step 2.3) a function U^V updates the value estimate of making the greedy decision x^* in state S. After T decision epochs, the algorithm returns the final estimate of the value function approximation.

For approximate value iteration, we use discounted reward as a proxy for average reward. We find this formulation to be more stable than approximating the average reward directly. The discount factor γ can therefore be regarded as a purely algorithmic parameter which has to be set sufficiently large to obtain a nearly average cost optimal policy without risking numerical stability.

Value Function Approximation by Stochastic Gradients

The updating function U^V is based on stochastic gradient algorithms, a popular class of methods for function approximation which are particularly well-suited for approximate value iteration (Bertsekas, 2007; Powell, 2007). In contrast to (non-)linear regression methods, stochastic gradient algorithms have only $\mathcal{O}(n)$ time complexity and are able to estimate the mean of a random variable *online* while new samples are being collected.

A stochastic gradient algorithm changes the parameters of a function approximator to fit the observations from a data set. Let the approximate value function $\bar{V}$ be a linear combination of basis functions ϕ_i with real-valued weights w_i and $i \in \{1, 2, ..., D\}$. A basis function ϕ_i may be any non-linear function of S and x. Denote w and Φ as the corresponding vectors of length

D, with $w^\top$ as transpose. Then, the approximate value function is given by

$$\bar{V}(S,x;w) = w^\top \Phi(S,x) = \sum_{i=1}^{D} w_i \phi_i(S,x). \tag{5.15}$$

A stochastic gradient algorithm adjusts the weight vector w after each observation in the direction that minimizes the *mean squared error*, $\min_w \frac{1}{2}\big(\bar{V}(S,x;w) - r\big)^2$. For a linear function, the *stochastic* sample gradient with respect to w is given by $\Phi(S,x)$. Since the true gradient is unknown, we adjust the weight vector in the direction of the gradient only by a small amount $\alpha^t \in (0,1]$, referred to as stepsize. The function U^V that updates the weight vector is then given by

$$w^t = U^V\big(w^{t-1}, S, x, r\big) = w^{t-1} + \alpha^t\big(r - \bar{V}(S,x;w^{t-1})\big)\Phi(S,x). \tag{5.16}$$

For a stationary policy, the weight vector w is guaranteed to converge to a local optimum as long as we gradually reduce the stepsize to zero (Bertsekas, 2007, p. 333). Practical convergence is thereby largely affected by choosing the right stepsize for each updating step. Although the optimal stepsize schedule is unknown, experimental work has shown that there exist simple stepsize rules that work well in practice (Powell, 2007, Ch. 6). One of these rules is the *generalized harmonic* stepsize, which is given by $\alpha^t = ab(a+t-1)^{-1}$, with $a \in \mathbb{R}^+$ and $b \in (0,1]$ as scaling parameters.

Piecewise-Constant Value Functions

We use a linear combination of piecewise-constant functions as approximation scheme and apply the stochastic gradient algorithm to update the weights of the constant function segments. We assign two separate functions to each production decision: one function for the case when the machine state is changed after a decision and one function for the case when the machine state remains the same. This separation takes the different sojourn times during setup or production of an item into account. We then model each of these functions as a linear combination of piecewise-constant basis functions of the inventory states.

Denote $\bar{v}_j$ as the *partial* value function, with $j \in \{0, ..., 2N\}$. The piecewise-constant function that assigns two partial value functions to each decision is given by

$$\bar{V}(S,x;w) = \begin{cases} \bar{v}_x(y;w) & \text{if } x \neq m \text{ or } x = 0, \\ \bar{v}_{N+x}(y;w) & \text{otherwise.} \end{cases} \tag{5.17}$$

Note that for the decision to go idle, $x = 0$, we use only one function, since the sojourn time for an idle epoch is independent of the machine state.

For the partial value functions $\bar{v}_j$, we test two different approximation schemes. The first approximation is the sum over N piecewise-constant functions of each inventory state variable. The intuition is that the expected immediate cost can be computed separately for each product, since expected holding and lost sales cost of one product are independent of other products. This makes the immediate cost function separable in the inventory state variables y_n. The separable (first-order) partial value function is then given by

$$\bar{v}_j(y; w) = \sum_{n=1}^{N} \bar{v}_{jn}^{(1)}(y_n; w). \tag{5.18}$$

Each function $\bar{v}_{jn}^{(1)}$ is a piecewise-constant function with K disjoint intervals of width $d_n^{(1)} = \frac{\bar{y}_n+1}{K} : n \in \{1, ..., N\}$,

$$\bar{v}_{jn}^{(1)}(y_n; w) = \sum_{k=1}^{K} w_{I(j,n,k)} \cdot \phi_{I(j,n,k)}(y_n) = \sum_{k=1}^{K} w_{I(j,n,k)} \cdot \mathbb{1}[(k-1)d_n^{(1)}, kd_n^{(1)})(y_n), \tag{5.19}$$

where $I(\cdot)$ maps the multi-dimensional index to a unique index of an element of the weight vector. Note that by setting $d_n^{(1)} = \frac{\bar{y}_n+1}{K}$, the right-open interval $[(K-1)d_n^{(1)}, Kd_n^{(1)})$ contains the maximum inventory level for $K \leq \bar{y}_n$.

If we take into account that the approximate value function is not only an estimate of the immediate cost, but also of the discounted value of future states and decisions, then we cannot assume separability any more. To consider dependencies among inventory state variables, we propose a second approximation scheme, where we add the sum of piecewise-constant functions over all $\binom{N}{2}$ combinations of inventory state variables to the first approximation. The (second-order) partial value function is then given by

$$\bar{v}_j(y; w) = \sum_{n=1}^{N} \Big(\bar{v}_{jn}^{(1)}(y_n; w) + \sum_{m=n+1}^{N} \bar{v}_{jnm}^{(2)}(y_n, y_m; w) \Big). \tag{5.20}$$

Each function $\bar{v}_{jnm}^{(2)}$ is a piecewise-constant function with two arguments and $L \times L$ disjoint segments with edge lengths of $d_n^{(2)} = \frac{\bar{y}_n+1}{L}$ and $d_m^{(2)} = \frac{\bar{y}_m+1}{L} : n, m \in \{1, ..., N\}$, so that

$$\begin{aligned}\bar{v}_{jnm}^{(2)}(y_n, y_m; w) =& \sum_{k=1}^{L} \sum_{l=1}^{L} w_{I'(j,n,m,k,l)} \cdot \phi_{I'(j,n,m,k,l)}(y_n, y_m), \\ =& \sum_{k=1}^{L} \sum_{l=1}^{L} w_{I'(j,n,m,k,l)} \cdot \mathbb{1}[(k-1)d_n^{(2)}, kd_n^{(2)})(y_n) \cdot \mathbb{1}[(l-1)d_m^{(2)}, ld_m^{(2)})(y_m),\end{aligned} \tag{5.21}$$

with $I'(\cdot)$ as another index function.

```
(1) Input arguments: initial guess $\mu^u$, trust region $\sigma^u$
(2) Do for $i = 1, 2, \ldots, I$
    (2.1) Get $(u^1, ..., u^K) \leftarrow G^M(K)$ from internal model
    (2.2) Do for $k = 1, 2, \ldots, K$
        (2.2.1) Do for $t = 1, 2, \ldots, T$
            (2.2.1.1) Compute $\big(c_t, \tau_t, S\big) \leftarrow S^M\big(S, \pi(S; u^k)\big)$
        (2.2.2) Compute $r^k \leftarrow \sum_{t=e+1}^{T} c_t \left(\sum_{t=1}^{T} \tau_t\right)^{-1}$
    (2.3) Update internal model $U^M\big((u^1, ..., u^K), (r^1, ..., r^K)\big)$
(3) Return best solution $u^*$
```

Figure 5.3: Generic policy search for production control

Let us briefly review the space complexity of the approximate value functions. The first approximation with a weight vector of length $D = \big(2N + 1\big)KN$ has a worst-case space complexity of $\mathcal{O}(KN^2)$, while the second approximation with $D = \big(2N + 1\big)\big(KN + L^2\frac{N(N-1)}{2}\big)$ has $\mathcal{O}(N^3L^2)$. Both schemes therefore have only polynomial space complexity, which is low compared to laying a coarse grid over the state space. A grid where each inventory state is aggregated into C intervals would produce a weight vector of length $D = \big(2N + 1\big)C^N$ which has an exponential worst-case complexity of $\mathcal{O}(NC^N)$ and would thus be itself subject to the curse of dimensionality.

5.2.3 Direct Policy Search

An alternative to approximating the value function and using this function to control the decision process is to directly search for optimal parameters of simpler control policies. To guide the search for the optimal parameter vector, we resort to the CMA-ES algorithm (Hansen and Ostermeier, 2001). CMA-ES generates new candidate vectors from a multivariate normal distribution, i.e., $\mathcal{N}\big(\mu^u, \mathrm{diag}(\sigma^u)\big)$, which serves as an internal model of promising search steps, where dependencies among parameters are described by the covariance matrix. Throughout the search process, the algorithm updates the distribution's mean vector and its covariance matrix to increase the likelihood of previously successful search steps.

Figure 5.3 outlines a generic formulation of the CMA-ES algorithm. Denote $\pi(\,\cdot\,; u)$ as control policy which is characterized by a (continuous) parameter vector u. The objective is to search for an u that minimizes the expected average cost. The algorithm is initialized with a *guess* of the best solution, μ^u, as well as a *trust* region, σ^u, in which the solution is likely to be found. The main loop consists of three steps: (Step 2.1) the algorithm generates a set of K candidate solutions $(u^1, ..., u^K)$ from the internal model G^M which controls the search process; (Step 2.2) for each of the resulting control policies, the algorithm simulates the transition process and records the average cost; (Step 2.3) the algorithm updates the internal model using the sampled information about the mapping of parameters $(u^1, ..., u^K)$ to average rewards $(r^1, ..., r^K)$. Evidently, the algorithm searches for the global optimum of a noisy, non-convex objective function, without guarantee of finding the best solution after I iterations. Let us now introduce four control policies which can be fully described by a parameter vector u.

Fixed-Cycle Policy

An intuitive solution to the problem of producing multiple products on a single machine is to fix a sequence in which these products are being produced in addition to quantities and (possibly) idle times. The fixed-cycle policy follows an idea originally proposed by Dobson (1987) for the ELSP and adapted by Federgruen and Katalan (1998) for the SELSP. The authors find the optimal production frequency for each product and then use this information to construct a fixed production sequence.

Denote $R \in \mathbb{N}^N$ as the set of integer frequencies and $Y \in \mathbb{N}^N$ as the set of order-up-to levels. We map the corresponding continuous parameter vector $u \in \mathbb{R}^{2N}$ to these two sets by decoding the vector into $R_n = \lfloor |u_{N+n}| \rfloor + 1$ and $Y_n = \lfloor |u_{N+n}| \rfloor + 1$. Since the product with the highest frequency can be scheduled at most every other time, we restrict the maximum frequency to be less than or equal to the sum of all other frequencies.

Figure 5.4 outlines a simple heuristic method to generate a production sequence $Q = \{Q_1, ..., Q_J\}$ of length J from a given set of integer frequencies. The heuristic assigns each product to a set of products with identical frequencies, $L_i = \{n : R_n = R_i\}$. Each set L_i is inserted into the sequence multiple times according to its frequency. In Step (3.2.2) the algorithm inserts L_i at position z and shifts the element currently at that position to the right. Suppose we have $R = \{2, 4, 1, 4, 2\}$, which gives us $L_1 = \{3\}, L_2 = \{1, 5\}, L_3 = \{\}, L_4 = \{2, 4\}$. Then, the heuristic would insert the L_i's into Q in the following way:

$$Q^1 = \{3\} \rightarrow Q^2 = \{1, 5, 3, 1, 5\} \rightarrow Q^3 = \{1, 5, 3, 1, 5\} \rightarrow Q^4 = \{2, 4, 1, 2, 4, 5, 3, 2, 4, 1, 2, 4, 5\}$$

The resulting production sequence features products being roughly evenly spaced over the entire cycle according to their integer frequencies.

For a given position j in sequence Q and order-up-to levels Y, the fixed-cycle policy is now given by

$$\pi(S; x) = \begin{cases} 0 & \text{if } \forall\ n : y_n = Y_n, \\ Q_{z(j)} & \text{otherwise,} \end{cases} \tag{5.22}$$

```
(1) Input arguments: integer frequencies $R$
(2) Group identical frequencies $L_i = \{n : R_n = i\}$
(3) Do for $i = 1, 2, \dots, \max_{n \in N} R_n$
    (3.1) $d \leftarrow \dim(Q) R_i^{-1}$
    (3.2) Do for $j = 1, 2, \dots, i$
        (3.2.1) $z \leftarrow \lfloor jd + 0.5 \rfloor + j \dim(L_i) + 1$
        (3.2.2) $Q \leftarrow Q' : Q'_k = \begin{cases} Q_k & \forall\, k < z, \\ L_{i,k-z} & \forall\, z \le k < z + \dim(L_i), \\ Q_{k-\dim(L_i)} & \forall\, k \ge z + \dim(L_i) \end{cases}$
(4) Return sequence $Q$
```

Figure 5.4: Heuristic method to generate an evenly spaced production sequence from given integer frequencies

where the recursive function z is defined as

$$z(j) = \begin{cases} j & \text{if } y_k < Y_k : k = Q_j, \\ z\big(j \bmod J + 1\big) & \text{otherwise.} \end{cases} \tag{5.23}$$

For a given position j, the function returns the next position in the sequence for which $y_n < Y_n$, where the modulus ensures that the production cycle is repeated as soon as $j = J$. Note that this approach allows for simultaneous optimization of production sequence and base-stock levels, in contrast to Anupindi and Tayur (1998) who propose a two-stage approach of fixing a schedule and then searching for base-stock levels.

As initial guess for the policy search, we propose to use a heuristic solution that is based on the *common cycle solution* to the ELSP. Denote k as a safety factor and $\hat{T}$ as the common cycle time. Then, we obtain a policy, where for each product n we set

$$R_n = 1, \quad Y_n = \max\left\{ \lfloor \mu_n \hat{T} + k\sigma_n \sqrt{\hat{T}} \rfloor, 1 \right\}. \tag{5.24}$$

Note that $Y_n \ge 1$ is a lower bound on the order-up-to level, since production would be zero otherwise. The common cycle time $\hat{T}$ and the safety factor k are given by

$$\hat{T} = \max\left\{ \sqrt{\frac{2\sum_{n=1}^{N} A_n}{\sum_{n=1}^{N} h_n \mu_n (1 - \mu_n p_n)}}, \frac{\sum_{n=1}^{N} W_n}{1 - \sum_{n=1}^{N} \mu_n p_n} \right\}, \quad k = \Phi^{-1}\left(\frac{v_n}{v_n + h_n \hat{T}} \right), \tag{5.25}$$

with Φ^{-1} as inverse normal distribution. Using the quantile of the compound Poisson distribution instead would add little additional value, since k is merely a parameter of the initial guess. As trust region we use $\sigma_i^u = \max\left\{\frac{1}{2}\mu_i^u, \delta\right\}$ for all policies, with $\delta \geq 1$ to ensure exploration in case $\mu_i^u = 0$.

Fixed-Cycle Policy with Preemption

A major drawback of using a fixed cycle in a stochastic production environment is its lack of flexibility. For example, assume that product i is next but still close to its order-up-to-level while product j, which is in line after i, is already out of stock. Then, it could be better to preempt production of j, instead of setting up for i and risking lost sales of j. Moreover, under the fixed-cycle policy the machine only goes idle when all products are at their order-up-to levels, which may not be the best choice when utilization is low, but inventory holding cost high.

To overcome these drawbacks, we suggest to add two additional control parameters, a *preemption* point and a *can-order* level. Denote $f \in \mathbb{R}_+^N$ as before, $Y^{(1)} \in \mathbb{N}^N$ as the set of preemption points, $Y^{(2)} \in \mathbb{N}^N$ as the set of can-order levels, and $Y^{(3)} \in \mathbb{N}^N$ as the set of order-up-to-levels. The corresponding parameter vector $u \in \mathbb{R}^{4N}$ can then be decoded into $f_n = |u_n| + 1$, $Y_n^{(2)} = \lfloor |u_{N+n}| \rfloor$, $Y_n^{(1)} = \max\{Y_n^{(2)} - 1 - \lfloor |u_{2N+n}| \rfloor, -1\}$ and $Y_n^{(3)} = Y_n^{(2)} + 1 + \lfloor |u_{3N+n}| \rfloor$.

Instead of rotating the entire sequence, as in Gallego (1990), we preempt the critical product, thereby turning the fixed production cycle into a dynamic one. When one or more products are at their preemption points or below, we update the production sequence and move the next product with $y_n \leq Y_n^{(1)}$ from its original position to the position that comes next. On the other side, when all products are still above their can-order level, the machine goes idle.

Given the current position j in sequence Q, the fixed-cycle policy with preemption is given by

$$\pi(S;x) = \begin{cases} 0 & \text{if } \forall\ n : y_n > Y_n^{(2)}, \\ Q'_{j+1} & \text{if } \exists\ n : y_n \leq Y_n^{(1)}, \\ Q_{z'(j+1)} & \text{otherwise}, \end{cases} \tag{5.26}$$

where $Q' = Z(Q, z''(j+1), j+1)$. Z is defined as

$$Z(Q,i,j) = \begin{cases} Q' \in \mathbb{Q} : Q'_j = Q_i, Q'_{k+1} = Q_k \forall\ j \leq k < i, Q'_k = Q_k \forall\ k < j \wedge k > i & \text{if } j \leq i, \\ Q' \in \mathbb{Q} : Q'_j = Q_i, Q'_{k-1} = Q_k \forall\ i < k \leq j, , Q'_k = Q_k \forall\ k < j \wedge k > i & \text{otherwise}, \end{cases} \tag{5.27}$$

with $\mathbb{Q}$ as the power set of Q. The function Z returns a new sequence which is identical to Q, except that the product at position i is moved to position j. The recursive functions z' and z''

are given by

$$z'(j) = \begin{cases} j & \text{if } y_k < Y_k^{(3)} : k = Q_j, \\ z'\big(j \bmod J + 1\big) & \text{otherwise,} \end{cases} \tag{5.28}$$

$$z''(j) = \begin{cases} j & \text{if } y_k \leq Y_k^{(1)} : k = Q_j, \\ z''\big(j \bmod J + 1\big) & \text{otherwise.} \end{cases} \tag{5.29}$$

Note that the preemption persists in the next decision epoch, so that Q' becomes Q after transition from S to S'.

As initial guess for the policy search, we propose to use the fixed-cycle policy and set $Y_n^{(3)} = Y_n$ and $Y_n^{(1)} = -1, Y_n^{(2)} = Y_n^{(3)} - 1$.

Base-Stock Policy

An alternative to following a fixed production sequence is to trigger new production orders based entirely on current inventory levels (Graves, 1980). In addition to an order-up-to level which determines production quantities, we define a reorder point that initiates new production orders. In case two or more products reach their reorder points after a production run has been completed, we use the earliest *run-out time* as priority rule (Gascon et al., 1994). The run-out time is defined as the average time until product n is out-of-stock minus its setup time.

Denote $Y_n^{(1)}$ as reorder point and $Y_n^{(2)}$ as order-up-to level, with $Y_n^{(1)} = |u_n|$ and $Y_n^{(2)} = Y_n^{(1)} + 1 + |u_{N+n}|$ for a given vector $u \in \mathbb{R}^{2N}$. Let $I = \{n : y_n \leq Y_n^{(1)}\}$ define the set of products with inventories below their reorder points. The control policy is then given by

$$\pi(S; x) = \begin{cases} m & \text{if } m > 0 \text{ and } y_m < Y_m^{(2)}, \\ \arg\min\limits_{i \in I} \left\{ \frac{y_i}{\mu_i} - W_i \right\} & \text{else if } \exists\ n : y_n \leq Y_n^{(1)}, \\ 0 & \text{else if } \forall\ n : y_n > Y_n^{(1)}. \end{cases} \tag{5.30}$$

As initial guess for the policy search, we propose to use a heuristic solution that is based on the Doll and Whybark heuristic adapted by Gascon et al. (1994). Denote k as a safety factor and $\hat{T}$ as the common cycle time as before. Then, we obtain an (s,S) policy, where for each product n we set

$$Y_n^{(1)} = \max\left\{ \lfloor \mu_n W_n + k \cdot \sqrt{\sigma_n^2 \hat{T}} \rfloor, 0 \right\}, \quad Y_n^{(2)} = Y_n^{(1)} + \max\left\{ \lfloor \mu_n (1 - \mu_n p_n) \hat{T} \rfloor, 1 \right\}. \tag{5.31}$$

Note that $Y_n^{(1)} \geq 0$ in order to trigger a production order, and $Y_n^{(2)} - Y_n^{(1)} \geq 1$ to avoid that production is zero.

Can-Order Base-Stock Policy

A major drawback of using a simple base-stock policy is its inability to respond to critical inventory levels during a production run. For example, assume that product i is set up and below its order-up-to level but far above its reorder point while product j is already out-of-stock. Then, it could be better to interrupt production of i and change over to j.

For the can-order base-stock policy, we suggest to use a *can-order* point as well as a *can-order-up-to* level in addition to reorder point and order-up-to-level. Denote $Y_n^{(1)}$, $Y_n^{(2)}$ as before and $Y_n^{(3)}$ as can-order point and $Y_n^{(4)}$ as can-order-up-to level, with $Y_n^{(3)} = Y_n^{(1)} + |u_{2N+n}|$ and $Y_n^{(4)} = Y_n^{(3)} + 1 + |u_{3N+n}|$ for a given $u \in \mathbb{R}^{4N}$. Let $I = \{n : y_n \leq Y_n^{(1)}\}$ as before and $I' = \{n : y_n \leq Y_n^{(3)}\}$ define the set of products with inventories below their can-order points. The control policy is then given by

$$\pi(S;x) = \begin{cases} m & \text{if } m > 0 \text{ and } y_m < Y_m^{(2)} \\ \arg\min\limits_{i \in I} \left\{ \frac{y_i}{\mu_i} - W_i \right\} & \text{else if } \exists\, n : y_n \leq Y_n^{(1)}, \\ m & \text{else if } \forall\, n : y_n > Y_n^{(1)} \text{ and } m > 0 \text{ and } y_m < Y_m^{(4)} \\ \arg\min\limits_{i \in I'} \left\{ \frac{y_i}{\mu_i} - W_i \right\} & \text{else if } \forall\, n : y_n > Y_n^{(1)} \text{ and } \exists\, n : y_n \leq Y_n^{(3)}, \\ 0 & \text{else if } \forall\, n : y_n > Y_n^{(3)} \end{cases} \tag{5.32}$$

The machine continues production as long as there exists a product with an inventory level below its can-order point. When a product is above its can order-up-to level but another drops below its reorder point, production can be interrupted to set up the critical product.

As initial guess for the policy search, we propose to use a simple base-stock policy and set $Y_n^{(1)} = Y_n^{(2)}, Y_n^{(3)} = Y_n^{(4)}\ \forall\, n$.

5.3 Numerical Results

5.3.1 Experimental Design

To answer our research questions, we carried out an extensive numerical study, for which we generated instances of model parameters which are maximally different and cover a large range of parameters.

For each instance of the problem, we choose seven values of model parameters for each of the N products: mean demand per period, its variance, lost sales cost, holding cost, setup cost, setup time, and production time. For our study, we first fixed mean demand and defined the variance through the coefficient of variation (CV Demand). Second, we expressed production time through the workload that would result from producing N products with identical demand and production rates, $\rho_n = N\mu_n p_n$ (Load Factor). Third, with the production time given by

Design Parameter	Min Value	Max Value
Avg Mean Demand	5	5
Div Mean Demand	0	0.5
Avg Lost Sales Cost	100	100
Div Lost Sales Cost	0	0.5
Avg Hold/LS Cost ($\bar{\kappa}$)	0.001	0.01
Div Hold/LS Cost	0	0.5
Avg Setup/Prod Time ($\bar{\eta}$)	1	100
Div Setup/Prod Time	0	0.5
Avg CV Demand ($\bar{c}_D$)	0.5	1.5
Div CV Demand	0	0.5
Avg Load Factor ($\bar{\rho}$)	0.3	0.9
Div Load Factor	0	0.5

Table 5.1: Intervals of the design parameters that span the experimental area

mean demand and load factor, we derived the setup time from the ratio of setup to production time (Setup/Prod Time). Fourth, we fixed the lost sales cost and expressed the holding cost through the ratio of holding to lost sales cost (Holding/LS Cost). Finally, we set all setup costs to zero, since setup cost often represent no more than the cost of working time during a setup, which is already covered by the setup time.

Since the number of parameters increases proportionally in the number of products N, we defined experimental design variables which summarize an entire set of N parameter values. If we view each value of a set as a realization of a uniform random variable, we can describe the set by an average and a coefficient of variation, with the latter serving as a measure of parameter diversity, e.g., diversity in mean demand or lost sales cost. For our study, we fixed the averages, Avg Mean Demand and Avg Lost Sales Cost, to 5 and 100, respectively, and then sampled the *diversity factors*, Div Mean Demand and Div Lost Sales Cost, from the interval $[0.0, 0.5]$. Averages (Avg) and diversity factors (Div) of CV Demand, Load Factor, Setup/Prod Time, and Holding/LS Cost were sampled accordingly. The ranges of these design parameters are given in Table 5.1.

With Avg Mean Demand and Avg Lost Sales Cost fixed, a problem instance can now be described by a ten-dimensional design point. For our numerical study, we generated 1000 design points for problems with $N \in \{3, 5, 10\}$ products by sampling a ten-dimensional Faure sequence. This gives us a so-called space-filling design which has the useful property that design points lie not only at the edges of the hypercube that spans the experimental area but also at its interior (Chen et al., 2006).

To decode a design point into a model configuration, we used a variant of *descriptive sampling* (Saliby, 1990). Denote X as a uniform random variable, with average $\bar{X}$ and coefficient of variation (diversity factor) c_X, to describe a set of N parameters. Using descriptive sampling,

		Num of Products (N)		
		3	5	10
AVI	T	100,000,000	200,000,000	500,000,000
	γ	0.01	0.01	0.01
	a	1,000,000	2,000,000	5,000,000
	b	0.1	0.05	0.02
	K	20	20	20
	L	20	20	20
DPS	IK	900	2,500	10,000
	T	100,000	100,000	100,000
	e	1000	1000	1000
	δ	5	5	5

Table 5.2: Algorithmic parameters

a realization of a parameter x_n associated with the n-th product is then given by

$$x_n = x'_{\Theta(n)}, \ x'_j = \bar{X} + \sqrt{6}\left(\tfrac{j-1}{N-1} - \tfrac{1}{2}\right)\bar{X}c_X, \ n, j \in \{1, \ldots, N\}, \tag{5.33}$$

with Θ serving as a random mapping from n to j, i.e., we shuffle. For example, for a Load Factor with average $\bar{\rho} = 0.5$ and coefficient of variation $c_\rho = 0.2$, one possible permutation of the set of parameters for a three-product problem would be $\{0.5, 0.378, 0.622\}$.

5.3.2 Implementation

We implemented the solution algorithms, the simulation model and our experiments in Java, and used SPSS 17 for our statistical analyses. As implementation of the CMA-ES algorithm, we used the Java source code provided by Hansen (2007). For all solution algorithms, we carried out pretests to optimize their algorithmic parameters which are summarized in Table 5.2. Note that for approximate value iteration (AVI), we have to define maximum inventory levels. We therefore first optimized the base-stock policy over an unbounded state space using direct policy search (DPS) and then set $\bar{y}_n = 1.2Y_n^{(2)}$. For policy evaluation, we generated a single sample path using common random numbers over 1,000,000 decision epochs, after an initial transient phase of 10,000 decision epochs. We then evaluated all policies that were optimized by AVI or DPS using this sample path.

5.3.3 Results

Influence of Model Parameters on Average Cost

We studied the influence of the design parameters on the expected average cost by conducting an analysis of variance (ANOVA). Since each additional product increases the total mean demand,

	BEST		FCP		BSP		AVI		
Factor	r^2	F	r^2	F	r^2	F	r^2	F	df
Num of Products	0.00	4.4	0.00	20.5	0.00	32.9	0.03	391.6	1
Div Mean Demand	0.00	9.3	0.00	41.0	0.01	150.3	0.00	1.0	1
Div Lost Sales Cost	0.00	41.1	0.00	21.8	0.00	0.2	0.00	51.5	1
Avg Hold/LS Cost	0.28	4720.6	0.29	4504.5	0.23	2728.0	0.22	3231.6	1
Div Hold/LS Cost	0.00	7.1	0.00	4.2	0.00	4.7	0.00	4.9	1
Avg Setup/Prod Time	0.22	3603.7	0.21	3341.5	0.19	2264.2	0.19	2907.4	1
Div Setup/Prod Time	0.00	5.0	0.00	4.3	0.00	1.9	0.00	1.1	1
Avg CV Demand	0.03	533.6	0.03	514.5	0.02	260.5	0.02	296.9	1
Div CV Demand	0.00	13.0	0.00	8.5	0.00	7.0	0.00	11.5	1
Avg Load Factor	0.28	4710.7	0.27	4268.7	0.29	3449.0	0.34	5157.2	1
Div Load Factor	0.00	0.4	0.00	0.7	0.00	7.1	0.00	9.2	1
Linear Model	0.82	1245.4	0.81	1161.3	0.75	812.1	0.81	1101.8	11

Sample Size = 3000, r^2 = coefficient of determination, F = F test statistic, df = degrees of freedom.

Table 5.3: ANOVA of the influence of design parameters on cost per product for different policies

the expected average cost increases proportionally in the number of products. To remove this effect, we refer to *cost per product* as the expected average cost divided by the number of products.

We ran separate ANOVAs for different policy groups: FCP, BSP, AVI and BEST. We refer to the group of fixed-cycle policies as FCP, base-stock policies as BSP and approximate value iteration with one of the two approximation schemes as AVI. For each simulated problem instance, we selected the lowest cost within a policy group for the analysis. Additionally, we created an auxiliary group, BEST, which contained the lowest known cost for each instance. The percentage of variance in cost per product that can be explained by a design parameter is measured by the coefficient of determination (r^2).

The results of the ANOVA are shown in Table 5.3. As can be seen from the last row (Linear Model), all factors together explain 75 to 82 percent of the variance in cost per product, irrespective of the policy group. Regardless of the solution method, the r^2 of all diversity factors is close to zero. This indicates that all policies were capable of compensating diversity in model parameters, so that the cost effect of diversity becomes negligible. The most important factors are Avg Load Factor, Avg Setup/Prod Time, and Avg Holding/LS Cost. Since Avg Load Factor and Avg Setup/Prod Time largely affect system utilization, this indicates that, within the given range of parameters, utilization is a more important cost driver than variability. However, variability has a much larger influence on FCP (F=514.5) than on BSP (F=260.6) or AVI (F=296.9), which indicates that dynamic-cycle policies are better able to deal with demand uncertainty than fixed-cycle policies. Also, except for AVI, the number of products does not have a noteworthy impact on cost per product, which can be seen from the comparatively low

N	$\bar{\rho}$	$\bar{\eta}$	Size	*BEST*	FCP$_0$	FCP$_1$	FCP$_2$	BSP$_0$	BSP$_1$	BSP$_2$	AVI$_1$	AVI$_2$
3	low	low	255	*0.57*	0.74	0.61	0.61	0.91	0.64	0.62	0.90	**0.59**
		high	247	*0.88*	1.13	0.97	0.96	1.48	1.03	1.00	1.38	**0.90**
	high	low	249	*0.93*	1.51	1.02	1.02	1.59	1.15	1.10	1.17	**0.98**
		high	249	*1.65*	3.68	1.83	**1.80**	3.12	2.17	2.08	2.02	**1.81**
5	low	low	251	*0.51*	0.65	0.55	**0.53**	0.66	**0.53**	**0.53**	1.14	0.55
		high	250	*0.87*	1.05	0.93	**0.90**	1.21	0.93	0.92	1.31	0.92
	high	low	255	*0.96*	1.43	1.02	**0.98**	1.47	1.13	1.10	1.44	1.14
		high	244	*1.72*	2.98	1.81	**1.73**	2.79	2.06	2.02	2.44	2.02
10	low	low	254	*0.47*	0.63	0.56	0.50	0.54	0.47	**0.47**	1.17	0.56
		high	246	*0.82*	1.03	0.96	0.87	0.98	**0.83**	**0.83**	2.20	1.05
	high	low	251	*0.92*	1.27	1.04	**0.95**	1.23	1.00	0.99	2.06	1.41
		high	249	*1.75*	2.92	1.91	**1.77**	2.83	2.12	2.05	3.85	2.57
	Mean		3000	*1.00*	1.58	1.10	**1.05**	1.56	1.17	1.14	1.75	1.20

Table 5.4: Comparison of standardized mean cost per product for different policies and problem categories

F-values. We therefore expect the solution quality of FCP and BSP to remain fairly stable even for larger problems.

Policy Evaluation

In our second analysis, we wanted to find out which solution method performs best. As before, we refer to the fixed-cycle policy as FCP (FCP$_0$ = common cycle solution, FCP$_1$ = fixed-cycle policy, FCP$_2$ = fixed-cycle policy with preemption) and to the base-stock policy as BSP (BSP$_0$ = Doll & Whybark heuristic, BSP$_1$ = base-stock policy, BSP$_2$ = can-order base-stock policy). For both types of policies, we used the less sophisticated policy as initial guess during direct policy search. We refer to AVI with the first approximation scheme as AVI$_1$ and with the second scheme as AVI$_2$. For all policies, we recorded the mean cost per product for 3, 5, and 10 products, as well as for low and high levels of Avg Load Factor (low: $\bar{\rho} < 0.6$, high: $\bar{\rho} \geq 0.6$) and Avg Setup/Prod Time (low: $\bar{\eta} < 51$, high: $\bar{\eta} \geq 51$). (For a justification, see Section 5.3.3.) The mean costs per product in each problem category were standardized by the mean cost per product of BEST across all problem instances.

The results of the analysis are shown in Table 5.4. The lowest mean in each category is highlighted in bold. (Note that when the difference to the second lowest average is not significant, i.e., $p \geq 0.01$, both values are highlighted.) Overall, FCP$_2$ yields the lowest mean cost per product. For small problems with three products, AVI$_2$ returns the lowest mean, but looses its competitiveness as the problem size increases. Since AVI$_1$ yields a much higher

				Mean			MAD			Frequency		
N	$\bar{\rho}$	$\bar{\eta}$	Size	FCP	BSP	AVI	FCP	BSP	AVI	FCP	BSP	AVI
3	low	low	255	0.61	0.62	**0.59**	0.05	0.10	**0.03**	16%	42%	**42%**
		high	247	0.95	1.00	**0.89**	0.11	0.14	**0.03**	30%	14%	**57%**
	high	low	249	1.01	1.09	**0.97**	0.10	0.20	**0.07**	25%	21%	**54%**
		high	249	1.79	2.07	**1.73**	0.25	0.44	**0.14**	46%	6%	**48%**
5	low	low	251	**0.53**	**0.52**	0.55	**0.02**	0.04	0.04	20%	**64%**	16%
		high	250	**0.89**	0.92	0.91	**0.04**	0.08	0.06	**40%**	37%	23%
	high	low	255	**0.98**	1.10	1.13	**0.03**	0.18	0.18	**62%**	27%	11%
		high	244	**1.72**	2.01	1.98	**0.04**	0.34	0.28	**82%**	12%	6%
10	low	low	254	0.50	**0.47**	0.56	0.03	**0.01**	0.09	3%	**97%**	0%
		high	246	0.87	**0.82**	1.05	**0.05**	0.07	0.23	9%	**91%**	0%
	high	low	251	**0.95**	0.98	1.39	**0.04**	0.18	0.47	34%	**66%**	0%
		high	249	**1.76**	2.04	2.50	**0.05**	0.45	0.75	**65%**	35%	0%
	Mean		3000	**1.04**	1.13	1.18	**0.07**	0.23	0.23	36%	**43%**	21%

Table 5.5: Comparison of different groups of policies

mean than AVI_2 across all categories, we conclude that an approximate value function that is completely separable in the inventory state variables is insufficient to approximate the true value function of the SMDP. The heuristics, FCP_0 and BSP_0, are also not competitive. Moreover, we observe that the cost induced by FCP_2 are considerably lower than those of BSP_2 when both, Avg Load Factor and Avg Setup/Prod Time, are high. While the difference between FCP_1 and FCP_2 gets larger as the number of products increases, the improvement of using BSP_2 over BSP_1 is small.

Again, like in the ANOVA, we grouped the policies into FCP, BSP and AVI. For each group and each category, we then recorded the standardized means of the lowest in-group cost, as before (Mean). Additionally, we recorded the mean absolute deviation (MAD) of the lowest in-group cost from the lowest cost across all groups (BEST). For a parameter instance i, the absolute deviation of FCP is given by

$$|\text{Cost}_i(\text{FCP}) - \min\{\text{Cost}_i(\text{FCP}), \text{Cost}_i(\text{BSP}), \text{Cost}_i(\text{AVI})\}|.$$

We then computed the MAD across all instances within a category, omitting cases where the policy yields the lowest known cost, which gives us the MAD over those instances where the policy was not best. Also, for each category, we recorded the relative frequency of how often a group provided the lowest cost (Frequency).

The results of the analysis are shown in Table 5.5. FCP yields the overall lowest mean across all instances. With the exception of the three product case, it also returns the lowest mean

Predictor	b	Std Error	Wald χ^2	df	p-Value	Odds Ratio
Num of Products	0.390	0.019	443.113	1	0.000	1.477
Avg Setup/Prod Time	-0.031	0.002	292.071	1	0.000	0.969
Avg Load Factor	-7.282	0.329	489.731	1	0.000	0.001
Avg Hold/LS Cost	-112.512	18.464	37.130	1	0.000	0.000
Div Mean Demand	-3.084	0.341	81.658	1	0.000	0.046
Constant	4.825	0.273	312.507	1	0.000	n/a
Test			χ^2	df	p-Value	
Log-Likelihood Ratio			1422.000	1	0.000	
Hosmer-Lemeshow			20.275	8	0.009	

Cox and Snell R^2=0.377, Nagelkerke R^2=0.504, b = beta coefficient, df = degrees of freedom, n/a = not applicable.

Table 5.6: Logistic regression analysis of the frequency for BSP having lower average cost than FCP

in categories with a high Avg Load Factor. On the other hand, BSP has the highest number of instances where it is the best policy, in particular for problem instances with 10 products. Considering that FCP yields the lowest MAD overall, this implies that whenever BSP is worse its difference to the best policy must be larger on average than whenever FCP is worse.

We conclude that, although a base-stock policy is more frequently better, a fixed-cycle policy is the more robust choice. Moreover, the fixed-cycle policies are generally better than the base-stock policies in highly utilized systems. In contrast, approximate value iteration only works well in systems with a small number of products, and only with a value function approximation that considers dependencies among inventory state variables.

A Discrete Choice Model for Production Policies

The previous analysis has shown that there are some model configurations where it is better to use a base-stock policy and others where it is better to use a fixed-cycle policy. To analyze the drivers that make one policy perform better than the other, we developed a discrete choice model, which can be described by the *logit* function

$$\text{logit}^{-1}(Y_i) = \frac{\exp(Y_i)}{1+\exp(Y_i)},\ Y_i = b_0 + b_1 X_{1i} + ... + b_k X_{ki}.$$

We used this model to specify the probability that BSP performs better than FCP. To estimate the coefficients of the logit function, we ran a binomial logistic regression. We used backwards stepwise regression with the design parameters as independent variables and 'BSP is best' as dependent variable. Starting with the variable with the lowest Wald statistic, we iteratively removed variables as long as the change in the log-likelihood ratio was not significant.

Table 5.6 summarizes the of the logistic regression analysis. The odds ratio of 1.477 indicates that a larger number of products makes it more likely for BSP to perform better than FCP. We can also see this tendency in Table 5.5, where the frequency of BSP as the best policy increases in the number of products. Avg Load Factor and Avg Setup/Prod Time, on the other hand, decrease the likelihood of BSP being the best policy (odds ratios < 1). Although Avg Holding/LS Cost and Div Mean Demand are significant, they are less important, which can be seen by their lower Wald statistics.

Table 5.7 compares the observed frequency for BSP having lower average cost than FCP to the frequency predicted by the discrete choice model. The prediction is accurate in about 80% of all cases, compared to 50% if we would randomly choose a policy. The standardized MAD between cost of BSP versus FCP in case of choosing the wrong policy is 0.033, compared to 0.07 when we always choose FCP. This implies that even if the wrong policy was chosen the error would be smaller than if the most robust policy was used.

The results highlight the strengths and weaknesses of each policy. Systems with a large number of products, for instance, require flexible production policies, which increases the odds that a base-stock policy performs better. In highly utilized systems, however, the base-stock policy discriminates products with low setup times, as it always skips to the product with the lowest runout time. This effect increases the odds that a fixed-cycle policy performs better, as this policy strictly follows a predefined production sequence. This also confirms the finding of Vaughan (2007) that order-point policies outperform cyclical policies in cases where the number of products is large and system utilization low.

Comparison With Optimal Policy

In our final analysis, we take a look at the difference between policies obtained through simulation optimization and the optimal policy derived from relative value iteration (RVI). To keep problems computationally tractable for RVI, we only study problems with three products and fixed the maximum inventory level at $\bar{y}_n = 25$. We stopped RVI when the gap between lower and upper bound was less than one percent. Since the resulting inventory state space is too restrictive for most problems in the previous sample, we created a new sample, where we reset the intervals of the design parameters Avg Hold/LS Cost and Avg Setup/Prod Time to $[0.01, 0.1]$ and $[1, 20]$,

	Predicted		
Observed	FCP is best	BSP is best	% Correct
FCP is best	1277	308	80.6
BSP is best	328	1087	76.8
Overall			78.8

Table 5.7: Observed and predicted frequency for BSP having lower cost per product than FCP

N	$\bar{\rho}$	$\bar{\eta}$	Size	*RVI*	FCP$_0$	FCP$_1$	FCP$_2$	BSP$_0$	BSP$_1$	BSP$_2$	AVI$_1$	AVI$_2$
3	low	low	64	*1.92*	2.25	2.02	2.03	2.24	2.03	**2.00**	2.13	2.04
		high	63	*2.68*	2.95	2.80	2.77	2.93	2.81	2.78	2.95	**2.74**
	high	low	61	*2.67*	3.25	2.92	2.90	3.25	3.00	2.91	3.04	**2.84**
		high	62	*3.62*	4.68	3.88	3.75	4.69	4.25	4.21	3.94	**3.69**
	Mean		250	*2.72*	3.30	2.92	2.87	3.29	3.04	2.99	3.03	**2.84**

Table 5.8: Comparison of standardized mean average cost for tractable model instances with 3 products

respectively.

Table 5.8 shows the mean cost per product over these 250 instances, categorized and standardized as in Tables 5.4 and 5.5, with a new definition of low and high levels of Avg Setup/Prod Time (low: $\bar{\eta} < 10.5$, high: $\bar{\eta} \geq 10.5$). As in Table 5.4, the best heuristic policy over all is obtained by AVI$_2$, with cost per product less than 5 percent above the upper bound of RVI on average. For FCP$_2$, cost are less than 6 percent above, for BSP$_2$, less than 10 percent above the upper bound on average. Since the number of products only has a small influence on cost per product (see Table 5.3), we conjecture that the performance of BSP and FCP will be similar for larger problems. Also note that results for BSP$_2$ have improved, since the mean Avg Setup/Prod Time in the new sample is now lower than in the old one, which supports the findings of the logistic regression analysis.

5.4 Discussion

We have studied simulation optimization methods for the stochastic economic lot scheduling problem. Based on a large-scale numerical study, we found that the classic ADP approach of approximate value iteration and stochastic gradient updates is not competitive for larger problems. Initial tests with alternative ADP approaches did not perform better. Using (recursive) least squares in place of stochastic gradients to update the value function had been considered but was computationally too expensive. Replacing the piecewise-constant approximation with a linear interpolation also did not significantly improve approximation quality. Applying soft-max or epsilon-greedy exploration during sampling provided no advantage over pure exploitation. These results turn the SELSP into a good benchmark for future research in ADP.

Simple production policies optimized by a global search algorithm provide the most comprehensible and robust solution to this problem. In our numerical study, base-stock policies were more often the better choice, but were outperformed by fixed-cycle policies in highly utilized systems. The most reliable choice overall was a fixed-cycle policy which preempts production of a product as soon as the inventory level of an upcoming product drops below a certain level.

When comparing this policy with the optimal policy for small, tractable problem instances, we observe that costs are less than six percent above the minimal costs on average.

An interesting extension of the current model would be to consider sequence-dependent setup times, which would require a reformulation of the value function approximation as well as the development of new control policies. Other extensions, such as backlogs instead of lost sales or uncertain process times could be motivated by specific applications but do not significantly change problem complexity with respect to simulation optimization.

Chapter 6

Conclusion

This thesis has shed new light on different stochastic-dynamic resource management problems and on how amenable these problem are to solutions based on approximate dynamic programming (ADP). On the one hand, there is the seemingly small stochastic production planning problem that resisted all efforts of state-of-the-art ADP techniques. On the other hand, there is the multi-stage, multi-reservoir management problem which could be solved close to optimality even for an industry-scale problem.

These findings highlight that ADP is not a panacea to solve every stochastic-dynamic decision problems, and there are problems that still remain intractable even for the state of the art. Nevertheless, ADP has shifted the boundary, and we are now capable of solving formerly intractable problems by combining careful modeling with good sampling strategies and suitable function approximators.

Whether a problem can be solved using ADP or not depends on the problem. All of the models studied in this thesis had in common that the size of the state space was driven by the number of resources that had to be managed over time, be it inventories as in the production planning problem or reservoirs as in the energy planning problem. However, there is a difference in the way decisions were made. While the action space in the production planning problem was discrete and a separate value function had to be stored for each production decision, the reservoir management problem could be reformulated as a convex optimization problem. This property provided us with a tremendous advantage, since we were able to extract gradient information from the optimal solution that could be used to construct a polyhedral, global estimate of the value function. Since the global estimate was optimistic everywhere but in states that had already been sampled, the sampling process explored prospective states until there were no more states worth exploring.

This result holds important implications for the optimization of other stochastic-dynamic resource management problems. As long as the optimization problem is convex and randomness is independent of the resource state, many problems that appear intractable at first can actually be solved quite efficiently. By contrast, if the optimization problem is discrete, there exists no structure that can be exploited to guide exploration. In that case, the problem remains hard to solve, even for approximate dynamic programming techniques, and simple but well-adjusted decision policies may be the better choice.

One direction for future research would be to explore these insights and construct polyhedral approximations to derive tight upper bounds of the value function of other stochastic-dynamic decision problems. For example, if we reformulated the production planning problem, so that the optimization problem at each stage was convex, we could easily derive a near-optimal solution by constructing a polyhedral approximation of the value function. However, it remains an open question, how well such an approximation would perform for the type of non-convex production planning problems typically encountered in practice.

Another direction for future work would be to establish the production planning problem with its discrete action space as a benchmark for new developments in approximate dynamic programming. In particular, current efforts to improve the efficiency of the information collection process would benefit from a challenging problem to demonstrate the effectiveness of new learning strategies.

Bibliography

Altiok, T. and G. Shiue (1994). Single-stage, multi-product production/inventory systems with backorders. *IIE Transactions 26*(2), 52–61.

Altiok, T. and G. Shiue (1995). Single-stage, multi-product production/inventory systems with lost sales. *Naval Research Logistics 42*(6), 889–913.

Anupindi, R. and S. Tayur (1998). Managing stochastic multiproduct systems: model, measures, and analysis. *Operations Research 46*(3), 98–111.

Atkeson, C. G., A. W. Moore, and S. Schaal (1997). Locally weighted learning. *Artificial Intelligence Review 11*, 11–73.

Auer, J. (2011). Hydropower in Europe. Market Study. Deutsche Bank Research.

Baíllo, A., M. Ventosa, M. Rivier, and A. Ramos (2004). Optimal offering strategies for generation companies operating in electricity spot markets. *IEEE Transactions on Power Systems 19*(2), 745–753.

Bathurst, G. N. and G. Strbac (2003). Value of combining energy storage and wind in short-term energy and balancing markets. *Electric Power Systems Research 67*, 1–8.

Bathurst, G. N., J. Weatherill, and G. Strbac (2002). Trading wind generation in short term energy markets. *IEEE Transactions on Power Systems 17*(3), 782–789.

Bertsekas, D. (2007). *Dynamic Programming and Optimal Control* (3rd ed.), Volume 2. Athena Scientific.

Bertsekas, D. P. and J. Tsitsiklis (1996). *Neuro-Dynamic Programming.* Athena Scientific.

Bourland, K. and C. Yano (1994). The strategic use of capacity slack in the economic lot scheduling problem with random demand. *Management Science 40*, 1690–1704.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning 49*, 233–246.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization.* Cambridge University Press.

Brander, P. and R. Forsberg (2006). Determination of safety stocks for cyclic schedules with stochastic demands. *International Journal of Production Economics 104*(2), 271–295.

Brunetto, C. and G. Tina (2007). Optimal hydrogen storage sizing for wind power plants in day ahead electricity markets. *IET Renewable Power Generation 1*(4), 220–226.

Cerisola, S., Á. Baíllo, J. M. Fernández-López, A. Ramos, and R. Gollmer (2009). Stochastic power generation unit commitment in electricity markets: A novel formulation and a comparison of solution methods. *Operations Research 57*(1), 32–46.

Chen, C., K.-L. Tsui, R. Barton, and M. Meckesheimer (2006). A review on design, modeling and applications of computer experiments. *IIE Transactions 38*(4), 273–291.

Chen, Z. L. and W. B. Powell (1999). Convergent cutting-plane and partial-sampling algorithm for multistage stochastic linear programs with recourse. *Journal of Optimization Theory and Applications 102*, 497–524.

Cormen, T., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms.* MIT Press.

Davis, S. G. (1990). Scheduling economic lot size production runs. *Management Science 36*(8), 985–998.

Dobson, G. (1987). The economic lot-scheduling problem: achieving feasibility using time-varying lot sizes. *Operations Research 35*(5), 764–771.

Draper, N. and H. Smith (1998). *Applied regression analysis.* Wiley series in probability and statistics: Texts and references section. Wiley.

Eichhorn, A., H. Heitsch, and W. Römisch (2009). Scenario tree approximation and risk aversion strategies for stochastic optimization of electricity production and trading. In *Optimization in the Energy Industry*, pp. 321–346. Springer.

Elmaghraby, S. (1978). The economic lot scheduling problem (ELSP): review and extensions. *Management Science 24*(6), 587–598.

Ernst, D., P. Geurts, and L. Wehenkel (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research 6*, 503–556.

Federgruen, A. and Z. Katalan (1996). The stochastic economic lot scheduling problem: cyclical base-stock policies with idle times. *Management Science 42*(6), 783–796.

Federgruen, A. and Z. Katalan (1998). Determining production schedules under base-stock policies in single facility multi-item production systems. *Operations Research 46*(6), 883–898.

Feeney, G. and C. Sherbrooke (1966). The (s-1,s) inventory policy under compound Poisson demand. *Management Science 12*(5), 391–411.

Flach, B., L. Barroso, and M. Pereira (2010). Long-term optimal allocation of hydro generation for a price-maker company in a competitive market: latest developments and a stochastic dual dynamic programming approach. *IET Generation, Transmission and Distribution 4*(2), 299–314.

Fleten, S.-E. and T. K. Kristoffersen (2007). Stochastic programming for optimizing bidding strategies of a nordic hydropower producer. *European Journal of Operational Research 181*(2), 916–928.

Fleten, S.-E. and T. K. Kristoffersen (2008). Short-term hydropower production planning by stochastic programming. *Computers and Operations Research 35*(8), 2656–2671.

Gallego, G. (1990). Scheduling the production of several items with random demands in a single facility. *Management Science 36*(12), 1579–1592.

García-González, J., R. M. R. De la Muela, L. M. Santos, and A. M. González (2008). Stochastic joint optimization of wind generation and pumped-storage units in an electricity market. *IEEE Transactions on Power Systems 23*(2), 460–468.

García-González, J., E. Parrilla, and A. Mateo (2007). Risk-averse profit-based optimal scheduling of a hydro-chain in the day-ahead electricity market. *European Journal of Operational Research 181*(3), 1354–1369.

Gascon, A., R. Leachman, and P. Lefrançois (1994). Multi-item, single-machine scheduling problem with stochastic demands: a comparison of heuristics. *International Journal of Production Research 32*(3), 583–596.

Gjelskvik, A., B. Mo, and A. Haugstad (2010). Long- and medium-term operations planning and stochastic modelling in hydro-dominated power systems based on stochastic dual dynamic programming. In *Handbook of Power Systems I*, pp. 33–55. Springer.

Glanz, F., W. Miller, and L. Kraft (1991). An overview of the CMAC neural network. In *IEEE Conference on Neural Networks for Ocean Engineering*, pp. 301–308.

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing 21*, 178–192.

Graves, F., T. Jenkin, and D. Murphy (1999). Opportunities for electricity storage in deregulating markets. *The Electricity Journal 12*(8), 46–56.

Graves, S. (1980). The multi-product production cycling problem. *AIIE Transactions 12*(3), 233–240.

Guigues, V. (2011). SDDP for some interstage dependent risk averse problems and application to hydro-thermal planning. PUC Rio, Rio de Janeiro, Brasil. Working paper.

Hansen, N. (2007). CMA-ES source code in Java.

Hansen, N. and A. Ostermeier (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*(2), 159–195.

Heitsch, H. and W. Römisch (2003). Scenario reduction algorithms in stochastic programming. *Computational Optimization and Applications 24*, 187–206.

Higle, J. L. and S. Sen (1991). Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research 16*(3), 650–669.

Hochreiter, R. and D. Wozabal (2010). A multi-stage stochastic programming model for managing risk-optimal electricity portfolios. In *Handbook of Power Systems II*, pp. 383–404. Springer.

Hsu, W.-L. (1983). On the general feasibility test of scheduling lot sizes for several products on one machine. *Management Science 29*(1), 93–105.

Jung, T. and D. Polani (2007). Kernelizing LSPE(λ). In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 338–345.

Korpås, M. and A. T. Holen (2006). Operation planning of hydrogen storage connected to wind power operating in a power market. *IEEE Transactions on Energy Conversion 21*(3), 599–606.

Krieg, G. and H. Kuhn (2002). A decomposition method for multi-product kanban systems with setup times and lost sales. *IIE Transactions 34*, 613–625.

Lagoudakis, M. G. and R. Parr (2003). Least-squares policy iteration. *Journal of Machine Learning Research 4*, 1107–1149.

Loehndorf, N. and S. Minner (2010). Optimal day-ahead trading and storage of renewable energies - an approximate dynamic programming approach. *Energy Systems 1*, 61–77.

Loehndorf, N. and S. Minner (2011). Simulation optimization for the stochastic economic lot scheduling problem. University of Vienna. Working paper.

Loehndorf, N., D. Wozabal, and S. Minner (2011). Approximate dual dynamic programming for the problem of optimal bidding with hydro storage systems. University of Vienna. Working paper.

Magnani, A., S. Lall, and S. Boyd (2005). Tractable fitting with convex polynomials via sum-of-squares. In *IEEE Conference on Decision and Control*, pp. 1672–1677.

Markowitz, D., M. Reiman, and L. Wein (2000). The stochastic economic lot scheduling problem: heavy traffic analysis of dynamic cyclic policies. *Operations Research 48*(1), 136–154.

Matevosyan, J., M. Olsson, and L. Söder (2009). Hydropower planning coordinated with wind power in areas with congestion problems for trading on the spot and the regulating market. *Electric Power Systems Research 79*(1), 39–48.

Matevosyan, J. and L. Söder (2006). Minimization of imbalance cost trading wind power on the short-term power market. *IEEE Transactions on Power Systems 21*(3), 1396–1404.

Nedic, A. and D. P. Bertsekas (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems 13*, 79–110.

Neubarth, J., O. Woll, C. Weber, and M. Gerecht (2006). Beeinflussung der Spotmarktpreise durch Windstromerzeugung. *Energiewirtschaftliche Tagesfragen 56*(7), 42–45.

Nowak, M. P., R. Schultz, and M. Westphalen (2005). A stochastic integer programming model for incorporating day-ahead trading of electricity into hydro-thermal unit commitment. *Optimization and Engineering 6*(2), 163–176.

Park, J. and I. W. Sandberg (1993). Approximation and radial-basis-function networks. *Neural Computation 5*(2), 305–316.

Paternina-Arboleda, C. and T. Das (2005). A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. *Simulation Modelling Practice and Theory 13*(5), 389–406.

Pereira, M. V. F. and L. M. V. G. Pinto (1991). Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming 52*(2), 359–375.

Philpott, A. and Z. Guan (2008). On the convergence of stochastic dual dynamic programming and related methods. *Operations Research Letters 36*(4), 450–455.

Philpott, A. B. and V. L. de Matos (2011). Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion. Electric Power Optimization Centre, University of Auckland, New Zealand. Working paper.

Powell, W. B. (2007). *Approximate dynamic programming. Solving the curses of dimensionality.* Wiley.

Pritchard, G., A. B. Philpott, and P. J. Neame (2005). Hydroelectric reservoir optimization in a pool market. *Mathematical Programming 103*(3), 445–461.

Puterman, M. L. (2005). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley.

Qiu, J. and R. Loulou (1995). Multiproduct production/inventory control under random demands. *IEEE Transactions on Automatic Control 40*(2), 350–356.

Rockafellar, R. T. and R. J.-B. Wets (1998). *Variational analysis*, Volume 317 of *Fundamental Principles of Mathematical Sciences.* Springer.

Rummery, G. A. and M. Niranjan (1994). On-line Q-Learning using connectionist systems. Cambridge University. Technical report.

Ryzhov, I. O., P. I. Frazier, and W. B. Powell (2011). A new optimal stepsize rule for approximate value iteration. Princeton University. Working paper.

Saliby, E. (1990). Descriptive sampling: a better approach to Monte Carlo simulation. *The Journal of the Operational Research Society 41*(12), 1133–1142.

Schainker, R. B. (2004). Executive overview: energy storage options for a sustainable energy future. In *IEEE Power Engineering Society General Meeting*, pp. 2309–2314.

Schweitzer, P. (1971). Iterative solution of the functional equations of undiscounted Markov renewal programming. *Journal of Mathematical Analysis and Applications 34*(3), 495–501.

Shapiro, A. (2003). Monte Carlo sampling methods. In *Stochastic programming*, Volume 10 of *Handbooks in Operations Research and Management Science*, pp. 353–425. Elsevier.

Shapiro, A. (2011). Analysis of stochastic dual dynamic programming method. *European Journal of Operational Research 209*(1), 63–72.

Sox, C., P. Jackson, A. Bowman, and J. Muckstadt (1999). A review of the stochastic lot scheduling problem. *International Journal of Production Economics 62*(3), 181–200.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning 3*, 9–44.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Vaughan, T. (2007). Cyclical schedules vs. dynamic sequencing: Replenishment dynamics and inventory efficiency. *International Journal of Production Economics 107*(2), 518–527.

Wagner, M. and S. Smits (2004). A local search algorithm for the optimization of the stochastic economic lot scheduling problem. *International Journal of Production Economics 90*(3), 391–402.

Wen, F. and A. K. David (2000). Strategic bidding in competitive electricity markets: a literature survey. In *IEEE Power Engineering Society Summer Meeting*, pp. 2168–2173.

Winands, E., I. Adan, and G. van Houtum (2011). The stochastic economic lot scheduling problem: a survey. *European Journal of Operational Research 210*(1), 1–9.

Xu, X., D. Hu, and X. Lu (2007). Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks 18*(4), 973–992.

Zuber, M. (2011). The European market for pumped-storage power plants. Market Study. ecoprog GmbH, Cologne.

Curriculum Vitae

Personal Data

Name	Nils Löhndorf
Date of Birth	02-12-1980
Nationality	German

Academic Education

2008-2011	PhD Management program, University of Vienna, Austria.
2008-2007	PhD studies at the Graduate School of Economic and Social Sciences, University of Mannheim, Germany.
2004-2007	Graduate studies in business administration, University of Mannheim, Germany.
2002-2004	Undergraduate studies in business administration, University of Kiel, Germany.

Publications

2011 Transchel, S., S. Minner, J. Kallrath, N. Löhndorf, U. Eberhard (2011). *A hybrid general lot-sizing and scheduling formulation for a production process with a two-stage product structure.* International Journal of Production Research 49(9), 2463-2480.

Francas, D., N. Löhndorf, S. Minner (2011). *Machine and labor flexibility in manufacturing networks.* International Journal of Production Economics 131(1), 165-174.

2010 Löhndorf, N., S. Minner (2010). *Optimal day-ahead bidding with renewable energies and storage.* Energy Systems 1(1), 61-77.

Presentations

2010 Löhndorf, N., D. Wozabal, S. Minner. *Optimizing the day-ahead bidding strategy of electricity storage using approximate dynamic programming.* INFORMS Annual Meeting 2010. Austin, TX, USA. November 2010.

Löhndorf, N., D. Wozabal, S. Minner. *Optimizing the day-ahead bidding strategy of electricity storage using approximate dynamic programming.* OR for the Public Interest Conference. Stanford University GSB, CA, USA. June 2010.

2009 Löhndorf, N., S. Minner. *Optimal day-ahead bidding with renewable energies and storage.* Power Systems Modelling 2009. University of Florida, Gainesville, FL, USA. March 2009.

2008 Löhndorf, N., S. Minner, S. Transchel. *On the value of energy storage in a grid with a considerable amount of renewable energies.* 10th Workshop of the EURO Working Group on Decentralized Decision-Making. Universitat Autonoma de Barcelona, Barcelona, Spain. September 2008.

Awards

2008 Diplom thesis award of the German OR Society (GOR).